



## 저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

공학박사학위논문

Deep Neural Network Based  
Multi-Objective Dispatcher  
for Re-Entrant Manufacturing Lines

재유입 제조라인을 위한  
심층신경망 기반 다중 목적 함수 디스패처

2019 년 2 월

서울대학교 대학원  
산업공학과

허 재 석

# Deep Neural Network Based Multi-Objective Dispatcher for Re-Entrant Manufacturing Lines

재유입 제조라인을 위한  
심층신경망 기반 다중 목적 함수 디스패처

지도교수 박 종 헌

이 논문을 공학박사 학위논문으로 제출함

2018 년 12 월

서울대학교 대학원

산업공학과

허 재 석

허재석의 공학박사 학위논문을 인준함

2018 년 12 월

위 원 장      조 성 준      (인)

부위원장      박 종 헌      (인)

위 원      문 일 경      (인)

위 원      정 재 윤      (인)

위 원      김 관 호      (인)

## **Abstract**

# Deep Neural Network Based Multi-Objective Dispatcher for Re-Entrant Manufacturing Lines

Jaeseok Huh

Department of Industrial Engineering

The Graduate School

Seoul National University

A re-entrant manufacturing line (RML) is a manufacturing line in which parts make several visits to the same stage before exiting the line. RMLs have intrigued interest in both academia and industry with the recent emergence of semiconductor manufacturing and thin film transistor-liquid crystal display (LCD) manufacturing lines. As small devices embedded with flash memory and LCD have grown in demand, relevant research effort has been motivated to date.

This thesis aims to propose real-time dispatchers (RTD) based on deep neural networks (DNN) that decrease flow time without deteriorating resource utilization at the bottleneck stage for real-world RMLs. Frequent re-entrant parts between multiple stages in RMLs make it challenging to achieve the dual goals of reducing flow time and improving resource utilization. To be more specific, the level of resource utilization can be kept high by simply providing a sufficiently large amount of work-in-process (WIP) to maximize throughput. On the contrary, an excessive amount of



WIP leads to a longer waiting time for parts in the next operations, thus increasing flow time for the parts.

This thesis suggests new methods as follows. First, a discrete event based simulator (DEBS) and monitoring tool are implemented to generate training data and evaluate the performance of dispatching decisions. DEBS plays a role in imitating real-world RMLs and generating training data used for DNNs. The monitoring tool is in charge of presenting the state of an RML at the time of each dispatching decision being made. Furthermore, it also provides the ability to observe changes in various performance indicators over time.

Second, two deep neural network based RTDs with different decision-making processes are presented by the thesis. In the training phase, the proposed RTDs learn the preferences of each alternative when dispatching decisions are required according to RML data generated by the application of the developed DEBS. Then, in the real-time dispatching phase, RTDs perform dispatching decisions by considering intentional delays. A preferred alternative records a higher value as the dispatching decision is likely to reduce the part's waiting time and decrease the idle time of the resources in the bottleneck stage.

The thesis makes contributions and holds utilitarian significance in three ways. First, it developed a monitoring tool that allows users to investigate each dispatching decision. Second, the proposed approach is capable of generating training data used for DNNs by merely performing a simulation while using the developed DEBS that imitates real-world RMLs. Finally, the proposed RTDs are successful in decreasing flow time while increasing resource utilization at the bottleneck stage by factoring in intentional delays in RMLs.

**Keywords:** Re-entrant manufacturing lines, Real-time dispatcher, Intentional delay, Deep neural network, Flow time, Utilization, Discrete event based simulator, Monitoring tool

**Student Number:** 2013-23211

# Contents

<b>Abstract</b>	<b>i</b>
<b>Contents</b>	<b>vi</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Background and motivation . . . . .	1
1.2 Objectives . . . . .	6
1.3 Thesis outline . . . . .	8
<b>Chapter 2 Literature Review</b>	<b>9</b>
2.1 Dispatching decisions in RMLs . . . . .	9
2.2 Neural network-based approaches to dispatching decisions . . . . .	13
<b>Chapter 3 Problem Definition</b>	<b>17</b>
3.1 Multiple-chip product (MCP) assembly lines . . . . .	17
3.2 Lot dispatching process in MCP assembly lines . . . . .	21

<b>Chapter 4 Frameworks for Data Generation and Performance Eval-</b>	<b>24</b>
<b>uation</b>	
4.1 Discrete event-based simulator . . . . .	24
4.1.1 Purpose of implementation . . . . .	24
4.1.2 Details of the structure . . . . .	25
4.2 Monitoring tool . . . . .	30
4.2.1 Purpose of implementation . . . . .	30
4.2.2 Details of functions . . . . .	32
<b>Chapter 5 Deep Neural Network Based Dispatcher</b>	<b>43</b>
5.1 Real-time rule selection dispatcher . . . . .	46
5.1.1 Dispatcher structure . . . . .	46
5.1.2 Training phase . . . . .	49
5.1.3 Real-time dispatching phase . . . . .	52
5.2 Real-time lot selection dispatcher . . . . .	53
5.2.1 Dispatcher structure . . . . .	53
5.2.2 Training phase . . . . .	57
5.2.3 Real-time dispatching phase . . . . .	58
<b>Chapter 6 Experiments</b>	<b>60</b>
6.1 Datasets . . . . .	60
6.2 Experiment settings . . . . .	61
6.3 Experiment results . . . . .	67
6.3.1 Performance comparison . . . . .	67
6.3.2 Performance differences according to weights . . . . .	79

6.3.3 Robustness test . . . . .	87
<b>Chapter 7 Conclusions</b>	<b>92</b>
7.1 Summary and contributions . . . . .	92
7.2 Limitations and future research . . . . .	94
<b>Appendices</b>	<b>96</b>
<b>Chapter A Performance comparison results</b>	<b>97</b>
<b>Chapter B Performance contour of RTRD with respect to <math>\lambda_w</math> and <math>\lambda_l</math></b>	<b>104</b>
<b>Chapter C Performance contour of RTLD with respect to <math>\lambda_w</math>, <math>\lambda_l</math>, and <math>\lambda_d</math></b>	<b>117</b>
<b>Bibliography</b>	<b>130</b>
<b>국문초록</b>	<b>146</b>

## List of Tables

Table 2.1	Overview of previous research on dispatching decisions of RMLs.	11
Table 2.2	Overview of previous research on dispatching decisions using neural networks. . . . .	14
Table 4.1	Descriptions on the event types for the event loop. . . . .	28
Table 5.1	Components of the state vector for $R_{q,s}$ of RTRD. . . . .	48
Table 5.2	Dispatching rules used to generate the action vector for $R_{q,s}$ of RTRD. . . . .	50
Table 5.3	Components of the lot-DA assignment vector, for $L_{i,k}$ and $R_{q,s}$ .	55
Table 6.1	Descriptions on the datasets used for the experiments. . . . .	62
Table 6.2	Problem description for experiments. . . . .	63
Table 6.3	Training results of the proposed dispatchers. . . . .	66
Table 6.4	<i>ALT</i> improvement rates of RTLD compared to the existing methods and RTRD. . . . .	72
Table 6.5	Statistically significant differences in <i>ALT</i> between RTRDs trained in different datasets. . . . .	89
Table 6.6	Statistically significant differences in <i>ALT</i> between RTLDs trained in different datasets. . . . .	91

# List of Figures

Figure 1.1	Concept of re-entrant manufacturing line. . . . .	2
Figure 3.1	Lot flow of DA and WB stages in MCP production. . . . .	19
Figure 3.2	Candidate lots according to status and intentional delay. . .	21
Figure 3.3	Illustration of how the dispatcher assigns a lot to a DA resource.	23
Figure 4.1	Structure of DEBS (Discrete Event Based Simulator). . . . .	26
Figure 4.2	Main screen of the monitoring tool. . . . .	32
Figure 4.3	Resource view of the monitoring tool. . . . .	33
Figure 4.4	Decision window of the resource view. . . . .	35
Figure 4.5	Statistics view of the monitoring tool. . . . .	37
Figure 4.6	KPI view of the monitoring tool. . . . .	38
Figure 4.7	WIP charts of the monitoring tool. . . . .	40
Figure 4.8	Comparison page of the monitoring tool. . . . .	42
Figure 5.1	Overall structure of the proposed approach. . . . .	45
Figure 5.2	The structure of RTRD. . . . .	46
Figure 5.3	The structure of RTLD. . . . .	54
Figure 6.1	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 1. . . . .	68

Figure 6.2	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 2. . . . .	69
Figure 6.3	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 7. . . . .	70
Figure 6.4	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 11. . . . .	71
Figure 6.5	WIP graphs of the proposed dispatchers, and SVR. . . . .	73
Figure 6.6	Gantt charts of the proposed dispatchers and SVR. . . . .	74
Figure 6.7	Utilization graphs of the proposed dispatchers and SVR. . .	75
Figure 6.8	Dispatching frequencies according to lot statuses over time.	76
Figure 6.9	Computation time of RTRD, RTLD, and SVR according to the average number of operations. . . . .	78
Figure 6.10	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 4. . . . .	80
Figure 6.11	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 9. . . . .	81
Figure 6.12	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 10. . . . .	82
Figure 6.13	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 11. . . . .	83
Figure 6.14	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 3. . . . .	84
Figure 6.15	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 6. . . . .	85



Figure 6.16	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 9. . . . .	86
Figure 6.17	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 12. . . . .	87
Figure A.1	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 1. . . . .	97
Figure A.2	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 2. . . . .	98
Figure A.3	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 3. . . . .	98
Figure A.4	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 4. . . . .	99
Figure A.5	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 5. . . . .	99
Figure A.6	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 6. . . . .	100
Figure A.7	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 7. . . . .	100
Figure A.8	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 8. . . . .	101
Figure A.9	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 9. . . . .	101
Figure A.10	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 10. . . . .	102

Figure A.11	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 11. . . . .	102
Figure A.12	<i>AWT</i> , <i>AIT</i> , and <i>ALT</i> results of the proposed dispatchers and the existing methods for dataset 12. . . . .	103
Figure B.1	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 1. . . . .	105
Figure B.2	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 2. . . . .	106
Figure B.3	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 3. . . . .	107
Figure B.4	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 4. . . . .	108
Figure B.5	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 5. . . . .	109
Figure B.6	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 6. . . . .	110
Figure B.7	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 7. . . . .	111
Figure B.8	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 8. . . . .	112
Figure B.9	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 9. . . . .	113
Figure B.10	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 10. . . . .	114

Figure B.11	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 11. . . . .	115
Figure B.12	Performances of RTRD against SVR depending on $\lambda_w$ and $\lambda_l$ in dataset 12. . . . .	116
Figure C.1	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 1. . . . .	118
Figure C.2	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 2. . . . .	119
Figure C.3	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 3. . . . .	120
Figure C.4	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 4. . . . .	121
Figure C.5	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 5. . . . .	122
Figure C.6	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 6. . . . .	123
Figure C.7	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 7. . . . .	124
Figure C.8	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 8. . . . .	125
Figure C.9	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 9. . . . .	126
Figure C.10	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 10. . . . .	127

Figure C.11	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 11. . . . .	128
Figure C.12	Performances of RTLD against SVR depending on $\lambda_w$ , $\lambda_l$ , and $\lambda_d$ in dataset 12. . . . .	129

# Chapter 1

## Introduction

### 1.1 Background and motivation

Re-entrant manufacturing lines (RML) have become at the center of attention in both academia and industry since semiconductor manufacturing and thin film transistor-liquid crystal display (LCD) manufacturing lines came along [1, 2, 3, 4, 5]. RMLs are systems where parts can visit the same stage several times before exiting the line [6, 7, 8]. Research in this discipline has been motivated with an increasing demand for small devices equipped with flash memory and LCD such as smart phones and wearable devices [9, 10].

Figure 1.1 shows the concept of RML and the dashed line at the bottom of the figure indicates the flow of parts which go back to the previous stage. There are two types of stockers where parts stay temporarily. The first one at the top of the figure is a place where parts, which have completed an operation at the previous stage, wait before they enter the current one. The second at the bottom of the figure, called a re-entrant stocker, illustrates a place where parts, which have completed the next stage, wait for the re-entrance to the current one.

There are parallel resources that are responsible for processing parts at each stage and assigning a part to a resource is referred to as the dispatching decision.

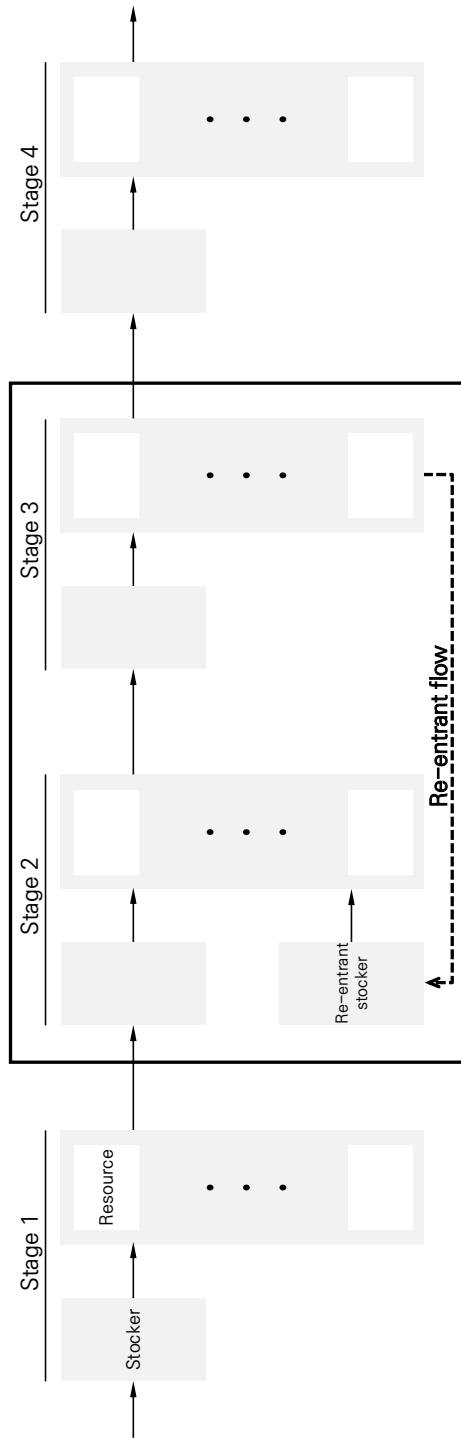


Figure 1.1: Concept of re-entrant manufacturing line.

Furthermore, a typical type of RML may regard a stage as a bottleneck if its resource utilization is above 85% [11, 12, 13]. This value is considerably subjective. In practice, it is maintained above 90% for semiconductor manufacturing lines [14]. Since the maximum throughput is determined by the bottleneck stage, it is essential to maintain a high level of resource utilization in the bottleneck [15].

It is achievable to keep the resource utilization level high by simply providing a sufficiently large amount of work-in-process (WIP). On the contrary, an excessive amount of WIP leads to increase in the waiting time for parts in the next operations, thus increasing flow time of parts [16]. Therefore, managing WIP levels properly matters in order to resolve the multi-objective problem of reducing flow time of parts and improving resource utilization [17].

Unfortunately, the re-entrant nature brings about a challenge to WIP level control [18]. Specifically, if newly arrived parts are frequently assigned to resources of the non-bottleneck stage with re-entrant parts not into consideration, the WIP level of the bottleneck stage will excessively increase. On the other hand, giving a high priority level to re-entrant parts in the non-bottleneck stage can result in a lack of WIP at the bottleneck stage, which decreases resource utilization in the bottleneck stage [19, 20].

From the remark above, it is clear that RMLs require dispatching decisions that differ from those performed in traditional manufacturing lines and flow shops [21, 22, 23]. For this reason, a number of scholars have presented methods for dispatching decisions in RMLs using optimization, meta-heuristics, and dispatching rules [19, 22, 24, 25, 26, 27, 28, 29, 30].

On one hand, approaches with a long computation time to yield dispatching de-

cisions, such as optimization and meta-heuristics, are not proper to be applied to real-world RMLs although they successfully improve objectives through an exhaustive search over solution spaces [31]. This is because, from a practical standpoint, a part has to be immediately assigned to a resource in the event that the resource requests a part.

On the other hand, the benefits of the rule based methods include computational efficiency and ease of implementation [32, 33]. However, the majority of previous dispatching rules are usually designed to address a sole objective, and have limitations when addressing various characteristics of real-world RMLs [34, 35].

In the meantime, it should be noted herein that the previous studies have focused mainly on selecting one among the waiting parts ready to be processed immediately. That is, a resource becomes idle only when there are no waiting parts in the stockers. Yet, it is well known that performance can be improved when an intentional delay is made in resource usage by idling a resource even through there are parts waiting for its processing [36, 37].

Motivated by the considerations above, this thesis attempts to suggest a dispatching method based on deep neural networks (DNNs) that decrease flow time without loss in resource utilization of the bottleneck stage for real-world RMLs. Whenever a dispatching decision is required, the proposed method choose the best part by considering both cases: when parts are processed directly; and when parts undergo an intentional delay in a resource. In other words, the proposed method maintains a high level of resource utilization and minimizes flow time by continuously prioritizing newly arrived parts and re-entrant parts according to the status of the RML.



To achieve this, we use a simulator to generate training data that are used to train the dispatcher. The main difference between the existing learning-based methods and our efforts lies in the fact that the existing work requires training data generated from optimal solutions which are difficult to obtain, while our method is capable of generating training data by simply performing simulations with random decision making. In detail, the performances of the decisions in randomly generated simulation logs are measured by the proposed score generator, and the evaluated simulation logs are used to train the DNN in the proposed dispatcher. In real-time dispatching phase, the proposed method quantifies the degree of preference for each decision with a numerical score and then completes the dispatching decision based on the score.

As mentioned above, to learn dispatching decisions considering intentional delay, we employ a DNN which is capable of capturing complex non-linear dynamics [38]. Furthermore, it is known that a DNN enables computational models to learn representations of data through multiple levels of abstraction to understand the hidden relationships among input features [39, 40]. Neural networks (NNs) have been successfully applied to a variety of areas which involve decisions, including dispatching and scheduling problems [41, 42, 43, 44, 45, 46, 47, 48].

## 1.2 Objectives

The objective of this thesis is to propose a real time dispatcher (RTD) for decreasing flow time while maintaining a high level of resource utilization in real-world RMLs. The thesis consists of two major studies to reach its goals. The first one focuses on implementing an underlying environment for conducting research. Then, the second one that deals with developing RTDs is sub-categorized into two parts according to how decisions are made. The basic concepts and purposes of the studies are summarized as follows.

First, a discrete event based simulator (DEBS) and monitoring tool are proposed in Chapter 4. DEBS is designed to imitate the RML described in Chapter 3, and calculate key performance indicators (KPIs) such as mean flow time and resource utilization. Furthermore, a DEBS takes on generating training data used to train a DNN. In addition to the functions of the existing Gantt chart, the monitoring tool presents various information on RMLs at the time of a dispatching decision being made. This feature is used to analyze the reason behind dispatching decision making, which is helpful to improving performance in RTDs.

Second, two DNN based RTDs with different decision-making processes are suggested in Chapter 5. RTDs are in charge of choosing the best part by considering both cases: when parts are processed directly; and when parts undergo an intentional delay in a resource at the non-bottleneck stage. To make that happen, we utilize the DEBS developed in Chapter 4 to generate training data used for RTDs in the training phase. In the real time dispatching phase, dispatching decisions can be made quickly thanks to the calculation of using parameters of the trained RTDs.

In detail, each alternative of the dispatching decision for a resource is represented

in the form of a vector. The proposed method quantifies the degree of preference for each vector with a numerical score, on which it completes the dispatching decision. A preferred vector receives a higher value as the dispatching decision is likely to shorten the part's waiting time and reduce the idle time of resources in the bottleneck stage.

Finally, the effectiveness and efficiency of the proposed methods are demonstrated through extensive experiments in Chapter 6. The performances of the proposed RTDs are compared with the conventional dispatching methods in terms of flow time and bottleneck resource utilization. Additionally, numerical experiments are conducted to investigate the merits and demerits of the proposed two RTDs.

### **1.3 Thesis outline**

This thesis is comprised of seven chapters and the remainder is organized as follows. In Chapter 2, previous research on the dispatching methods for RMLs is examined and DNN based techniques applied to the dispatching decision are reviewed. Chapter 3 describes the problem under consideration and defines the notations used in the thesis. The features and functions of DEBS and monitoring tool are presented in Chapter 4. The proposed approaches, consisting of two RTDs, a score generator, and learning algorithm, is introduced in Chapter 5. Subsequently, the experimental results are summarized and discussed in Chapter 6. Finally, we conclude this work with contributions and future work of this thesis in Chapter 7.

## Chapter 2

### Literature Review

#### 2.1 Dispatching decisions in RMLs

A considerable amount of literature has been published on dispatching methods in RMLs. Previous research is classified according to their approaches and performance metrics, as presented in Table 2.1.

Simulation-based studies have attempted to understand the characteristics of part flows by executing tasks virtually in advance [49, 50]. Most studies utilized simulation techniques to analyze or improve their objectives [51, 52, 53, 54]. On the other hand, researchers in [55, 56] sought simulation methods that precisely represent real-world RMLs and quickly process events.

Another line of research aims to perform dispatching decisions by utilizing dispatching rules [15, 57]. Publications on rule-based dispatching can be classified into two categories. The first group proposes methods for selecting a dispatching rule among the existing ones to obtain desired performance measures in a specific situation [14, 58, 59, 60]. To effectively cope with changes in the bottleneck stage, algorithms to detect bottleneck stage are also to be applied in [14, 60].

The second category focuses mainly on improving objectives by developing their own rule-based methods [61, 62, 63, 64]. In particular, Bard et al. [63] presented a

multi-stage approach consisting of three steps, which decides on the best resource-tooling configurations and the way to assign parts to resources. They succeeded in increasing weighted throughput in small-size problems. However, their method requires a longer computation time as problems become as large as those in the real world.

To overcome the limitations of the dispatching rules, Ma et al. [65] investigated a dynamic scheduling method based on support vector regression (SVR). Specifically, they proposed a composite dispatching rule - a linear combination of multiple dispatching rules with a weight assigned to each rule. The scheduling model trained with SVR determines the weights of the composite dispatching rule for a given production line state. Their method outperformed simple dispatching rules in terms of multiple performance measures such as flow time and resource utilization.

Besides the rule-based methods, some studies investigating meta-heuristics have been conducted to improve their particular objectives through an exhaustive search over solution spaces [26]. Genetic algorithm (GA) based methods were popularly used to decrease flow time and increase resource utilization [19, 29, 66]. In particular, the work in [29, 66] utilized the heuristic algorithm as proposed in [73] to reduce time spent on searching solution space.

Additionally, a Tabu search (TS) based algorithm, presented in [30], aims to minimize tardiness. Kang et al. [30] introduced a rolling horizon method that limits the area of unnecessary neighborhood solutions, thus decreasing computation time. Although the existing studies of meta-heuristics attempted to reduce computation time, it is difficult to introduce them into the real-world RMLs where dispatching decisions are required to be made in a real-time manner.

Table 2.1: Overview of previous research on dispatching decisions of RMLs.

<b>Approaches</b>	<b>Performance metrics</b>	<b>References</b>
Simulation	Flow time	[49], [50], [51], [52], [53]
	Simulation cost	[55], [56]
	Tardiness	[54], [52]
	Throughput	[52]
Dispatching rule	Flow time	[14], [15], [57], [58], [59]
	Tardiness	[61], [62], [60]
	Throughput	[60], [63]
	Utilization	[58], [63], [64]
Support vector regression	Flow time	[65]
	Utilization	
Meta-heuristic	Flow time	[19], [29], [66]
	Tardiness	[30]
	Throughput	[26]
	Utilization	[19], [31]
Case-based reasoning	Utilization	[34]
Mathematical programming	Flow time	[27], [67]
	Tardiness	[68], [69]
	Throughput	[28]
	Utilization	[21], [25], [28]
Reinforcement learning	Flow time	[70]
	Throughput	[23], [71], [72]

To resolve the disadvantages of meta-heuristics, such as a long computation time to obtain solutions, Lim et al. [34] extended the earlier work in [31] using case-based reasoning. Unfortunately, they failed to achieve as much resource utilization as the existing method provides [31].

Studies were conducted to analyze dispatching decisions using the mathematical formulation of part flow in RMLs [27, 68, 69]. To reduce complexity of the problem of determining dispatching decisions, the works in [21, 25, 28, 67] divided RMLs in different hierarchical layers. Furthermore, reinforcement learning based methods also have been proposed to perform dispatching decisions in RMLs [71, 72, 23, 70]. These methods are characterized in that they attempted to improve the performance of cumulative dispatching results rather than that of immediate dispatching results.



## 2.2 Neural network-based approaches to dispatching decisions

Recently, there has been a considerable interest in using NNs for dispatching decisions in various manufacturing domains [43, 74, 75]. Table 2.2 presents the summary of the previous studies on dispatching decisions with the help of NNs. They are categorized into four cases according to subject.

The field of job shops have attracted attention among many researchers, and the dispatching decision in the job shop system is traditionally known as a complex task [47, 76]. Two NNs are proposed in order to decide different dispatching rules locally for each resource [77]. Due to the structure of the developed NNs, the proposed method was not robust to the number of resources.

An attempt was made to generate training datasets for NNs by using a GA, which can obtain the optimal solution to job shop problems [44]. Although a trained NN successfully yielded performances at the closest level to those of the GA, the NN was designed solely to work for 6x6 job shop problems. Branke et al. [78] used NNs to automatically design dispatching rules in a dynamic stochastic job shop scenario. They also compared three different techniques for automated rule design: NNs, a linear combination of attributes, and a tree representation. Their numerical experiments indicated that NNs outperformed the rest for small-sized problems.

Meanwhile, a sensitivity analysis was conducted [47] in order to find which input attributes of the NN has significant impact on the performance of dispatching results. To be more specific, it measured the relative importance among the inputs of the NN and illustrated how NN output is changed in response to variations in input.

A flow shop is different from the job shop in that all jobs follow the same process-

Table 2.2: Overview of previous research on dispatching decisions using neural networks.

Subject	References	Description
Job shop	[77]	Two NNs that decide different dispatching rules locally for each resource
	[44]	Prioritizing each job using NN and GA
	[78]	Automated designing dispatching rules through network representation
	[47]	Conducting sensitivity analysis of the effectiveness of the input variables of NN
Flow shop	[79]	Adjusting the learning rate dynamically by utilizing the proposed algorithm
	[41]	Determining the weights of the NN with the developed SA
FMS	[80]	Assigning different dispatching rules to each of the resources using self-organizing map NN
	[81]	Integrating GA and NN to select the optimal subgroup of features from the state of system
RML	[59]	Selecting dispatching rules when desired performance measures are given
	[45]	Approximating the optimal value function using NN
	[33]	Dynamically determining the parameters of NN based on real-time information of the line
	[82]	Predicting the performance of the dispatching rule selected by decision tree

ing order [83]. Mouelhi-Chibani and Pierreval [41] suggested a NN based approach for assigning the most suited dispatching rule to a resource each time the resource becomes available. In their research, weights of the NN are determined with the simulated annealing (SA) method rather than using training examples.

Unlike the flow shop, a hybrid flow shop (HFS) contains at least one stage that consists of multiple resources [84]. The delta-bar-delta algorithm was developed to further speed up the convergence of the weights of NNs in the HFS [79]. This algorithm is tasked with adjusting the learning rate dynamically based on the variation of training errors.

Some authors investigated how to use NNs in a flexible manufacturing system (FMS) defined in [85]. On the one hand, Shiue and Guh [81] presented a hybrid learning framework that integrates a NN and GA to select the optimal subgroup of features from the state of the FMS. Although the performance of the framework is superior to other machine learning methods, it takes an excessively long computation time to discover the chromosomes and determine the learning parameters of the NN.

Meanwhile, Guh et al. [80] developed a method to assign different dispatching rules in each of the resources using self-organizing map (SOM) NNs. In detail, the proposed method determines appropriate multiple dispatching rules (MDRs) for a specific period. The results showed that their method outperforms two alternatives with the same dispatching rule in all resources.

Adding to this, many studies were carried out on the dispatching decisions using NNs in RMLs [33, 45, 59, 82]. To take a closer look, Min and Yih [59] proposed an approach for the selection of dispatching rules when desired performance measures are given with the status of the RML. However, they made an unsuccessful effort

to obtain high quality datasets of dispatching decisions, which has been left to be addressed.

In order to dynamically determine the parameters of a NN, Li et al. [33] devised an adaptive dispatching rule (ADR) that takes into account real-time state information of RMLs. They demonstrated that their method was superior to the existing dispatching rules by doing numerical experiments on semiconductor fabrication facilities.

A hybrid knowledge discovery framework was developed to decide the most appropriate dispatching rule using a decision tree and NN [82], which are responsible for selecting one among the existing dispatching rules, and then predicting the performance of the selected rule. In addition, Zhou et al. [45] attempted to approximate the optimal value function by using a NN. More specifically, they presented a dynamic dispatching approach for RMLs by combining dynamic programming (DP) with DNNs.

In spite of the fact that previous research successfully addresses the use of NNs for dispatching decisions on various manufacturing systems, most conventional methods can learn dispatching strategies only if training datasets are obtained from optimal solutions. This only implies inefficiency as their methods necessitate a solver that yields optimal solutions for given problems. Furthermore, as mentioned above, they do not factor in intentional delay decisions, which can possibly improve performance if they are done properly.

## Chapter 3

### Problem Definition

#### 3.1 Multiple-chip product (MCP) assembly lines

We consider a multiple-chip products (MCPs) assembly line for semiconductor manufacturing which is the most representative one of RMLs [17]. MCP production involves complex and correlated assembly stages consisting of backlap, wafer sawing, die attach (DA), wire bonding (WB), and molding [29, 61]. Especially, in the DA and WB stages, wafers are grouped as a lot and processed by a resource. Here, assigning a lot to a resource is referred to as the lot dispatching decision.

For producing the large capacity MCPs, frequently re-entrant lots between the DA and WB stages are necessary to assemble multiple chips into one single packaging module [61]. In particular, the capacity of MCP tends to be proportional to the number of visits to these stages [31]. The WB stage is usually considered as a bottleneck compared to the DA stage due to its extremely long processing time for soldering a number of wires to each die [86]. To efficiently operate assembly lines, maintaining high utilization of resources in the WB stage is essential.

To manage the WIP level, in an attempt to decrease the flow time without loss in resource utilization of the bottleneck stage, in this thesis, we focus on controlling the lot flow in the DA stage. This is because the lot dispatching decision in a non-

bottleneck stage has a significant impact on the WIP level of an assembly line [54, 86]. Furthermore, the utilization of the DA stage is not necessary to be kept high if that of the WB stage does not decrease because the throughput of the assembly line is determined by the bottleneck stage [15].

For lot dispatching in the bottleneck stage, a higher utilization rate of resources can be achieved simply by processing lots primarily with longer processing time [86, 87]. Therefore, the lot dispatching decisions of the WB stage in this thesis are carried out by using the rule that assigns a high priority level to the lot which has the longest processing time for a resource.

We are given a set of resource types,  $M = \{M_q | q = 1, \dots, N_M\}$ , where  $M_q$  is associated with  $n_q$  resources,  $R_{q,1}, \dots, R_{q,n_q}$ . For each operation, its available resources and processing time are determined according to the resource type. There is a set of job types,  $J = \{J_i | i = 1, \dots, N_J\}$ , where  $J_i$  consists of a sequence of operations specified in a predetermined order. We represent the  $j^{th}$  operation of  $J_i$  as  $O_{i,j}$ , and  $A(O_{i,j})$  indicates a set of resource types capable of processing  $O_{i,j}$ . The  $k^{th}$  lot for  $J_i$  is denoted as  $L_{i,k}$ ,  $k = 1, \dots, n_i$ , where  $n_i$  is the number of lots of type  $J_i$ . Thus,  $L_{i,k}$  is processed according to the operation sequence corresponding to  $J_i$ , and  $I(L_{i,k})$  returns the smallest index among those of the operations waiting to be processed. Additionally, the processing time of a lot is to be proportional to the number of chips in the lot.

Fig. 3.1 illustrates the lot flow of the MCP production process considered in this thesis. Specifically, a lot is required to be processed in the DA stage prior to the WB stage, and the final operation of a lot is to complete in the WB stage. The dashed line at the bottom represents the flow of lots which revisit the DA stage

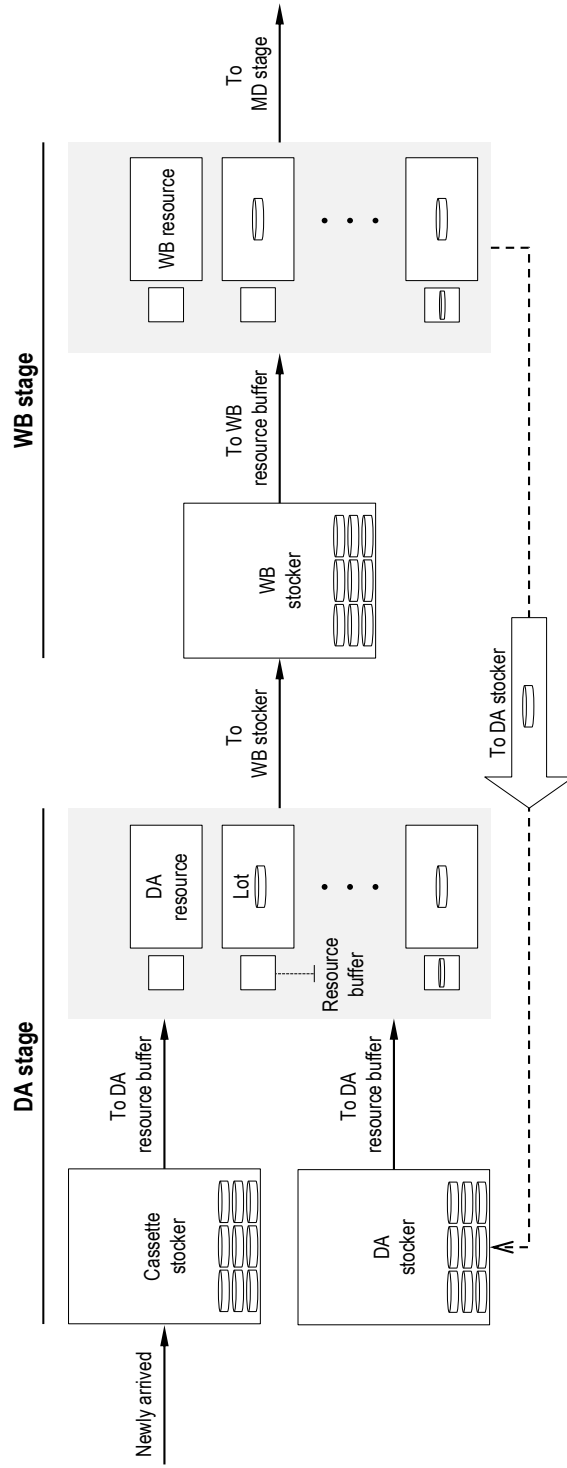


Figure 3.1: Lot flow of DA and WB stages in MCP production.

after finishing the WB operation.

As shown in Fig. 3.1, there are three types of stockers, namely cassette, DA, and WB stockers where lots stay temporarily. First, the cassette stocker provides locations for where newly arrived lots wait for the first DA operation. Next, the DA stocker is a re-entrant stocker where lots that have completed a WB operation wait for the re-entrance to the next DA operation. Finally, the WB stocker is where lots that have completed a DA operation wait before they enter their WB operation.

Lots in either the cassette or DA stockers are transported to the WB stocker after DA operations are finished. This means that newly arrived and re-entrant lots are located together in the WB stocker, which yields complex lot flows. For this reason, it becomes challenging to manage the WIP level of the WB stocker at appropriate level, which is highly likely to decrease the utilization of the WB resources or increase the waiting time of lots in the WB stocker.

In front of each resource, there is a resource buffer in which a lot waits for the operation until the resource becomes idle. The capacity of a resource buffer is assumed to be one. A lot is not interrupted once its operation starts, and an operation is carried out by one resource at a time. Additionally, it is assumed that there is no setup time between lots of different job types.

Regarding MCP assembly lines under the characteristics described above, we aim to minimize the waiting time of lots and the idle time of WB resources in order to reduce the flow time while maintaining high utilization of the bottleneck stage. This is due to the fact that the flow time consists of processing time, moving time, and waiting time. Since the processing and moving time are necessary to complete all operations of a lot, the reduction in flow time is mainly achievable by decreasing



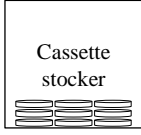

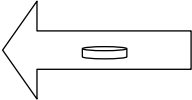

Candidate lots				
Status	In-Cassette-Stocker	In-DA-Stocker	To-DA-Stocker	At-WB-Resource
				
Intentional delay	None		Less	More

Figure 3.2: Candidate lots according to status and intentional delay.

the waiting time. Furthermore, the average utilization rate of the resources increases as resources perform operations with shorter idle periods [17].

### 3.2 Lot dispatching process in MCP assembly lines

A candidate lot refers to one that is assignable to a DA resource when its resource buffer is empty. The types of a candidate lot according to its status are illustrated in Fig. 3.2. A lot dispatching method determines the assignment between a candidate lot,  $L_{i,k}$ , and a DA resource with an empty resource buffer,  $R_{q,s}$ , based on the decision policy if  $M_q \in A(O_{i,I(L_{i,k})})$ . Furthermore, once a lot is dispatched, it is excluded from the candidate lots.

For a DA resource, a lot can be moved from the stocker to the DA resource buffer immediately whenever a candidate lot whose status is **In-Cassette-Stocker** or **In-DA-Stocker** is selected to be dispatched. Otherwise, in case that a candidate lot whose status is either **To-DA-Stocker** or **At-WB-Resource** is selected, this results in an intentional delay on the DA resource due to the additional time to carry out

the remaining WB operation and/or to arrive at a DA stocker. The details of how the dispatcher assigns a lot to a DA resource are described in Figure 3.3. The bottom part of the step 3 shows the time for each lot to arrive the DA resource after the lot is selected by the dispatcher.

In particular, the flow time of a lot begins to be measured when the lot in the cassette stocker is dispatched. This is a well-known practice in MCP assembly lines where the product type of each lot is determined when the first operation of the lot is performed. In other words, the time lots spend in the cassette stocker is not the interest in terms of WIP management.

In the WB stage, intentional delays are not necessary since high utilization of resources should be achieved. Accordingly, among the lots in the WB stocker, the lot with longest processing time is assigned to a WB resource when its resource buffer is empty.

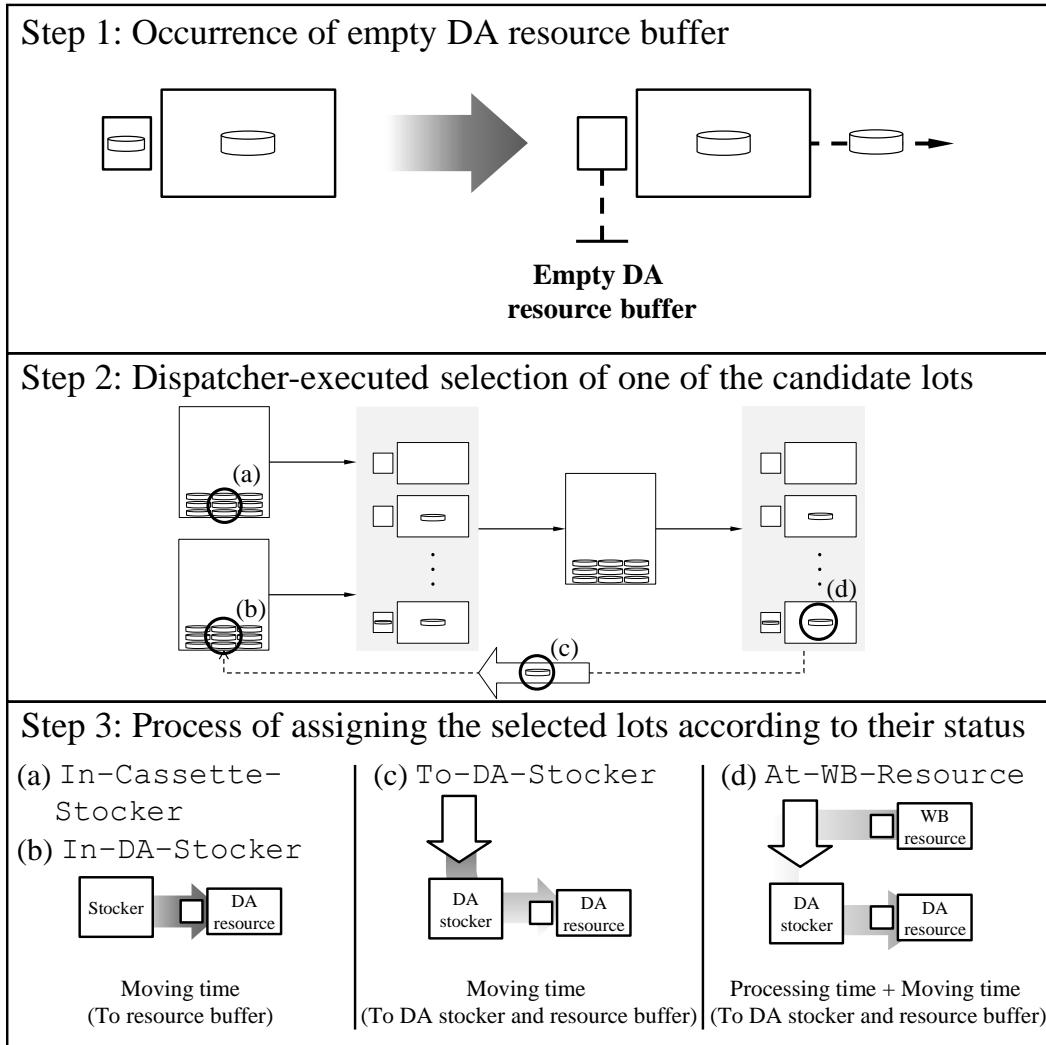


Figure 3.3: Illustration of how the dispatcher assigns a lot to a DA resource.

## Chapter 4

# Frameworks for Data Generation and Performance Evaluation

In this chapter, we present a DEBS and monitoring tool which are essential for generating training data and evaluating the performance of dispatching decisions. Python and JavaScript are adopted to implement DEBS and the monitoring tool, respectively, as their programming language.

### 4.1 Discrete event-based simulator

#### 4.1.1 Purpose of implementation

A discrete event simulation (DES) is a widely used approach to analyzing and understanding the dynamics of manufacturing lines. It is a highly flexible tool that can evaluate different alternatives of system configurations and operation strategies to determine decisions in the manufacturing lines [88]. Under this rational, the purpose of implementing DEBS directly in this thesis can be summarized in three ways.

The first purpose is to simulate real-world RMLs where practical constraints and various types of events exist. A DEBS is required to perform dispatching decisions in the situation where resource types capable of processing an operation are determined. The resource buffer capacity and the physical path of the lot should also be

considered. Furthermore, implementation is carried out on an event with an empty resource buffer as well as all events that change the state of the manufacturing line.

More importantly, the second aim is to generate training data used for DNNs. When a lot dispatching decision is required, a DEBS has to extract desired information from various parameters representing the status of RMLs. After all simulations are completed, the extracted information is printed in the form of a vector suitable for the input layer of the DNN.

Finally, KPIs such as waiting time, idle time, WIP level and resource utilization are calculated by a DEBS. The KPI calculations are used to evaluate dispatching results and compare the proposed method with the existing methods. In addition, a DEBS is in charge of generating a text file used as input data of the monitoring tool. The text file contains not only the information needed to illustrate the Gantt chart but also the information captured at the time each dispatching decision being made.

#### **4.1.2 Details of the structure**

Fig. 4.1 depicts the overall process of the proposed DEBS, which is a simulator where state changes happen exclusively in discrete instances in time. In other words, not any change in state occurs by the DEBS between two consecutive events [89]. The description of each component of the DEBS is as follows.

##### **Input**

The input of the DEBS includes: (a) resource-related information that contains resource types, the number of resources, and operations that each resource can process;

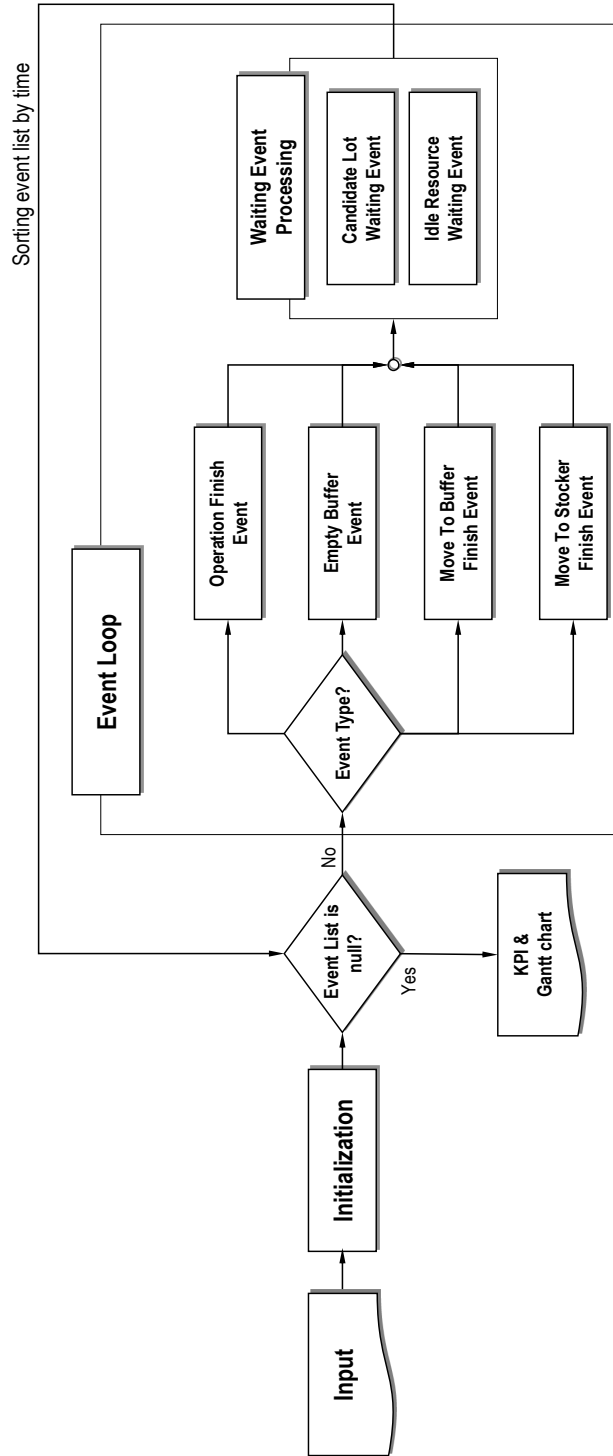


Figure 4.1: Structure of DEBS (Discrete Event Based Simulator).

(b) job-related information, including job types, the operation sequence of each job type, the number of lots assigned to each job type, and the number of chips in each lot; and (c) physical constraints which determine processing and moving time for lots, the capacity of a resource buffer, and whether pre-emption is allowed.

## **Initialization**

Based on the input, the initialization step builds a set of variables for representing the status of a RML and calculating the performance of dispatching decisions. These variables are used to generate vectors for a DNN when a lot dispatching decision is required. Furthermore, this step creates an event list and data structures for the monitoring tool.

Another important role of the initialization step to play is to insert initial events into the event list. As all resource buffers of the RML are empty, multiple lot dispatching decisions are required in this step. After lot dispatching decisions are determined, each event corresponding to each decision is generated and added to the event list.

## **Event loop**

An event contains information of the lot and resource related to the event, and timestamp. The timestamp indicates the time when the state changes due to the processing of the event. In addition, the generation of an event leads to the determination of its type. There are six types of events as shown in Fig. 4.1.

An event loop is a loop that is terminated when all events on the event list are exhausted. This loop repeatedly consumes and processes events on the list by

Table 4.1: Descriptions on the event types for the event loop.

Event	Description	Generated event
Operation-Finish	Event triggered when a resource finishes processing a lot	Move-To-Stocker-Finish
Empty-Buffer	Event triggered when a resource buffer is empty	Move-To-Buffer-Finish
Move-To-Buffer-Finish	Event triggered when a lot arrives at a resource buffer	Empty-Buffer, Operation-Finish
Move-To-Stocker-Finish	Event triggered when a lot arrives at a DA or WB stocker	-
Candidate-Lot-Waiting	Event waiting for candidate lots for a resource with an empty resource buffer	Move-To-Buffer-Finish
Idle-Resource-Waiting	Event waiting for a resource to become idle to process the lot in the resource buffer	Empty-Buffer, Operation-Finish

ascending order of timestamp among the rest. This process executes some code of the DEBS to materialize an appropriate change in state, which is likely to result in the generation of a new event. The details of events are summarized in Table 4.1.

If an **Operation-Finish** event is selected on the event list, a resource finishes processing a lot and the status of the resource becomes idle. If the lot has remaining operations, a DEBS sends the lot to the stocker for the next operation and generates a **Move-To-Stocker-Finish** event.

An **Empty-Buffer** event is an event requiring a lot dispatching decision. When this type of event is triggered, a DEBS creates a list of candidate lots for the resource with an empty resource buffer. If there is no candidate lot for the resource, the DEBS adds a **Candidate-Lot-Waiting** event whose timestamp is  $\infty$  to the event list. Otherwise, a lot dispatching decision is made via a lot dispatching method.



Once a lot is assigned to an empty resource buffer as a result of the lot dispatching decision, a **Move-To-Buffer-Finish** event is generated.

When a **Move-To-Buffer-Finish** event is selected on the event list, a lot arrives at a resource buffer. If the resource is processing the other lot, a DEBS adds an **Idle-Resource-Waiting** event whose timestamp is  $\infty$  to the event list. Otherwise, the resource starts to process the lot in the resource buffer, which causes **Empty-Buffer** and **Operation-Finish** events to be included on the event list.

A **Move-To-Stocker-Finish** event is triggered when a lot with remaining operations arrives at the stocker for the next operation. This event is different from other types of events in that it changes the state but does not generate a new event.

Since the timestamp of waiting events is  $\infty$ , **Candidate-Lot-Waiting** and **Idle-Resource-Waiting** events are not able to be selected in the event loop. Therefore, for each iteration of the event loop, a DEBS performs waiting event processing after any event with a finite timestamp is selected among the four events.

In the waiting event processing, for **Candidate-Lot-Waiting** events, a DEBS searches for candidate lots of the resource associated with each **Candidate-Lot-Waiting** event. If there is a candidate lot of any resource, the waiting event of the resource is removed from the event list. Then, a lot dispatching decision is made by a lot dispatching method, and a **Move-To-Buffer-Finish** event is added to the event list.

Meanwhile, a DEBS checks out the idleness of each resource related to **Idle-Resource-Waiting** events. If a resource is idle, the lot in the resource buffer starts to be processed by the resource. Consequently, **Empty-Buffer** and **Operation-Finish** events are to be added to the event list. Then, the **Idle-Resource-Waiting** event

corresponding to that resource is removed from the event list.

## **KPI and Gantt chart**

Once the event list is exhausted, KPIs such as waiting time, idle time, and resource utilization are calculated by a DEBS. Furthermore, the DEBS yields the simulation log containing the entire dispatching history, which is used to generate training datasets for NNs.

For the monitoring tool, DEBS also writes a text file in JSON (JavaScript Object Notation) format, which can be easily parsed in different programming languages [90, 91]. Based on the text file, the monitoring tool illustrates a Gantt chart for the sequence of dispatched lots of each resource. In addition, the text file contains the information of the RML at the time each lot dispatching decision being made.

## **4.2 Monitoring tool**

### **4.2.1 Purpose of implementation**

Gantt chart is known as a basic schedule representation tool that displays each resource’s operational status by changing the color or position of bars [92, 93]. A few studies attempted to extend the basic Gantt chart. Jo et al. [94] proposed a framework containing algorithms to explore the schedule of large-scale manufacturing lines. An interface was also proposed by [95] to deal with specific disruptions in resources and the historical analysis of manufacturing line performance.

Although the previous research successfully improved the basic Gantt chart, their framework has limitations in performing the analysis of dispatching decision units. Therefore, we propose a novel monitoring tool with additional functions needed

to monitor KPIs and analyze RTD performances. The purpose of developing the monitoring tool is presented as follows.

One of the most important purposes is to display a variety of information about RMLs at the time each lot dispatching decision being made. The monitoring tool is required to present figures including all alternatives, the amount of WIP, the number of resources in operation. Through examining these values, a user can analyze the reason why the lot dispatching decisions were conducted. The results of the analysis are based on improving the learning framework or the decision making method of the proposed dispatchers.

The second purpose is to illustrate how various performance indicators change over time. Since dispatching decisions occur sequentially, it is important to observe indicators that change in value according to the decisions made. Therefore, the proposed monitoring tool is essential to presenting time-dependent changes in indicators such as the amount of WIP, resource utilization, and the number of lots that complete all operations.

Finally, the monitoring tool is required to be designed to compare multiple dispatching results. As there are a few indicators that represent the performance of dispatching decisions, it is challenging to simply compare and analyze multiple dispatching results. Accordingly, the proposed function provides the ability to compare the performances of multiple RTDs with DNNs learned under different parameters in a single screen.

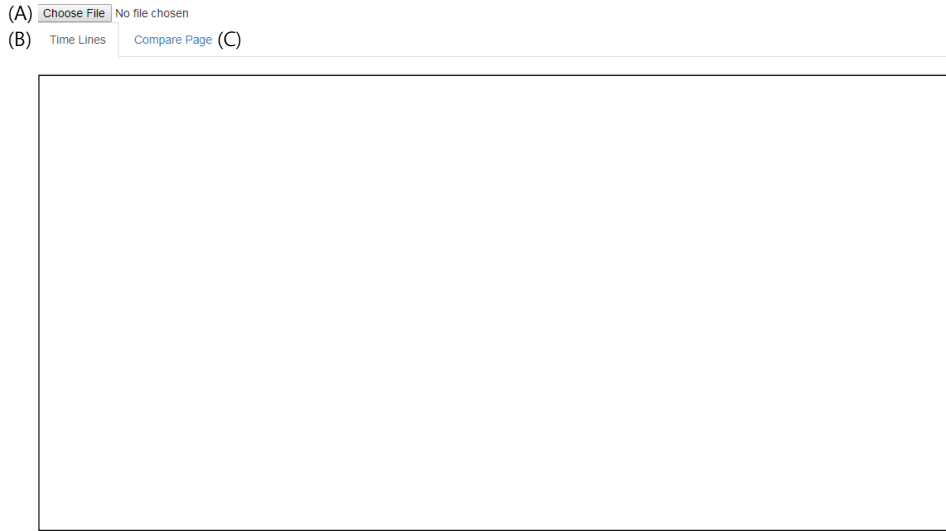


Figure 4.2: Main screen of the monitoring tool.

#### 4.2.2 Details of functions

The main screen of the monitoring tool is presented in Fig. 4.2. A user can load a DEBS-generated text file by pushing the ‘Choose file’ button on the upper left side (A). When a text file is loaded, the ‘Time Lines’ tab (B) displays several views in the bottom box. The ‘Compare Page’ (C) shows the performances of multiple dispatching results on a page when text files are additionally loaded.

##### Resource view

Fig. 4.3 illustrates the resource view when a text file is loaded with the ‘Choose file’ button pushed. This view differs from the existing Gantt chart. The idle periods caused by intentional delays are shown differently from those by the lack of WIP. The former is visualized in the form of a rectangle filled with diagonal stripe patterns;

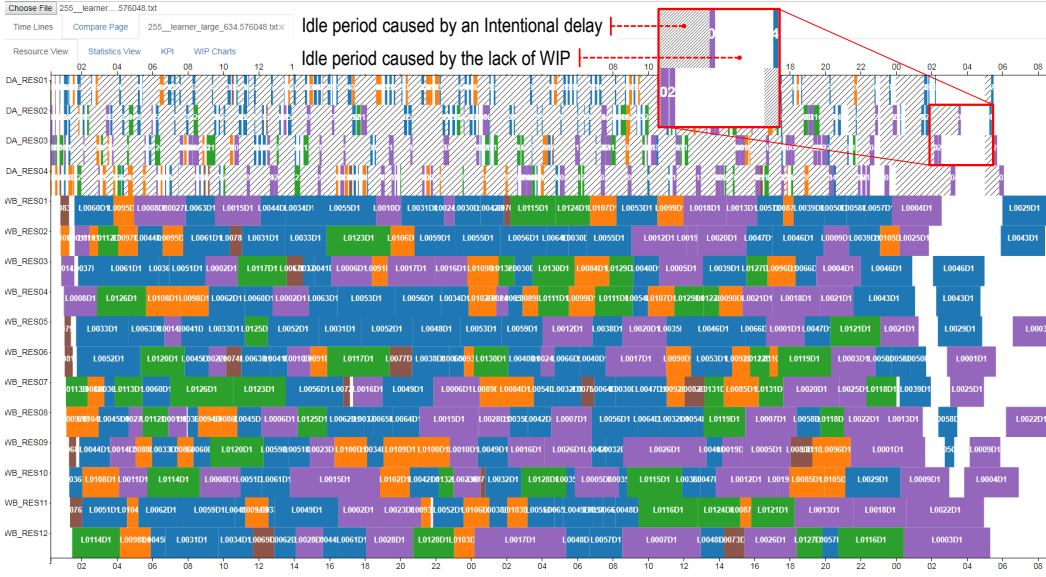


Figure 4.3: Resource view of the monitoring tool.

while the latter is shown in the shape of simple white rectangles.

If a user selects any operation in the resource view, the screen switches to the one shown in the Fig. 4.4. The selected operation is highlighted and surrounded by a red line (A), and only the operations that belong to the lot corresponding to the selected operation are displayed on the screen. The vertical red line (B) signifies the decision time - the time at which the lot dispatching decision corresponding to the selected operation was made.

Furthermore, a pop-up window shows up, elaborating the properties of the lot and resource involved in the lot dispatching decision and the status of the RML at the decision time. Specifically, the first column (C) shows lot-related information such as the number of chips, operation type and the start and end time of the operation in the resource. The job and resource types of the lot and resource involved in the

lot dispatching decision are presented in the second column (D). The last column (E) displays RML-related properties including the amount of WIP, the number of resources in operation, and the number of lots that departed the cassette stocker at the decision time.

Finally, the table at the bottom (F) represents alternatives, including the one chosen as a result of the dispatching decision at the decision time. For each alternative, lot-related information and the scores marked in terms of waiting and idle time are presented. In detail, lot-related information includes the operation type of the lot and the location where the lot stays at the decision time, and the job type of the lot.

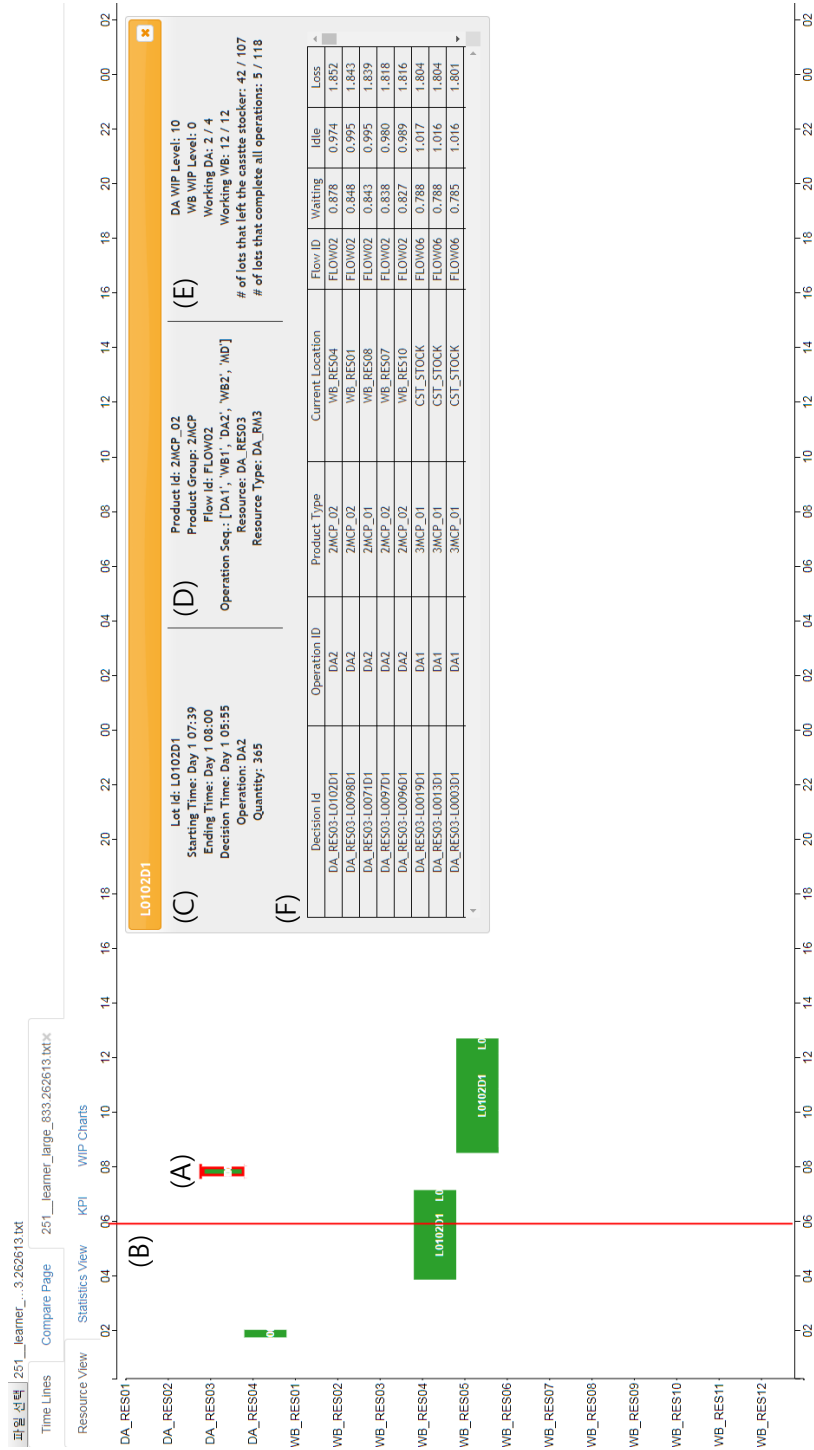


Figure 4.4: Decision window of the resource view.

## Statistics view

The statistics view illustrates how the four performance indices change over time as shown in Fig. 4.5. The upper left graph (A) shows the amount of WIP for DA and WB stages, which means the numbers of lots in the DA and WB stockers, respectively. The number of lots that departed the cassette stocker per day for each job type is visualized in the upper right graph (B).

The bottom left graph (C) with the increasing trends indicates the number of lots with all operations complete. Finally, the graph on the bottom right represents the resource utilization at each stage. The values in the graph mean the number of resources processing a lot for each stage that is divided by the total number of resources in the corresponding stage.



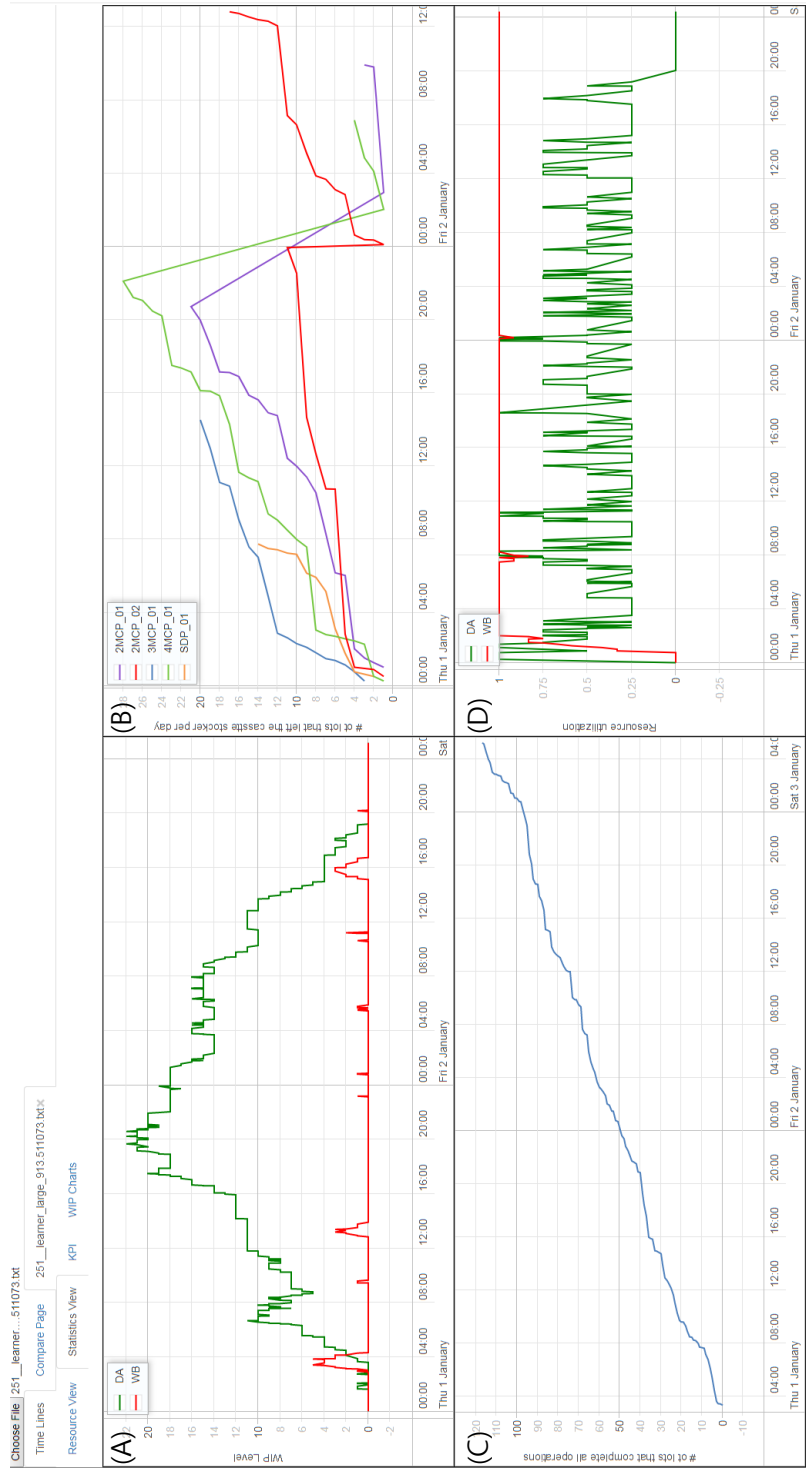


Figure 4.5: Statistics view of the monitoring tool.

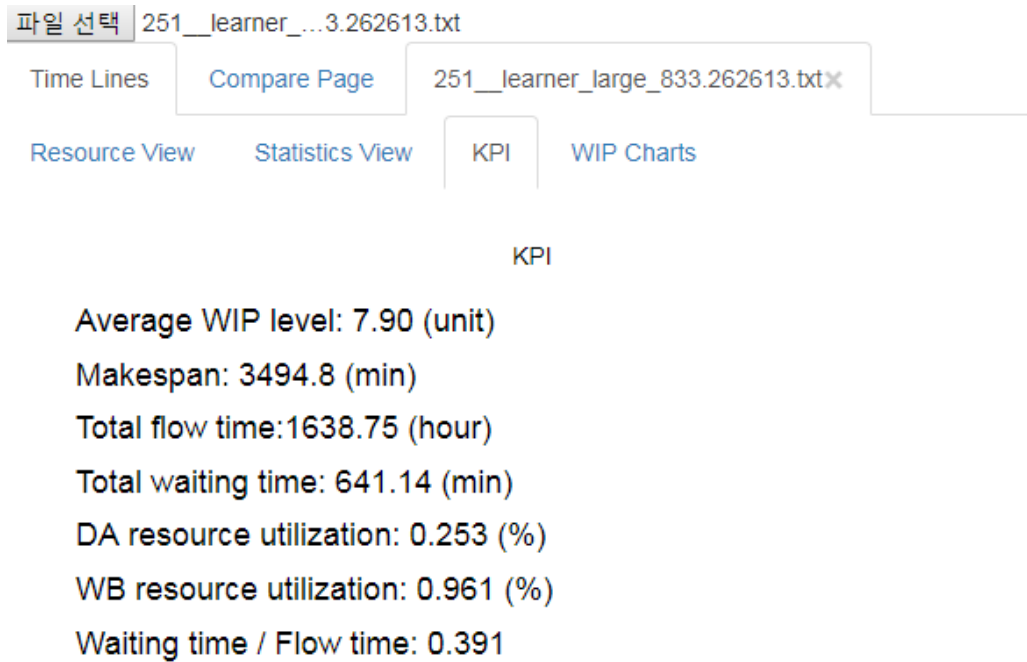


Figure 4.6: KPI view of the monitoring tool.

### KPI view

As shown in Fig. 4.6, the KPI view shows the performances yielded after all dispatching decisions are completed. Unlike other views of the monitoring tool, this function shows the overall performance rather than time-based changes or information on dispatching decision units.

To be more specific, ‘Average WIP level’ is calculated as the average of the number of WIPs that are recorded every time that of WIPs changes. ‘Total flow time’ indicates the sum of the flow time of all lots while ‘Total waiting time’ means the sum of the waiting time of all lots. Here, waiting time is the sum of the time

that a lot waits in stockers or resource buffers from the start of the first operation to the completion of all operations.

The average utilization rate of resources is calculated separately for each stage. ‘DA resource utilization’ is calculated as the average of the utilization of DA resources. The utilization of each resource is obtained by dividing the total processing time of the operations assigned to the resource by the dispatching horizon which represents a time period for which the lots necessary for satisfying production requirements are dispatched. Similarly, the value of ‘WB resource utilization’ is also obtained.

## **WIP charts**

Fig. 4.7 visualizes the amount of WIP in more detail than the statistics view does. In WIP charts, the number of graphs represented is twice that of job types which appear on the assembly line. Specifically, a row is generated to illustrate how the amount of WIP for each job type changes over time. The row is divided into two columns, each of which contains a graph.

A graph in the left column (A) shows the amount of WIP for DA and WB stages, which means the number of lots belonging to one job type in the DA and WB stockers, respectively. On the other hand, a graph in the right column (B) indicates the amount of WIP for each operation type, which means the number of lots belonging to the corresponding operation type in stockers.

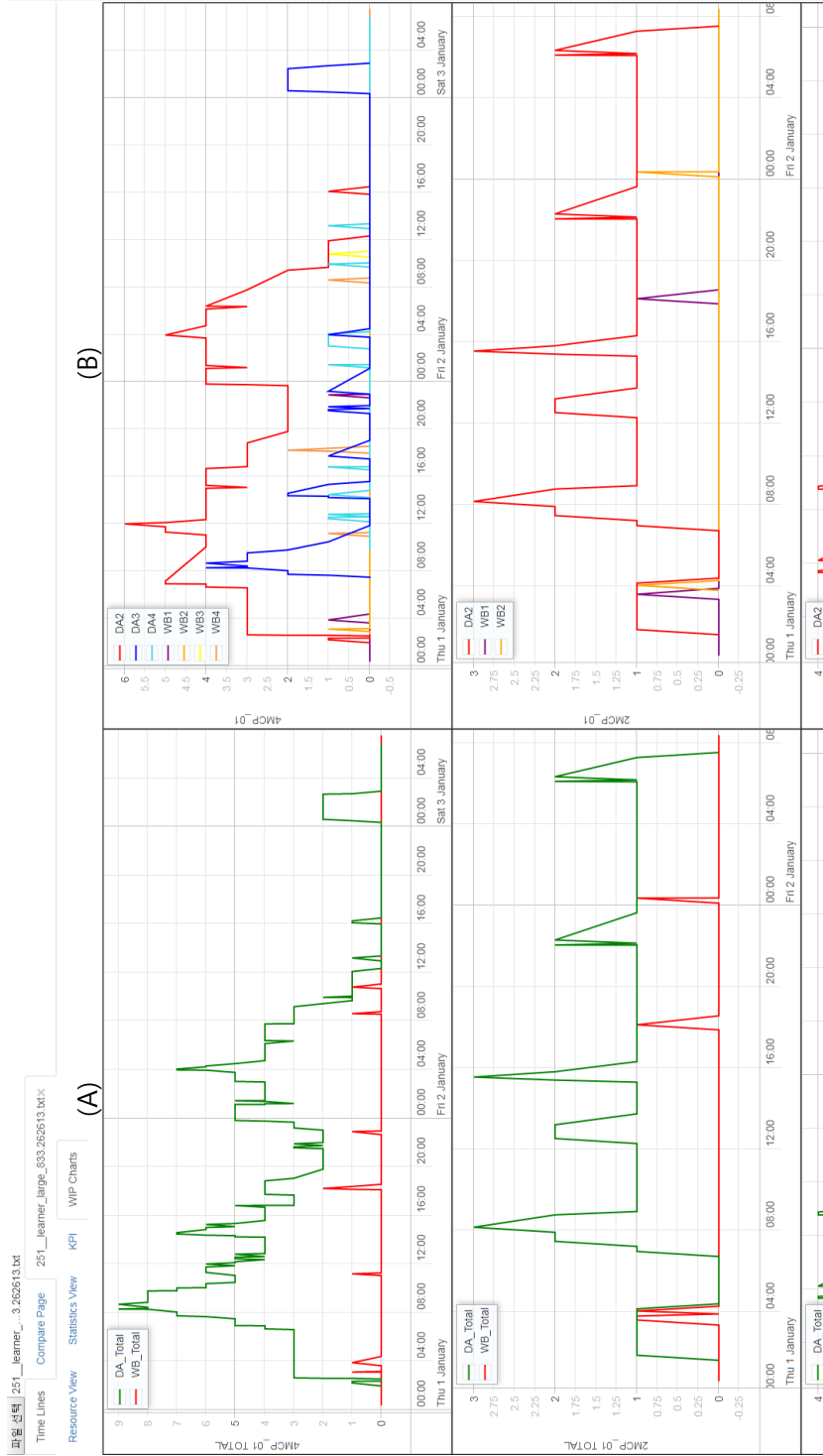


Figure 4.7: WIP charts of the monitoring tool.

## **Compare page**

The ‘Compare page’ is activated when multiple text files are loaded, allowing a user to compare multiple dispatching results on a single screen. As shown in Fig. 4.8, this page lists KPI and statistics views for each text file in the vertical direction, and a column in the same format is added whenever text files are additionally loaded.



## Chapter 5

### Deep Neural Network Based Dispatcher

Fig. 5.1 depicts the overall process of the proposed approach. In the training phase, we deploy a simulator that executes the DA and WB stages in MCP production as shown in Fig. 3.1 to generate simulation logs. By utilizing the simulator, all lot dispatching decisions of a problem are determined using a random decision generator (RDG) which is responsible for randomly assigning one of the candidate lots to a DA resource with an empty buffer. The performances of the decisions by RDG are then measured, and the scored simulation logs will be used by a learning algorithm to train the DNNs embedded in RTDs. In the real-time dispatching phase, for a given test problem, the simulator calls the trained dispatcher whenever a lot dispatching decision is required.

In this thesis, we propose two DNN based RTDs with different decision-making processes. The first one is a real-time rule selection dispatcher (RTRD). It determines the best dispatching rule among the existing ones at the time when a lot dispatching decision is required. According to the determined rule, one of the candidate lots is assigned to a DA resource with an empty resource buffer. Secondly, a real-time lot selection dispatcher (RTLTD) marks scores for all candidate lots and assigns the lot with the highest score to the DA resource. The details of RTDs are described in the

following sections.



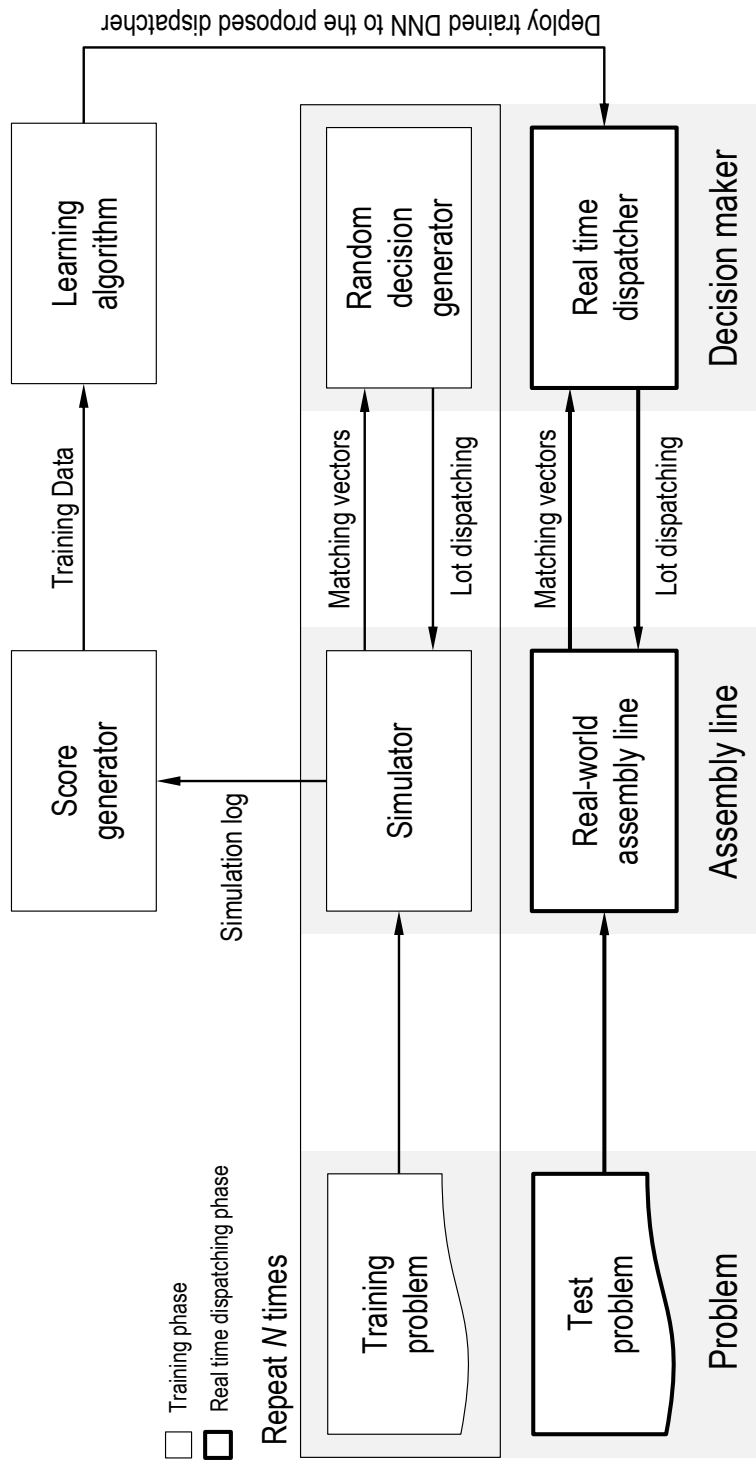


Figure 5.1: Overall structure of the proposed approach.

## 5.1 Real-time rule selection dispatcher

### 5.1.1 Dispatcher structure

Fig. 5.2 illustrates the architecture of RTRD which consists of five layers: an input layer, three hidden layers, and an output layer. The input layer contains 37 nodes, and the number of nodes in each hidden layer are 64, 32, and 16, respectively, whereas the output layer has one node. The numbers of hidden layers and nodes in the hidden layers are empirically determined to reduce the training error. The rectified linear unit (ReLU),  $f(z) = \max(0, z)$ , is applied before each hidden layer in order to provide a non-linear transformation, and all layers are fully connected [96].

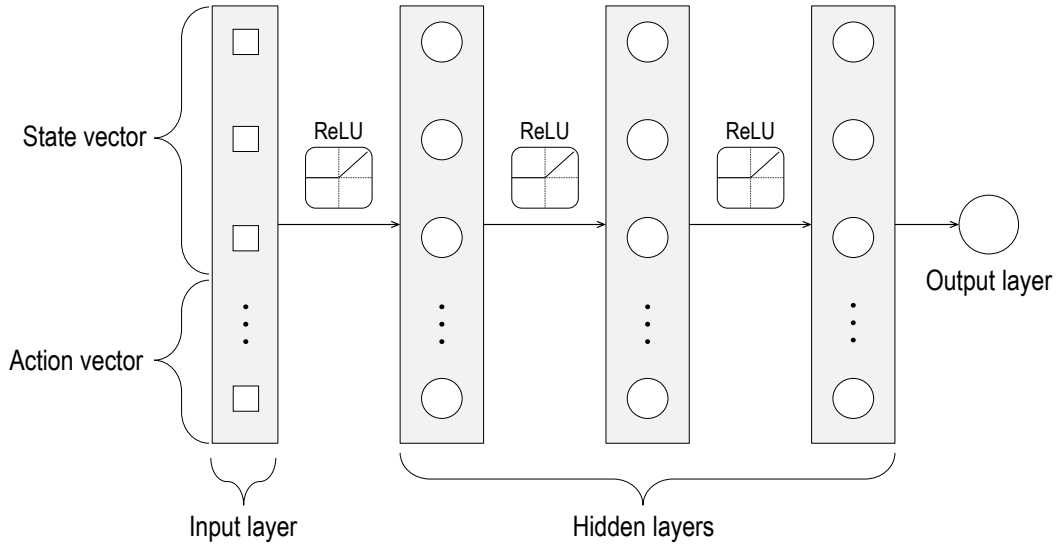


Figure 5.2: The structure of RTRD.

The input layer can be divided into two groups: state and action vectors. The former describes the status of an MCP assembly line when a particular DA resource,

$R_{q,s}$ , has an empty resource buffer, while the latter indicates the dispatching rule used to select the lot to be assigned to  $R_{q,s}$ . We propose the components of the state vector, and then introduce dispatching rules for the action vector.

Table 5.1 presents the details of the state vector for  $R_{q,s}$ . From the state vector, a RTRD is capable of capturing the characteristics of candidate lots and the flow of lots in the assembly line. This is the basis on which the RTRD determines dispatching rules for  $R_{q,s}$ . Specifically, the state vector contains 26 features that are categorized into DA resource, statistics of candidate lots, and the distribution of lots as shown in Table 5.1. First, a single feature associated with  $R_{q,s}$  indicates how much time remains until  $R_{q,s}$  becomes idle.

Next, the 12 features associated with candidate lots represent the minimum, maximum, and average values of the four attributes that each lot has. The four attributes are: the time when the lot departed the cassette stocker, the number of chips in the lot, the number of the remaining operations to be processed, and the processing time on  $R_{q,s}$ . Other attributes of the lot were excluded since they are not related to the performance measures that are considered in this thesis. RTRD utilizes the statistics of all candidate lots rather than the characteristics of each one because RTRD is not able to specify the lot to be assigned to  $R_{q,s}$  until the dispatching rule is determined, which differentiates RTRD from RTLD.

Finally, each feature belonging to the lot distribution includes the number of lots that correspond to one of the 13 different locations, covering all areas where a lot can exist in assembly lines. With these features, the proposed dispatcher knows not only the distribution of lots in the assembly line, but also the progress of the overall processes.

Table 5.1: Components of the state vector for  $R_{q,s}$  of RTRD.

Categories	Descriptions
DA resource	The time remaining until $R_{q,s}$ becomes idle
Statistics of candidate lots	Earliest time when lots depart the cassette stocker
	Latest time when lots depart the cassette stocker
	Average time when lots depart the cassette stocker
	Minimum number of chips in lots
	Maximum number of chips in lots
	Average number of chips in lots
	Minimum number of remaining operations assigned to lots
	Maximum number of remaining operations assigned to lots
	Average number of remaining operations assigned to lots
	Minimum processing time of lots on $R_{q,s}$
	Maximum processing time of lots on $R_{q,s}$
	Average processing time of lots on $R_{q,s}$
Lot distribution	# of lots in the cassette stocker
	# of lots being moved from the cassette stocker to DA resource buffers
	# of lots in DA resource buffers
	# of lots being processed on DA resources
	# of lots being moved from WB stocker to WB resource buffers
	# of lots in WB resource buffers
	# of lots being processed on WB resources
	# of lots being moved from WB resources to the DA stocker
	# of lots in the DA stocker
	# of lots being moved from the DA stocker to DA resource buffers
	# of lots being moved from DA resources to the WB stocker
	# of lots in the WB stocker
	# of lots that completed all operations

Table 5.2 summarizes dispatching rules used to generate the action vector for  $R_{q,s}$ , and the manners in which each rule chooses one of candidate lots. Each rule was modified to be used in the assembly line considered. Among the existing dispatching rules, we chose the ones that affect the waiting time of lots and the utilization of resource [62, 97]. STOCKER is a rule developed in this thesis according to the assembly line considered herein.

When a lot dispatching decision is required, an 11-dimensional action vector with zero values of all elements is generated. Then, if a specific dispatching rule is selected, the value of 1 is assigned to the element corresponding to that rule in the action vector. For example, in the case of MOR, the form of the action vector is  $(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$  while LPT generates  $(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$ .

The output layer represents the preference score of the input vector. All values in each node are normalized to a range  $[0, 1]$  using the min-max normalization to accommodate the inconsistencies of different units [98].

### 5.1.2 Training phase

In the training phase, each generated problem is solved multiple times by using RDG which is responsible for randomly selecting one of dispatching rules presented in Table 5.2. Once all lot dispatching decisions of each problem are determined by RDG, the generated simulation log containing the entire dispatching history each of which consists of a dispatched lot and an assigned DA resource, and their input vector is sent to the score generator.

The score generator evaluates each lot dispatching decision based on the waiting time of the dispatched lot and the idle time of the WB resource that processed the

Table 5.2: Dispatching rules used to generate the action vector for  $R_{q,s}$  of RTRD.

Rules	How to select one among candidate lots
FIFO (First in, first out)	The oldest lot that has been dispatched from the cassette stocker
LOR (Least operation remaining)	The lot with the smallest number of successor operations
MOR (Most operation remaining)	The lot with the largest number of successor operations
SMALL	The lot with the smallest number of chips
LARGE	The lot with the largest number of chips
SPT (Shortest processing time)	The lot with the shortest processing time on $R_{q,s}$
LPT (Longest processing time)	The lot with the longest processing time on $R_{q,s}$
SNQ (Smallest number in queue)	After grouping lots with the same $O_{i,I(L_{i,k})}$ , select the lot of the group with the smallest number of lots.
LNQ (Largest number in queue)	After grouping lots with the same $O_{i,I(L_{i,k})}$ , select the lot of the group with the largest number of lots.
FLNQ (Fewest lots at the next queue)	After grouping lots with the same $O_{i,I(L_{i,k})}$ in the WB stocker, select the lot corresponding to the group with the smallest number of lots.
STOCKER	Select one of lots in the cassette or DA stockers (The lots in question do not cause an intentional delay on $R_{q,s}$ ).

lot. Here, the waiting time indicates the time during which a lot stays in the WB stocker after being completed by a DA resource. In contrast, the idle time of WB resource is calculated by subtracting the time at which its last operation ends from the time at which it starts processing the lot.

For each dispatching decision, the score generator calculates score of the decision in the range of  $[0, 1]$  according to the concept of min-max normalization [98]. The formulas for calculating the waiting and idle scores are presented in Equations (5.1) and (5.2), respectively. Here,  $w$  and  $l$  respectively stand for the waiting and idle time related to a lot dispatching decision.

$$s_w = \max\left(-\left(\frac{w - w_{\min}}{w_{\max} - w_{\min}}(s_{\max} - s_{\min})\right) + s_{\max}, s_{\min}\right), \quad (5.1)$$

$$s_l = \max\left(-\left(\frac{l - l_{\min}}{l_{\max} - l_{\min}}(s_{\max} - s_{\min})\right) + s_{\max}, s_{\min}\right), \quad (5.2)$$

where  $w_{\min}$  and  $w_{\max}$  indicate the minimum and maximum values among all the possible values of  $w$ .  $l_{\min}$  and  $l_{\max}$  stand for the minimum and maximum values among all the possible values of  $l$ . Also,  $s_{\min}$  and  $s_{\max}$  refer to the minimum and maximum values among the possible values of score.

In Equations (5.1) and (5.2),  $w_{\max}$  and  $l_{\max}$  are set to be twice the mean of  $w$  and  $l$  values, respectively. This prevents a considerably high  $w$  and  $l$  from being converted to an unwanted positive score, which makes it possible to construct well-balanced training data. This is reasonable because we do not focus on predicting scores precisely; the primary aim is to determine the lot dispatching decision expected to minimize the waiting and idle time.

Based on the training data generated by using two types of scores,  $s_w$  and  $s_l$ , two DNNs, each of which correspond to each score, are trained to predict scores. As a loss function, we used squared errors [99], defined as  $(s_w - s_w^{pre})^2$  and  $(s_l - s_l^{pre})^2$ , and the back-propagation training algorithm is used to minimize the loss function [100]. Here,  $s_w$  and  $s_l$  are the calculated score values for a dispatching decision by using Equation (5.1) and (5.2), respectively, and  $s_w^{pre}$  and  $s_l^{pre}$  mean the predicted scores in terms of the waiting and idle time, respectively, for the decision in the training phase.

The  $t$ th correction of the weights, denoted as  $\theta$ , using the back-propagation training algorithm, is described as

$$\Delta\theta_t = -\eta \frac{\partial E}{\partial \theta_{t-1}} + \alpha \Delta\theta_{t-1}, \quad (5.3)$$

where  $\eta$  means the learning rate,  $\alpha$  is the momentum factor, and  $E$  is the total error between the predicted score and the actual score and is expressed as

$$E = \frac{1}{2} \sum_{i=1}^{N_s} (s(i) - s^{pre}(i))^2 \quad (5.4)$$

Here,  $s(i)$  and  $s^{pre}(i)$  are the actual and predicted scores for the  $i$ th training data, respectively, and  $N_s$  is the total number of training data.

### 5.1.3 Real-time dispatching phase

In the real-time dispatching phase, when a DA resource buffer is empty, input vectors of the number of dispatching rules presented in Table 5.2 are generated for the DA resource. If multiple DA resource buffers are empty at the same time, this process



is performed on the first DA resource that is expected to be in the idle state.

The generated input vectors are given to RTRD as the input, and RTRD predicts two types of scores for each vector. Then the final preference score of each input vector is calculated by the following Equation (5.5).

$$totalscore = \lambda_w \times s_w^{pre} + \lambda_l \times s_l^{pre}, \quad (5.5)$$

with  $\lambda_w + \lambda_l = 1$ . Value of  $\lambda_w$  is the weight or importance of the flow time of lots, and  $\lambda_l$  is the weight or importance of resource utilization in the bottleneck stage. As a result of the equation, the dispatching rule involved in the vector with the highest *totalscore* is selected as the lot dispatching decision. Then, one of the candidate lots is assigned to the DA resource according to the selected dispatching rule.

The proposed dispatcher is anticipated to reach a better lot dispatching decision quickly compared to the conventional meta-heuristics through a simple calculation using the weights predetermined during the training. Therefore, it is expected that the proposed method can be introduced into the real-world assembly lines where lot dispatching decisions are required to be determined in a real-time manner.

## 5.2 Real-time lot selection dispatcher

### 5.2.1 Dispatcher structure

Fig. 5.3 illustrates the architecture of RTLD with the same number of layers as RTRD. The input layer contains 25 nodes, and the number of nodes in each hidden layer are 64, 32 and 16, respectively, whereas the output layer has one node. As in RTRD, the ReLU is applied before each hidden layer in order to provide a non-linear

transformation, and all layers are fully connected [96].

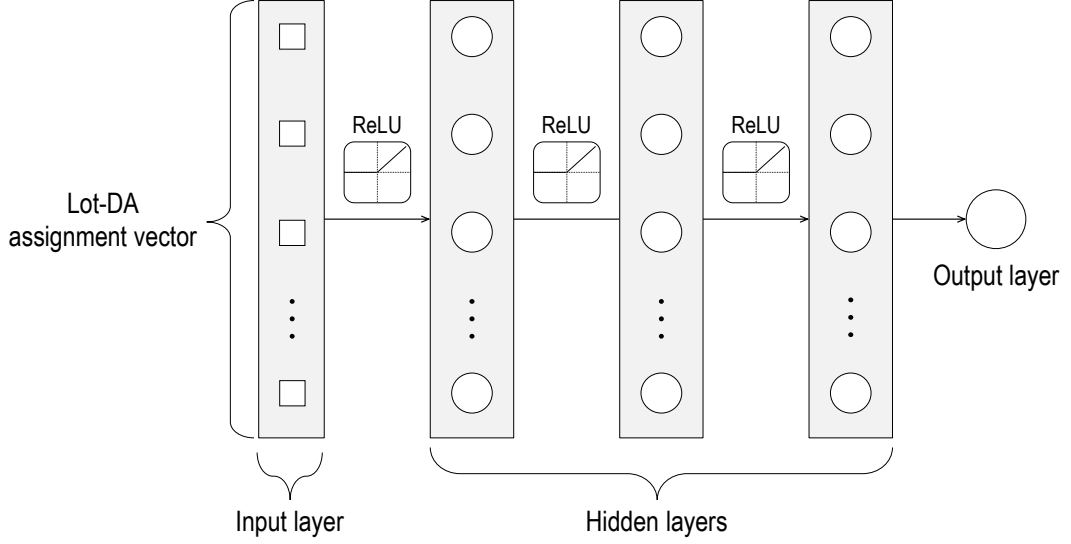


Figure 5.3: The structure of RTLD.

A pair of a lot among the candidate lots and a DA resource is represented as a vector called a lot-DA assignment vector for the input to RTLD. Specifically, Table 5.3 presents the components of the lot-DA assignment vector, for a particular lot,  $L_{i,k}$ , and a particular DA resource,  $R_{q,s}$ . The main difference between RTRD and RTLD, when it comes to generating an input vector for DNN, is that RTRD only utilizes information about the distribution of lots, while RTLD considers the characteristics of each lot in the candidate lots.

Using the defined features of a lot-DA assignment vector, RTLD is capable of predicting how long the lot waits in the WB stocker after it is processed by a DA resource. Furthermore, the features provide a clue for estimating the idle time of the WB resource that will process the lot involved in the vector. As a result, RTLD is expected to be able to conduct lot dispatching decisions which reduce the waiting

Table 5.3: Components of the lot-DA assignment vector, for  $L_{i,k}$  and  $R_{q,s}$ .

Categories	Descriptions
Conflicting lots of $L_{i,k}$	# of conflicting lots being moved from cassette and DA stockers to DA resource buffers
	# of conflicting lots in DA resource buffers
	# of conflicting lots being processed on DA resources
	# of conflicting lots being moved from DA resources to a WB stocker
	# of conflicting lots waiting in the WB stocker
Characteristics of $L_{i,k}$	# of chips in $L_{i,k}$
	Progress rate of $J_i$ (# of lots that complete all operations assigned/ $n_i$ )
	The smallest index among those of the operations waiting to be processed ( $I(L_{i,k})$ )
	# of WB resources that are able to process $L_{i,k}$
Characteristics of $R_{q,s}$	The remaining time until $R_{q,s}$ becomes idle
	Processing time of $L_{i,k}$ on $R_{q,s}$
Delay time	The interval between the time when the lot dispatching decision is made and the time when $L_{i,k}$ starts being processed by $R_{q,s}$
	# of lots in the cassette stocker
Lot distribution	# of lots being moved from the cassette stocker to DA resource buffers
	# of lots in DA resource buffers
	# of lots being processed on DA resources
	# of lots being moved from WB stocker to WB resource buffers
	# of lots in WB resource buffers
	# of lots being processed on WB resources
	# of lots being moved from WB resources to the DA stocker
	# of lots in the DA stocker
	# of lots being moved from the DA stocker to DA resource buffers
	# of lots being moved from DA resources to the WB stocker
	# of lots in the WB stocker
	# of lots that completed all operations

time of lots and the idle time of WB resources.

That is, the lot-DA assignment vector contains 25 features categorized into conflicting lots, the characteristics of the lot and DA resource, the delay time, and the distribution of lots as shown in Table 5.3. First, the concept of conflicting lots is introduced to represent lots that compete for WB resources. Lot,  $L_{i',k'}$ , is called a conflicting lot of  $L_{i,k}$  if  $L_{i',k'}$  and  $L_{i,k}$  satisfy either of the conditions presented in Equations (5.6) and (5.7).

$$A(O_{i',I(L_{i',k'})}) \cap A(O_{i,I(L_{i,k})+1}) \neq \emptyset \quad (5.6)$$

$$A(O_{i',I(L_{i',k'})+1}) \cap A(O_{i,I(L_{i,k})+1}) \neq \emptyset \quad (5.7)$$

The five features corresponding to the conflicting lots of  $L_{i,k}$  are presented in Table 5.3 according to their status. Each feature represents the number of conflicting lots that correspond to one of the five different states. RTLD is capable of capturing the distribution of the lots by collectively using all these features.

Next, there are four features representing various characteristics of  $L_{i,k}$  and two for  $R_{q,s}$ . Features associated with  $L_{i,k}$  include the number of chips in  $L_{i,k}$ , the progress rate of  $J_i$ ,  $I(L_{i,k})$ , and the number of WB resources capable of processing  $L_{i,k}$ . The last one is included as a feature to capture the degree of potential conflict among the lots in the WB stage. In contrast to other features, this one for  $L_{i,k}$  has a fixed value according to its resource type required in the WB stage regardless of the other lot dispatching decisions.

The first feature related to  $R_{q,s}$  is identical to that in the state vector of RTRD.

The second one is the processing time of  $L_{i,k}$  on  $R_{q,s}$ , which indicates how long  $L_{i,k}$  will stay on  $R_{q,s}$ . The features corresponding to the lot distribution category have the same meaning as those in the state vector of RTRD.

Finally, the delay time refers to how long it takes from the moment the lot dispatching decision is made until  $R_{q,s}$  starts processing  $L_{i,k}$ . If the status of  $L_{i,k}$  is **In-Cassette-Stocker** or **In-DA-Stocker**, the value of the delay time is calculated as the sum of the time required for  $L_{i,k}$  to move from the stocker to the buffer of  $R_{q,s}$  and the time that  $L_{i,k}$  spends in the buffer. Otherwise, the time required for  $L_{i,k}$  to move from the current location to DA stocker is added to the value mentioned above. Through this feature, RTLTD is capable of inferring whether or not  $L_{i,k}$  causes an intentional delay in  $R_{q,s}$ .

The output layer represents the preference score of the assignment vector. All values in each node are normalized to a range  $[0, 1]$  using the min-max normalization to accommodate the inconsistencies of different units [98].

### 5.2.2 Training phase

In the training phase, unlike in RTRD, RDG is responsible for randomly selecting one among candidate lots. Accordingly, the distribution of candidate lots by status is reflected in the probability that lot is selected by RDG. This leads to the result that the generated simulation logs do not make a significant difference in terms of performances when a problem is solved by RDG multiple times.

Therefore, to obtain various simulation logs in terms of the flow time and resource utilization, for each simulation, the intentional delay level with a value between 0 and 1 is randomly selected. The intentional delay level close to one means a high prob-

ability of selecting lot whose status is **To-DA-Stocker** or **At-WB-Resource**, whereas the intentional delay level close to zero indicates a high probability of selecting lot whose status is in **In-Cassette-Stocker** or **In-DA-Stocker**.

Specifically, whenever a lot dispatching is required, a new random number with a value between 0 and 1 is generated. If this random number does not exceed the intentional delay level for that simulation, RDG makes a dispatching decision by using only lots whose statuses are **To-DA-Stocker** or **At-WB-Resource** among candidates, to simulate the intentional delay by letting the DA resource associated with the decision to be idle until the dispatched lot ready to be processed by the resource.

The remaining processes, after RDG completes all the lot dispatching decisions of each problem, are identical to those in the training phase of RTRD.

### 5.2.3 Real-time dispatching phase

In the real-time dispatching phase, when a DA resource buffer is empty, lot-DA assignment vectors for all the possible assignments between candidate lots and DA resources are generated. The generated lot-DA assignment vectors are given to RTLD as the input, and RTLD predicts the two types of scores,  $s_w^{pre}$  and  $s_l^{pre}$ , for each vector.

Due to the fact that RTLD can evaluate individual lots unlike RTRD, another score, called  $s_d$ , is introduced to RTLD. The trained dispatchers are able to predict the waiting time of the lots in the WB stocker. However, the time that lots spend in the DA stocker can be easily overlooked. Although the DA stocker is not at a bottleneck stage, the time during which a lot stays in the DA stocker has to be addressed since the flow time naturally increases if lots stay in the DA stocker for a

long time. The formula for calculating  $s_d$  is presented in Equation (5.8).

$$s_d = \max\left(-\left(\frac{d - d_{min}}{d_{max} - d_{min}}(s_{max} - s_{min})\right) + s_{max}, s_{min}\right), \quad (5.8)$$

where  $d$  represents the time during which a lot stays in the DA stocker after being completed by a WB resource. Unlike  $s_w^{pre}$  and  $s_l^{pre}$ , the training phase for  $s_d$  does not have to be predicted since the value of  $s_d$  is simply calculated by the simulator when a lot dispatching decision is required.

Then, the final preference score of each input vector is calculated by the following Equation (5.9).

$$totalscore = \lambda_w \times s_w^{pre} + \lambda_l \times s_l^{pre} + \lambda_d \times s_d, \quad (5.9)$$

with  $\lambda_w + \lambda_l + \lambda_d = 1$ . The meanings of  $\lambda_w$  and  $\lambda_l$  are the same as those in RTRD while a new term is added to Equation (5.5). The value of  $\lambda_d$  is the weight or importance of lots' waiting time in the DA stocker. Consequently, the lot and DA resource involved in the vector with the highest *totalscore* are selected as the lot dispatching decision. Then, the lot is assigned to the DA resource.

# Chapter 6

## Experiments

### 6.1 Datasets

To validate the proposed approaches, we prepared 12 datasets that correspond to diverse configurations by varying the numbers of resources, job types, and lots, as shown in Table 6.1. A lot is assigned a particular number uniformly distributed between 100 and 750 when it comes to the number of chips it has. It is assumed that dispatching practice is to satisfy as many production requirements as possible that are given for the next 48 hours, as described in [34]. Production requirements for the assembly line always exceed the line capacity and it takes more than 48 hours to finish all the operations [31].

Datasets prepared for the experiments can be classified into four groups: datasets 1 to 3, 4 to 6, 7 to 9 and 10 to 12. Compared to datasets 1 to 6, datasets 7 to 12 represent the dispatching problems with the large number of job types. Furthermore, the problems of datasets 4 to 6 have more resources than those of datasets 1 to 3, and the same relationship goes with datasets 10 to 12 and datasets 7 to 9. The three datasets in each group represent the different levels of difficulty for the dispatching problem of assembly lines. In this experiment, a level of difficulty is measured in terms of the number of operations to complete the entire processing of lots. A



higher number of operations involved in a dataset leads to a greater computational complexity in it because the number of decisions required to resolve problems in the dataset increases proportionally.

In each dataset, 100 problems were generated by varying the quantity of the lots and the total number of lots for each job type. Specifically, 30 and 20 problems were used to train and validate, respectively, the proposed dispatcher, while, after it was trained, the remaining problems were used to test the performance of the proposed dispatchers. In the experiments,  $N_M$  and  $N_J$  are set to be 6 and 8, respectively. Specifically, for each problem, both the DA and WB stages have three different resource types. For the problems with 16 resources, four and 12 resource are assigned to DA and WB stages, respectively, and for the problems with 70 resources, 20 and 50 resources are assigned to DA and WB stages, respectively.

Table 6.2 represents the operations for each job type and resource types that can perform each operation. The odd-numbered operations of a job type are assumed to be processed in the DA stage, whereas the even-numbered operations are processed in the WB stage. The last column in Table 6.2 shows the processing time of each operation per chip in the corresponding resource type. Additionally, it takes 900 seconds for a lot to move from the stocker to the resource buffer, and vice versa.

## 6.2 Experiment settings

In the training phase, we generated simulation logs by solving each problem 100 times using RDG. The scores of the generated simulation logs were calculated by the score generator based on Equations (5.1) and (5.2). The performance of lot dispatching is measured by means of the waiting time of lots and the idle time of

Table 6.1: Descriptions on the datasets used for the experiments.

Dataset No.	Number of resources	Avg. numbers of lots of each job type								Avg. number of operations
		$J_1$	$J_2$	$J_3$	$J_4$	$J_5$	$J_6$	$J_7$	$J_8$	
1	16	33.99	-	24.33	24.78	-	24.48	14.20	-	524.90
2	16	24.16	-	24.62	24.29	-	24.30	24.6	-	586.56
3	16	14.72	-	24.68	24.90	-	25.06	34.92	-	657.48
4	70	103.41	-	74.79	74.34	-	74.51	44.93	-	1609.84
5	70	74.38	-	74.46	74.78	-	74.51	74.90	-	1791.98
6	70	45.12	-	74.79	74.76	-	75.00	104.57	-	1975.00
7	16	24.26	24.66	22.81	17.75	17.84	12.07	9.91	11.02	571.30
8	16	17.68	17.34	17.71	17.07	17.54	17.67	17.33	17.54	664.30
9	16	10.39	10.62	12.48	17.34	17.48	22.04	24.38	24.72	756.26
10	70	64.66	64.51	59.76	54.73	49.73	34.63	33.93	34.48	1670.28
11	70	49.52	49.10	50.01	49.24	49.26	49.55	49.74	49.33	1881.14
12	70	29.46	29.21	34.16	49.99	49.68	54.42	69.72	69.36	2091.82

Table 6.2: Problem description for experiments.

$J_i$	$O_{i,j}$	Stages	$A(O_{i,j})$	Processing time
$J_1$	$O_{1,1}$	DA	$(M_1, M_2, M_3)$	$(3, 3, 4.5)$
	$O_{1,2}$	WB	$(M_4, M_5, M_6)$	$(16, 18, 18)$
$J_2$	$O_{2,1}$	DA	$(M_1, M_2, M_3)$	$(3, 4.5, 4.5)$
	$O_{2,2}$	WB	$(M_5, M_6)$	$(20, 20)$
$J_3$	$O_{3,1}$	DA	$(M_1, M_2, M_3)$	$(6, 7.5, 6)$
	$O_{3,2}$	WB	$(M_4, M_5, M_6)$	$(26, 28, 28)$
	$O_{3,3}$	DA	$(M_1, M_2, M_3)$	$(7.5, 7.5, 7.5)$
	$O_{3,4}$	WB	$(M_4, M_5, M_6)$	$(26, 28, 28)$
$J_4$	$O_{4,1}$	DA	$(M_2, M_3)$	$(7.5, 6)$
	$O_{4,2}$	WB	$(M_4, M_5, M_6)$	$(36, 36, 36)$
	$O_{4,3}$	DA	$(M_2, M_3)$	$(7.5, 7.5)$
	$O_{4,4}$	WB	$(M_4, M_5, M_6)$	$(44, 46, 44)$
$J_5$	$O_{5,1}$	DA	$(M_1, M_3)$	$(6, 6)$
	$O_{5,2}$	WB	$(M_4, M_6)$	$(14, 20)$
	$O_{5,3}$	DA	$(M_1, M_3)$	$(6, 6)$
	$O_{5,4}$	WB	$(M_4, M_6)$	$(14, 20)$
$J_6$	$O_{6,1}$	DA	$(M_2, M_3)$	$(9, 9)$
	$O_{6,2}$	WB	$(M_4, M_5, M_6)$	$(36, 36, 40)$
	$O_{6,3}$	DA	$(M_2, M_3)$	$(7.5, 7.5)$
	$O_{6,4}$	WB	$(M_4, M_6)$	$(40, 70)$
	$O_{6,5}$	DA	$(M_2, M_3)$	$(7.5, 7.5)$
	$O_{6,6}$	WB	$(M_5, M_6)$	$(46, 44)$
$J_7$	$O_{7,1}$	DA	$(M_1, M_2, M_3)$	$(6, 6, 4.5)$
	$O_{7,2}$	WB	$(M_4, M_5, M_6)$	$(56, 50, 50)$
	$O_{7,3}$	DA	$(M_1, M_2)$	$(6, 6)$
	$O_{7,4}$	WB	$(M_4, M_5, M_6)$	$(40, 40, 24)$
	$O_{7,5}$	DA	$(M_1, M_2)$	$(4.5, 4.5)$
	$O_{7,6}$	WB	$(M_4, M_5, M_6)$	$(40, 40, 24)$
	$O_{7,7}$	DA	$(M_1, M_2, M_3)$	$(4.5, 4.5, 4.5)$
	$O_{7,8}$	WB	$(M_4, M_5, M_6)$	$(40, 40, 24)$
$J_8$	$O_{8,1}$	DA	$(M_1, M_2, M_3)$	$(6, 6, 4.5)$
	$O_{8,2}$	WB	$(M_4, M_5)$	$(50, 30)$
	$O_{8,3}$	DA	$(M_2, M_3)$	$(6, 6)$
	$O_{8,4}$	WB	$(M_4, M_5)$	$(50, 30)$
	$O_{8,5}$	DA	$(M_2, M_3)$	$(4.5, 4.5)$
	$O_{8,6}$	WB	$(M_4, M_5)$	$(30, 50)$
	$O_{8,7}$	DA	$(M_1, M_2, M_3)$	$(4.5, 4.5, 4.5)$
	$O_{8,8}$	WB	$(M_4, M_5, M_6)$	$(30, 50, 40)$

the resources of the WB stage. The average waiting time of lots,  $AWT$ , is calculated as:

$$AWT = \frac{\sum_{i=1}^{N_J} \sum_{k=1}^{n_i} (c_{i,k} - r_{i,k} - t_{i,k})}{\sum_{i=1}^{N_J} n_i}, \quad (6.1)$$

where  $c_{i,k}$  and  $r_{i,k}$  are the time that the last WB operation of  $L_{i,k}$  is completed and  $L_{i,k}$  leaves the cassette stocker, respectively, and  $t_{i,k}$  indicates the sum of processing time of  $L_{i,k}$  on resources.

Additionally, the average idle time of the WB resources,  $AIT$ , is defined as follows:

$$AIT = \frac{\sum_{\forall q \in \Omega} \sum_{s=1}^{n_q} (f_{q,s} - t_{q,s})}{\sum_{\forall q \in \Omega} n_q}, \quad (6.2)$$

where  $f_{q,s}$  represents the time at which  $R_{q,s}$  completes its last operation and  $t_{q,s}$  is the total processing time of the operations assigned to  $R_{q,s}$  between time 0 and  $f_{q,s}$ , and  $\Omega$  is a set of indices of the resource types which belong to the WB stage.

Moreover, to collectively measure the overall performance in terms of  $AWT$  and  $AIT$ , we used the average loss time, or  $ALT$ , which is the arithmetical mean of these two values. However, in the real-world, the weights for the two performance measures,  $AWT$  and  $AIT$ , depend on the characteristics of RMLs or the operators' judgement. For instance, when the capacity of stockers is sufficient or changes in demand are insignificant, reducing  $AIT$  may be more important than reducing  $AWT$ . Therefore, in section 6.3.2, we measured the change in the values of  $AWT$ ,  $AIT$ , and  $ALT$  while changing the weights in the Equations (5.5) and (5.9) for each dataset.

For comparison purpose, we implemented the conventional dispatching rules presented in Table 5.2, all of which are widely used to reduce the flow time or increase resource utilization [62, 97, 101, 102]. Here, FIFO selects the oldest lot that has been dispatched from the cassette stocker among the candidate lots. This rule randomly dispatches the lot from the cassette stocker when the lots in the status of **In-Cassette-Stocker** exist in the candidate lots only.

Furthermore, we also compared the proposed dispatchers with the composite dispatching rule using SVR proposed by [65]. Among the dispatching rules presented in Table 5.2, five rules including FIFO, MOR, LARGE, LNQ, and STOCK were used to construct a linear combination of the composite dispatching rule. This is due to the fact that the combination above yielded the best performance among all the combinations of dispatching rules. The weight assigned to each rule is determined by the model trained with SVR whenever a lot dispatching decision is required. The feature set and the parameters of SVR used in the experiments are identical to those in [65].

For each dataset, we trained two proposed dispatchers, RTRD and RTLD, using generated training data. Table 6.3 presents the training results. In Table 6.3,  $val_w$  and  $val_l$  represent the minimum values of the validation error in terms of the waiting and idle time, respectively. In addition,  $iter_w$  and  $iter_l$  are the numbers of iteration required to achieve the validation error for the waiting and idle time, respectively. The dispatchers in each dataset use the DNNs obtained when the numbers of iteration of the training process in terms of waiting and idle time reach the values of  $iter_w$  and  $iter_l$ , respectively.

For RTRD,  $\lambda_w$  and  $\lambda_l$  are set to 1/2 and 1/2, respectively. For RTLD,  $\lambda_w$ ,  $\lambda_l$ ,

and  $\lambda_d$  are set to 1/3, 1/3 and 1/3, respectively.

Table 6.3: Training results of the proposed dispatchers.

Dataset No.	Dispatchers	$val_w$	$val_l$	$iter_w$	$iter_l$
1	RTRD	0.0123	0.0988	35	150
	RTLD	0.0122	0.0385	31	318
2	RTRD	0.0229	0.0942	23	112
	RTLD	0.0155	0.0370	32	160
3	RTRD	0.0271	0.0898	41	76
	RTLD	0.0196	0.0355	30	86
4	RTRD	0.0001	0.0853	59	229
	RTLD	0.0151	0.0136	22	303
5	RTRD	0.0004	0.0876	10	283
	RTLD	0.0158	0.0124	49	266
6	RTRD	0.0003	0.0873	3	112
	RTLD	0.0162	0.0116	43	75
7	RTRD	0.0157	0.1064	17	80
	RTLD	0.0129	0.0400	56	154
8	RTRD	0.0252	0.1000	12	95
	RTLD	0.0149	0.0370	33	130
9	RTRD	0.0323	0.0925	27	119
	RTLD	0.0188	0.0349	39	115
10	RTRD	0.0001	0.0839	14	280
	RTLD	0.0127	0.0131	16	166
11	RTRD	0.0573	0.0616	11	130
	RTLD	0.0157	0.0352	50	158
12	RTRD	0.0002	0.0828	7	190
	RTLD	0.0163	0.0111	67	204

## 6.3 Experiment results

### 6.3.1 Performance comparison

The performance comparison results of the proposed dispatchers, SVR, and the existing dispatching rules are presented in Figures 6.1, 6.2, 6.3, and 6.4 which show the results for datasets 1, 2, 7, and 11, respectively. The detailed results for each dataset are presented in Appendix A. In terms of *AWT*, *FIFO*, *LOR*, *SMALL*, *LARGE*, *SPT*, *LPT*, *FLNQ*, and, *SNQ* show the better performances than the proposed dispatchers for all the datasets.

In detail, *FIFO* assigns a low priority level to the lot whose status is **In-Cassette-Stocker**, and *LOR* prefers the job type whose number of operations is small. As a result, the waiting time of lots are reduced owing to the decrease in the number of lots in the WB stocker. In the case of *LARGE*, *SMALL*, *SPT*, *LPT*, *FLNQ*, *SNQ*, there is a high probability that lots preferred by each rule are repeatedly selected until all operations of them are completed. The result is that lots in the status of **To-DA-Stocker** or **At-WB-Resource** are selected, which makes it difficult for newly arrived lots to enter the assembly line. Accordingly, the amount of WIP decreases and the idle time of WB resources naturally increases.

Meanwhile, *MOR*, *LNQ*, and *STOCK* showed better performances in terms of *AIT* than other dispatching rules. *MOR* tends to disallow resources from being idle since this rule assigns a higher priority to a lot with a higher number of operations to be processed. Since *LNQ* attempts to balance the number of lots as per their operation types, lots corresponding to the first operation type for each job type are mainly selected at an early stage with a low WIP level. This increases the amount of

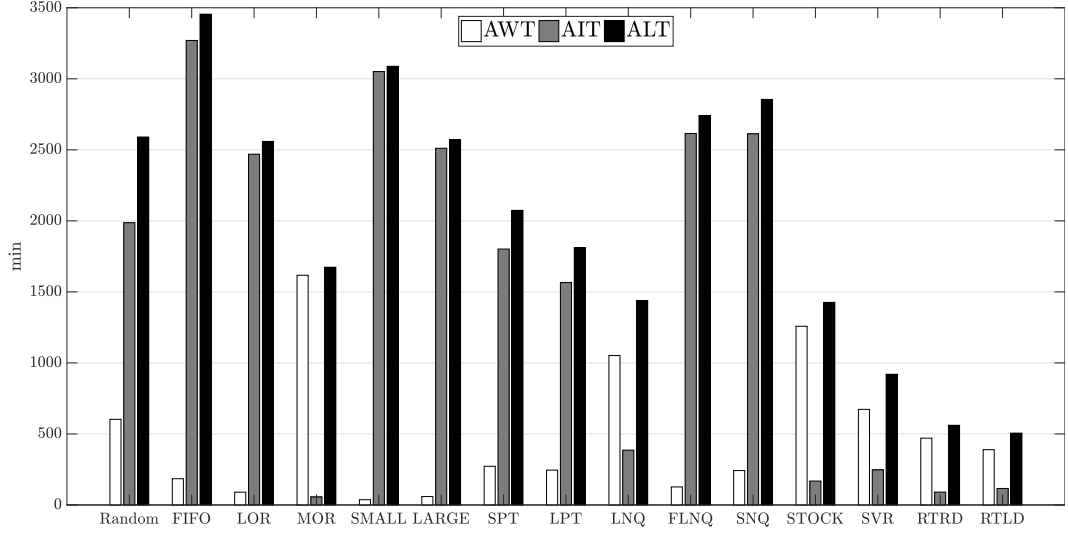


Figure 6.1: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 1.

WIP, leading to a shorter idle time of WB resources. STOCK only selects lots in the status of *To-DA-Stocker* or *At-WB-Resource* when there is not any lot in cassette or DA stockers. Because of this tendency, the lots that undergo an intentional delay on DA resources are rarely selected.

In terms of *ALT*, the proposed dispatchers and SVR outperformed the other methods in all datasets. While most existing methods tend to minimize only one performance measure, the proposed dispatchers addressed both two measures at the same time. Among the two proposed dispatchers, RTLD showed better performance than RTRD. Although RTRD achieved a lower value of *AIT* than RTLD, *AWT* of RTRD was more than twice that of RTLD for multiple datasets. This difference in performance can be attributed to the fact that RTLD considers the time that lots spend in the DA stocker due to the existence of *score<sub>d</sub>*.

SVR achieved a lower *ALT* than RTRD in some datasets (shown in Figures 6.3



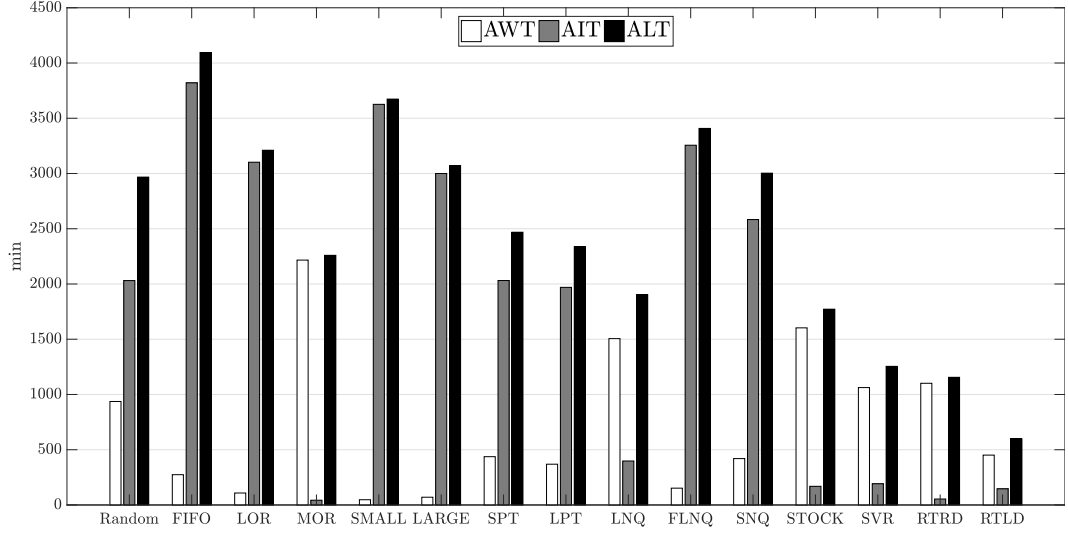


Figure 6.2: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 2.

and 6.4). However, it showed poorer performances than RTLD in terms of *ALT* for all datasets. This is due to the lack of features that can represent the concept of conflicting lots and candidate lots including ones with possible intentional delays on a DA resource, although SVR attempts to choose a proper dispatching rule according to a given assembly line state.

Based on the above observation, the proposed dispatchers successfully reduce both *AWT* and *AIT* at the same time in contrast to the existing methods which focus only on one performance measure. Therefore, the proposed dispatchers appear to achieve a reduction in the flow time while maintaining high utilization of resources in the bottleneck stage.

Table 6.4 highlights the improvement rate of RTLD over the other dispatching methods in terms of *ALT*. Except for RTRD, RTLD has reduced *ALT* from at least 33% to 85% compared to the existing methods. To provide further analysis of the

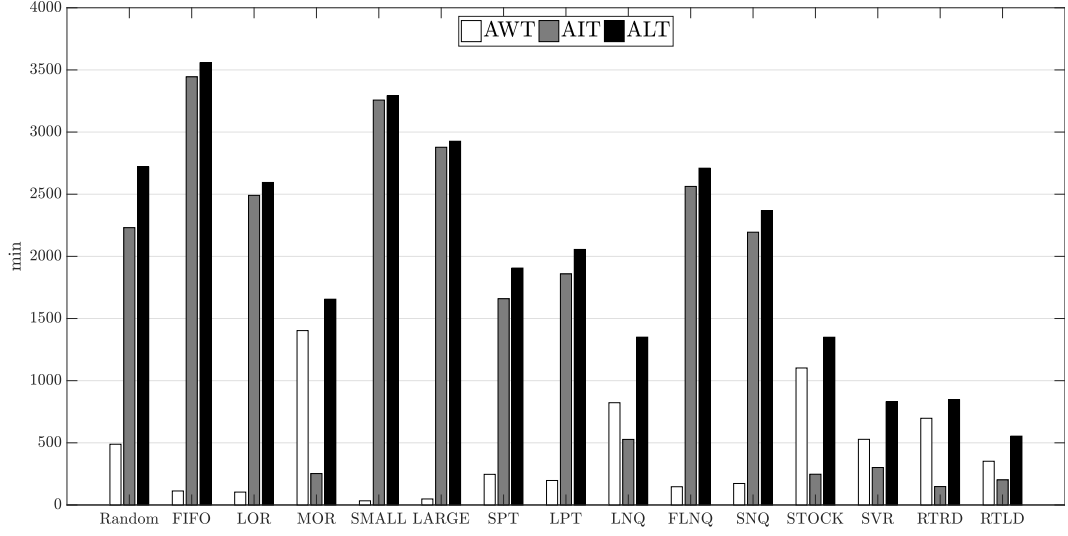


Figure 6.3: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 7.

effects of the lot dispatching decisions, we compared various performance indicators between the dispatching methods for one problem belonging to dataset 2, which has the greatest difference in performance between RTRD and RTLD. Figures 6.5–6.7 present WIP levels of each stage; the dispatching results of each resource over time; and resource utilization of each stage obtained by using dataset 2, respectively. Here, the values on the utilization graph represent the numbers of resources processing a lot divided by the total number of resources.

According to Figures 6.5b and 6.5c, RTRD and SVR keep the WIP level of WB stage close to zero while the WIP level of Figure 6.5a often reached a value of two or above. However, the two dispatching methods, RTRD and SVR, failed to prevent the WIP level of DA stage from increasing dramatically. This is because they are not able to address the WIP level of the DA stocker. Therefore, even if the rules such as SPT, which are known to minimize the flow time, are used in the RTRD's

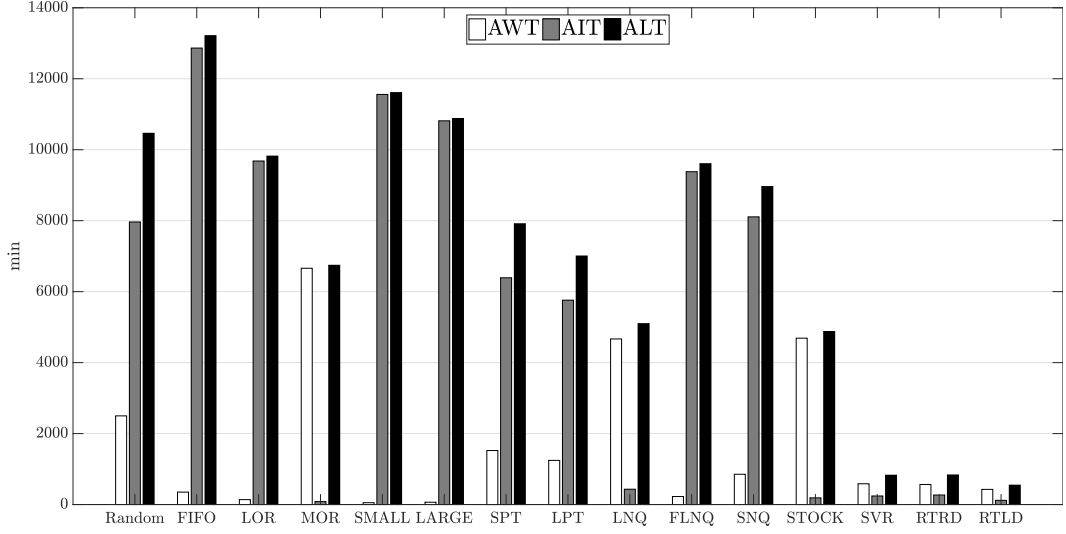


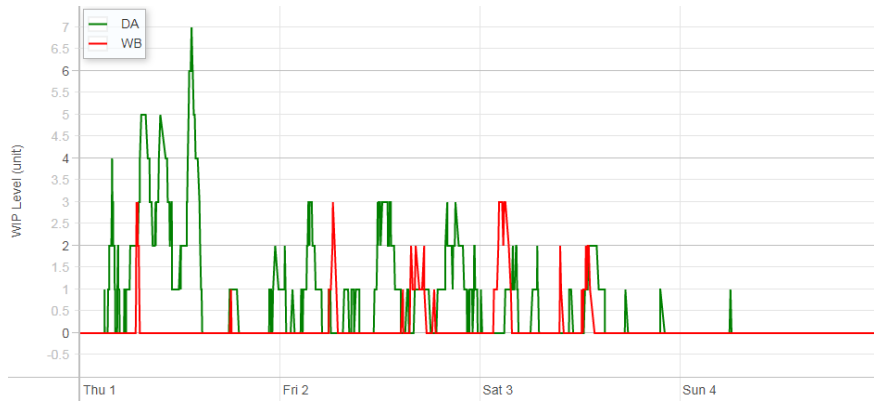
Figure 6.4: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 11.

decision-making process, RTRD has a limitation in reducing the waiting time of lots in the DA stage. On the other hand, it is seemed that RTLD is able to effectively control the amount of WIP in the DA stocker due to the presence of  $s_d$ .

The results in Figures 6.6a and 6.7a indicate that RTLD keeps the utilization of WB resources around 100% at all times except the early stage. Figures 6.6b and 6.6c present that RTRD and SVR often involve the WB resource idle state, which causes fluctuations in WB resource utilization (shown in Figures 6.7b and 6.7c). This may be because the two methods tend to excessively reduce the amount of WIP in the WB stocker.

Table 6.4: *ALT* improvement rates of RTLD compared to the existing methods and RTRD.

Dataset No.	Random	FIFO	LOR	MOR	SMALL	LARGE	SPT	LPT	LNQ	FLNQ	SNQ	STOCK	SVR	RTRD
1	80%	85%	80%	70%	84%	80%	76%	72%	65%	82%	82%	65%	45%	10%
2	80%	85%	81%	74%	84%	81%	76%	74%	69%	82%	80%	66%	52%	48%
3	80%	83%	79%	75%	81%	77%	73%	71%	67%	81%	80%	64%	58%	37%
4	67%	78%	70%	49%	73%	64%	56%	51%	45%	73%	73%	47%	37%	6%
5	74%	82%	76%	63%	78%	70%	63%	62%	60%	79%	73%	58%	49%	11%
6	67%	78%	69%	60%	72%	61%	54%	52%	52%	74%	71%	50%	44%	13%
7	80%	84%	79%	67%	83%	81%	71%	73%	59%	80%	77%	59%	33%	35%
8	79%	85%	81%	70%	84%	82%	77%	74%	66%	81%	79%	62%	48%	39%
9	80%	84%	80%	71%	82%	80%	77%	71%	66%	80%	79%	62%	58%	39%
10	75%	82%	76%	60%	80%	74%	58%	65%	55%	78%	75%	58%	33%	7%
11	95%	96%	94%	92%	95%	95%	93%	92%	89%	94%	94%	89%	34%	35%
12	74%	84%	80%	66%	82%	77%	72%	67%	64%	82%	78%	61%	63%	41%



(a) RTLD

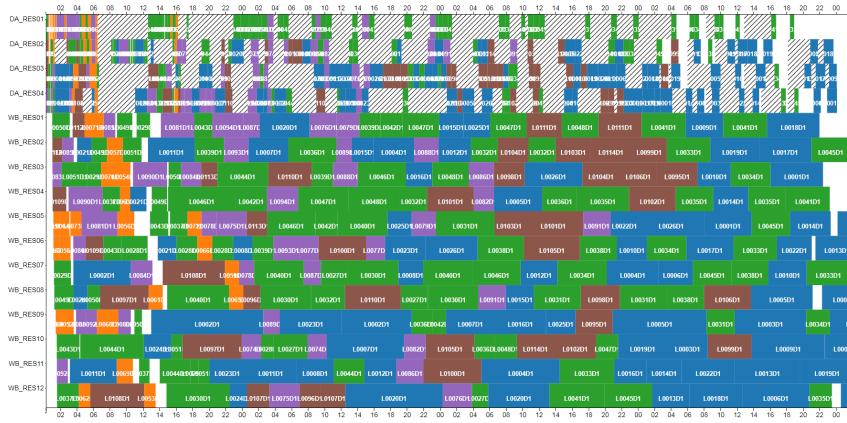


(b) RTRD

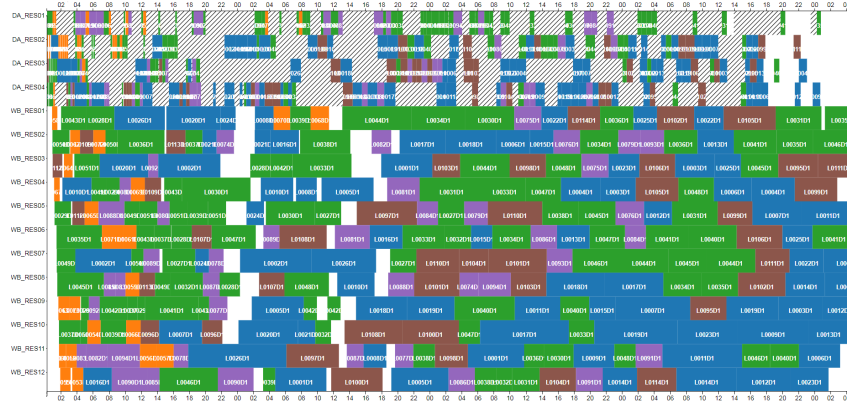


(c) SVR

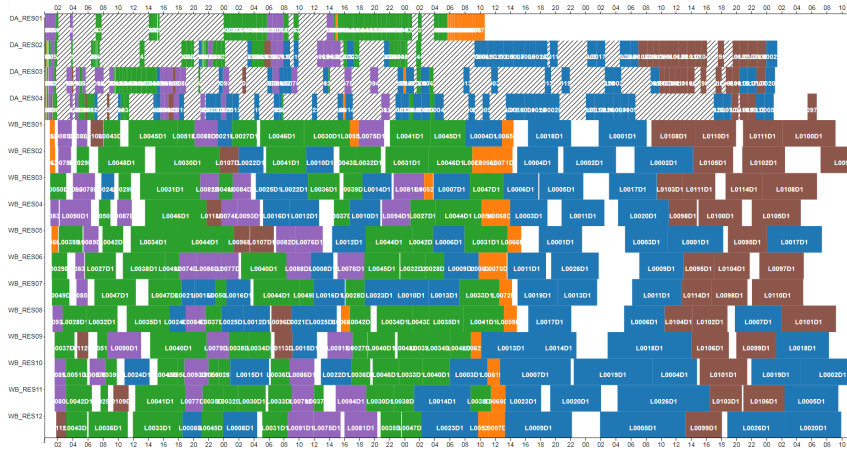
Figure 6.5: WIP graphs of the proposed dispatchers, and SVR.



(a) RTLD

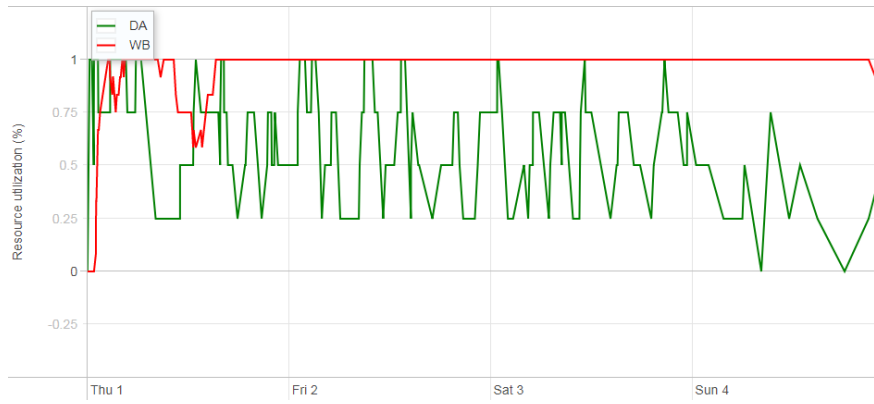


(b) RTRD

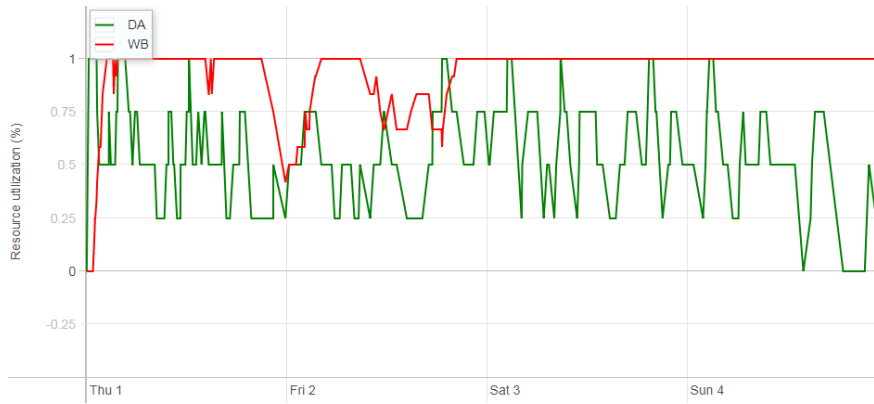


(c) SVR

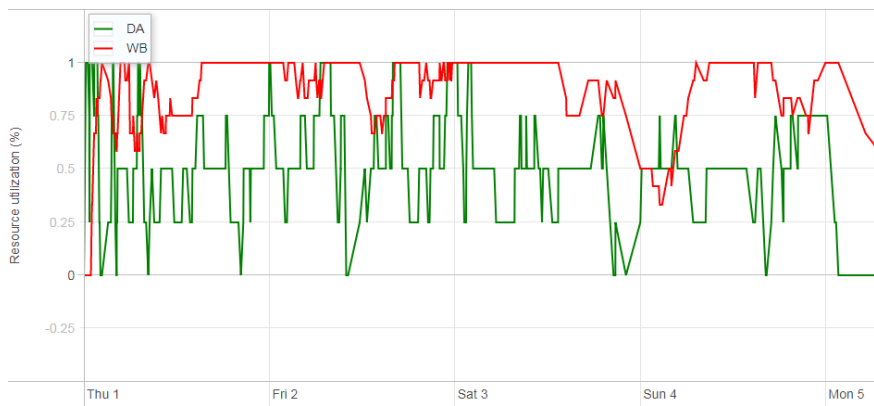
Figure 6.6: Gantt charts of the proposed dispatchers and SVR.



(a) RTLD

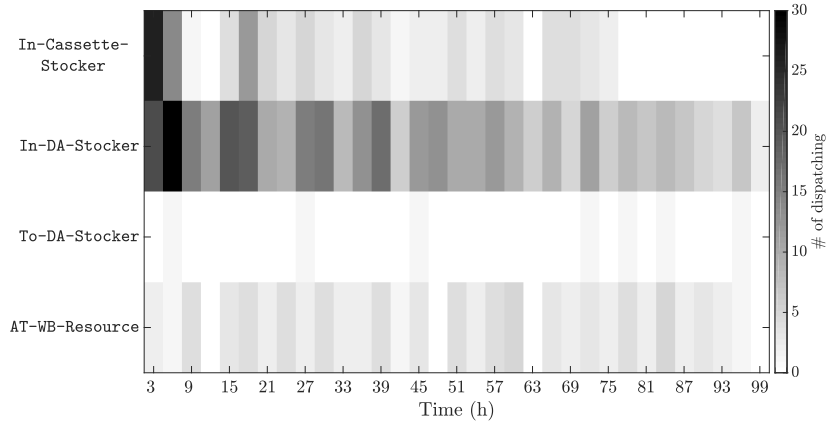


(b) RTRD

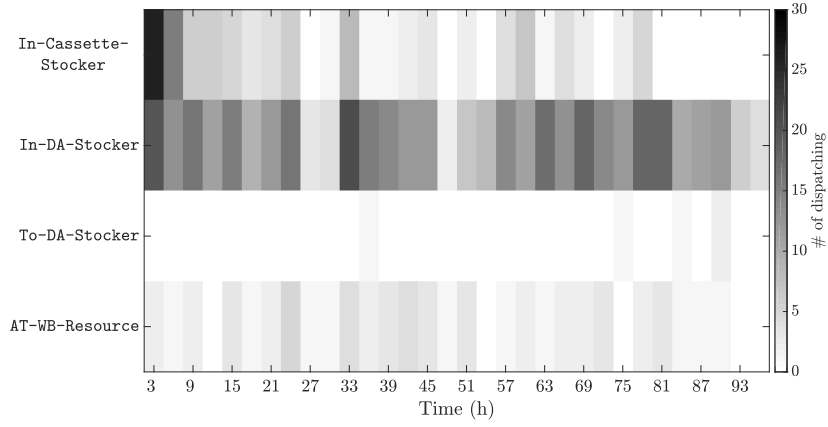


(c) SVR

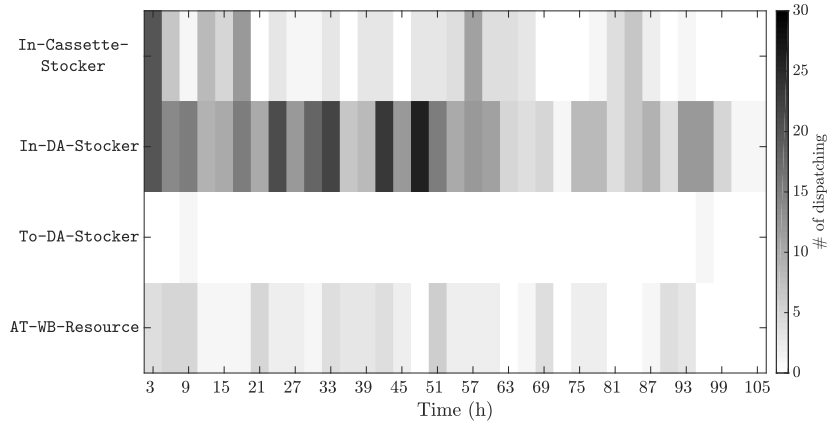
Figure 6.7: Utilization graphs of the proposed dispatchers and SVR.



(a) RTLD



(b) RTRD



(c) SVR

Figure 6.8: Dispatching frequencies according to lot statuses over time.



Additionally, we compared the lot dispatching decision patterns over time by the proposed dispatchers and SVR. In Figure 6.8, heat maps visualize the frequency of status of the lots dispatched by each dispatching method over time, where Figures 6.8a–c represent the results of RTLD, RTRD, and SVR, respectively. The value in Figure 6.8 indicates the number of dispatched lots that correspond to each status for the interval of three hours.

In Figure 6.8a, RTLD appears to increase the utilization of WB resources by dispatching lots in the status of **In-Cassette-Stocker** at an early stage with a low WIP level. Subsequently, when the WIP level reaches a sufficient extent, it tends to prevent the waiting time of lots from increasing by dispatching lots in the status of **In-DA-Stocker** or **At-WB-Resource**. RTRD has a dispatching pattern similar to that of RTLD while there are differences in the timing of selecting lots whose status is **In-DA-Stocker**. As can be seen from Figure 6.8b, RTRD mainly focused on selecting lots whose status is **In-DA-Stocker** only when the amount of WIP in the DA stocker increases (shown in Figure 6.5b).

Compared to the proposed dispatchers, SVR gives a lower priority to the lots whose status is **In-Cassette-Stocker** and a higher one to the lots whose status is **At-WB-Resource** as presented in Figure 6.8c. This has contributed to the increase in the amount of WIP in the DA stocker, causing the flow time of lots to rise.

Finally, an analysis was conducted on the computation time of the proposed dispatchers and SVR. Figure 6.9 presents the computation time of the dispatching methods spent processing given lots according to the number of operations. For all of the average of the number of operations, the computation time of SVR is always longer than that of the proposed dispatchers. Combined with the previous

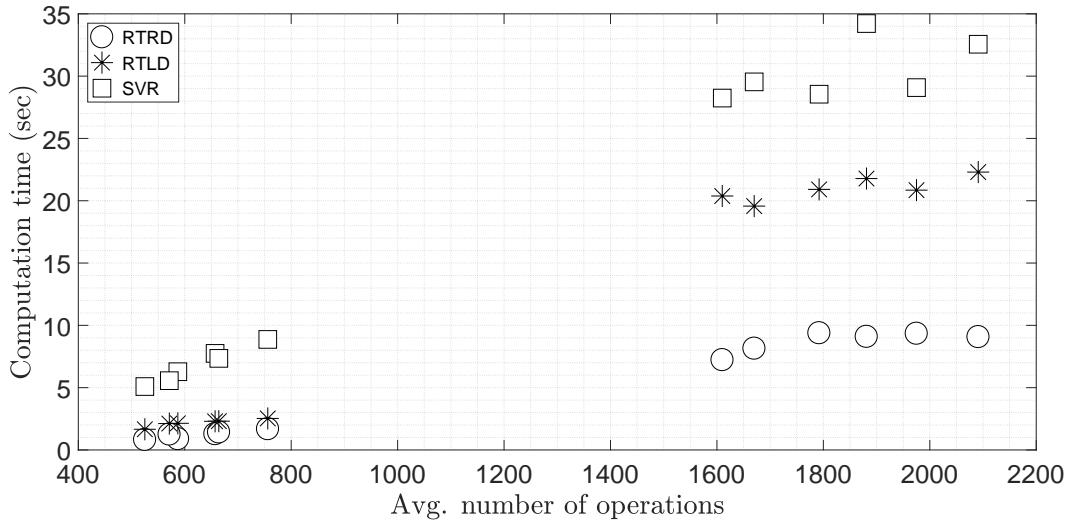


Figure 6.9: Computation time of RTRD, RTLD, and SVR according to the average number of operations.

experimental results, clearly, the proposed approaches excelled SVR in terms of performance and computation time.

Meanwhile, according to Figure 6.9, RTLD involves a much larger increase in the computation time when the number of operations increases compared to the RTRD. This result can be explained by the difference in the number of alternatives between the two dispatchers when a dispatching decision is required. In detail, although the scale of the RMLs becomes large, the number of alternatives of RTRD is always equal to that of dispatching rules used in the training phase while RTLD has the same number of alternatives as the number of candidate lots. Because of this, as the number of operations increases, the difference is likely to be larger in the computation time between the two dispatchers.

### 6.3.2 Performance differences according to weights

In this section, experiments were carried out to investigate how the performances of RTRD and RTLD change according to the values of weights in Equations (5.5) and (5.9). In Figures 6.10, 6.11, 6.12, and 6.13, contour lines visualize the performances of RTRD against SVR depending on values of  $\lambda_w$  and  $\lambda_l$  in datasets 4, 9, 10, and 11, respectively. The detailed results for each dataset are presented in Appendix B. The value in each figure means the performance measure of RTRD divided that of SVR marked in percentage. Specifically, values less than 100 indicate that RTRD outperforms SVR, and values greater than 100 represent the opposite. The X symbol indicates the minimum value of the performance measure.

For most datasets, the minimum values of  $AWT$  and  $AIT$  were observed at the lower right and upper left, respectively. This result is in line with our expectation that the larger the value of  $\lambda_w$  is, the more likely it is to make a decision for reducing the waiting time, and the larger the value of  $\lambda_l$ , the more likely it is to make a decision for decreasing the idle time. However, in datasets 4, 9, 10, and 11, the minimum value of  $AIT$  was obtained when the value of  $\lambda_w$  was close to 1 (shown in Figures 6.10b, 6.11b, 6.12b, and 6.13b). Furthermore, according to Figure 6.11a, the minimum value of  $AWT$  was achieved when the value of  $\lambda_l$  was close to 1. This rather contradictory result might be explained by the fact that there exists some room for improvement in terms of the waiting time.

From a viewpoint of  $ALT$ , the X symbol of each dataset was observed at different locations. In addition, the locations, where the performance of RTRDs was worse than that of SVR, were found according to the values of  $\lambda_w$  and  $\lambda_l$ . These findings suggest that setting weight values according to the characteristics of the problem is

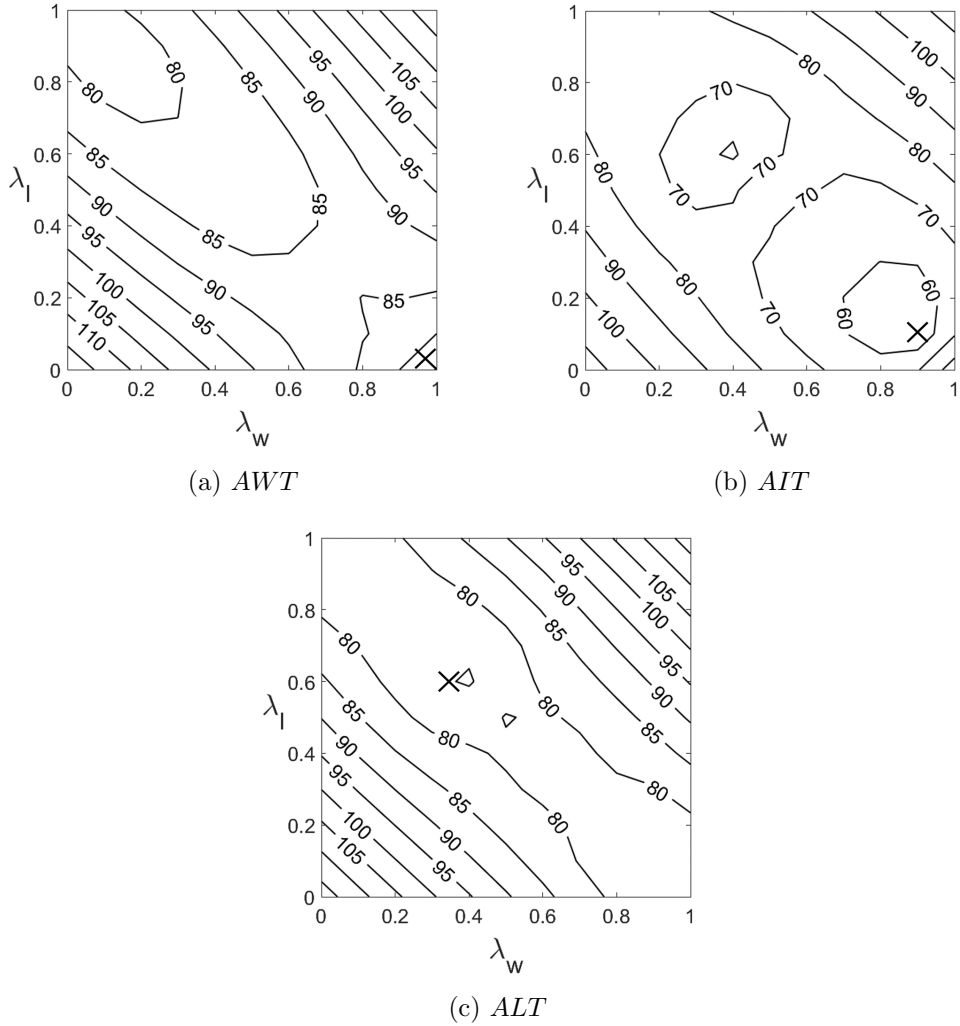


Figure 6.10: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 4.

crucial to the performances of RTRD.

In Figures 6.14, 6.15, 6.16, and 6.17, contour lines visualize the performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in datasets 3, 6, 9, and 12, respectively. The detailed results for each dataset are presented in Appendix C.

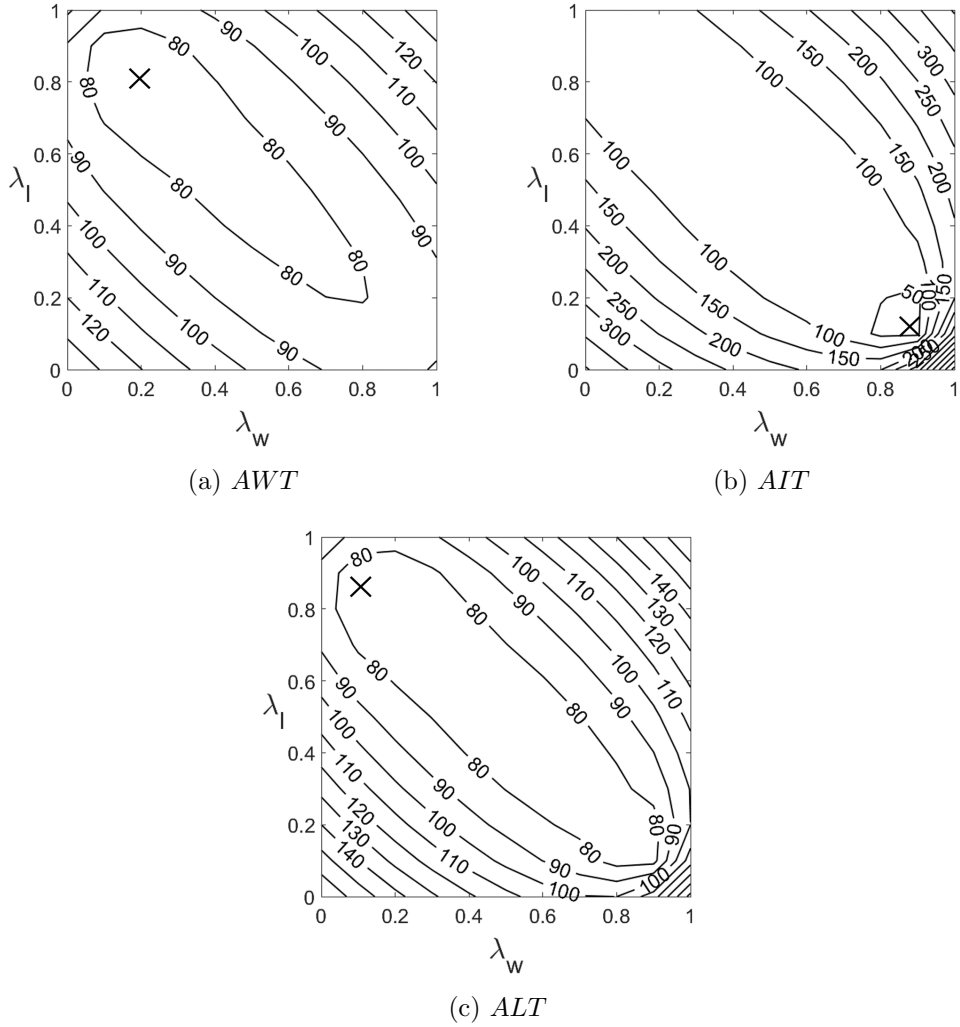


Figure 6.11: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 9.

The value in each Figure means the performance measure of RTRD divided that of SVR marked in a percentage. Specifically, values less than 100 indicate that RTRD outperforms SVR, and values greater than 100 represent the opposite. The X symbol indicates the minimum value of the performance measure.

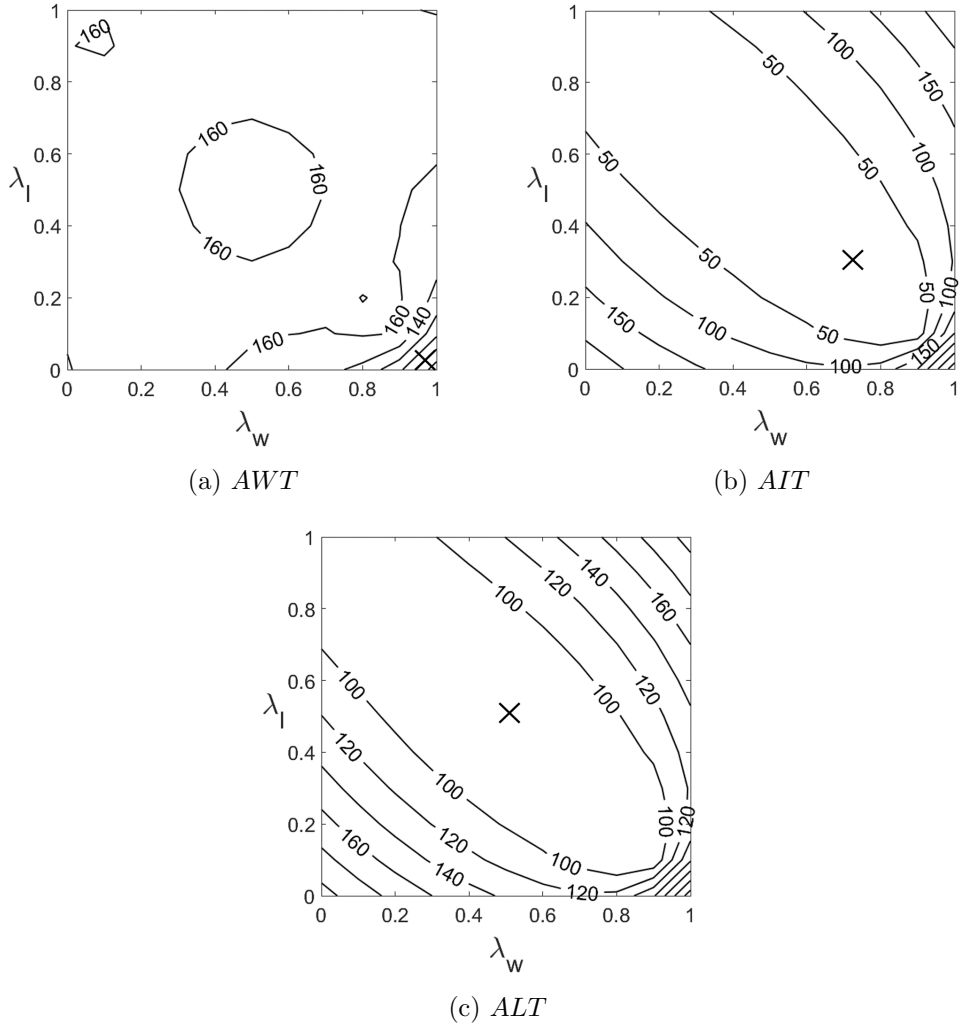


Figure 6.12: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 10.

For most datasets, the minimum values of *AWT* were mostly observed at the top of the triangle where the value of  $\lambda_d$  is close to 1. This result suggests that  $score_d$  is a key factor in the difference of *AWT* between RTRD and RTLD. In Figure 6.17a, X symbol is observed at the lower right corner where the value of  $\lambda_l$  is larger and

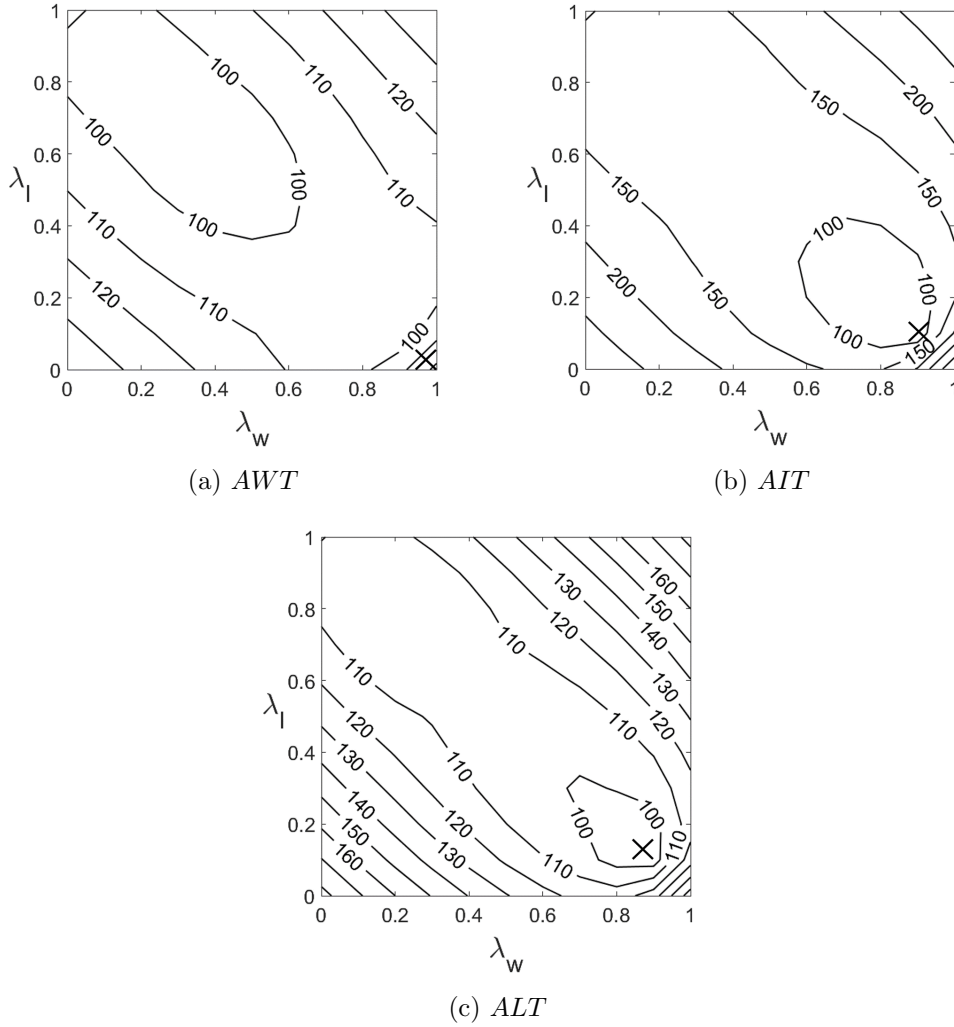
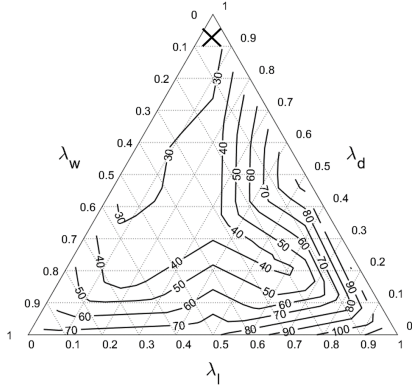


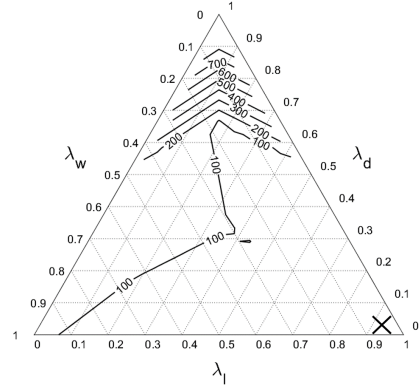
Figure 6.13: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 11.

the values of  $\lambda_w$  and  $\lambda_d$  are smaller. This finding was an unexpected result, but the minimum value of this location is approximately identical to the value at the top of the triangle.

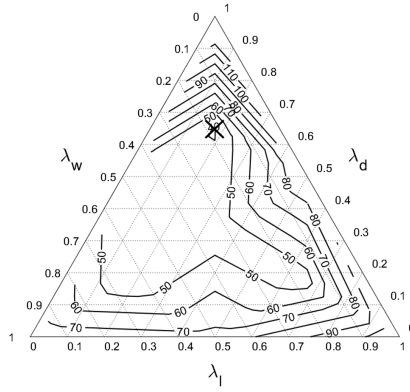
On the other hand, the minimum values of *AIT* were observed at the mid-bottom



(a) *AWT*



(b) *AIT*

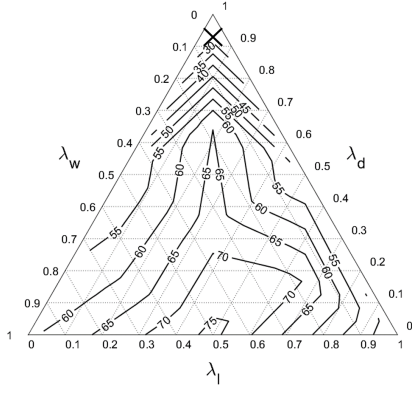


(c) *ALT*

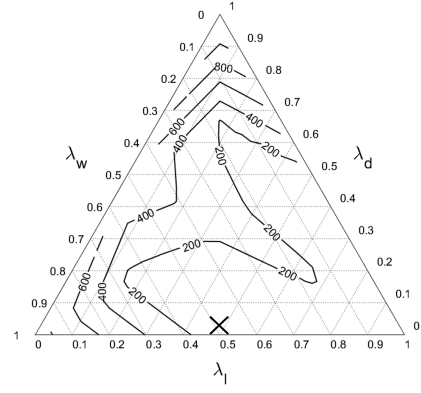
Figure 6.14: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 3.

( $\lambda_w = 0.5$  and  $\lambda_l = 0.5$ ), the middle of the right side ( $\lambda_d = 0.5$  and  $\lambda_l = 0.5$ ), and the lower right corner ( $\lambda_l = 1$ ). Judging from this result, it can be seen that the minimization of *AIT* is achieved by dispatching decisions considering both the waiting and idle time depending on the datasets.

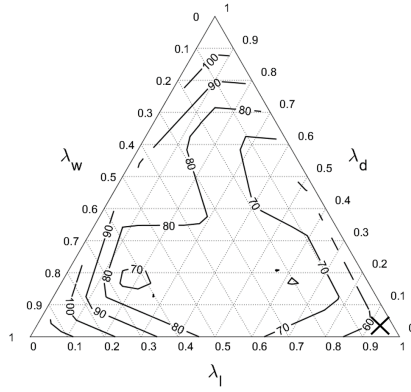




(a) *AWT*



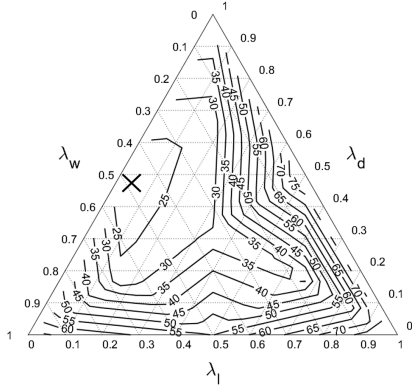
(b) *AIT*



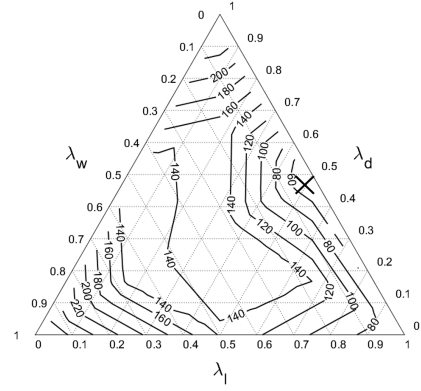
(c) *ALT*

Figure 6.15: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 6.

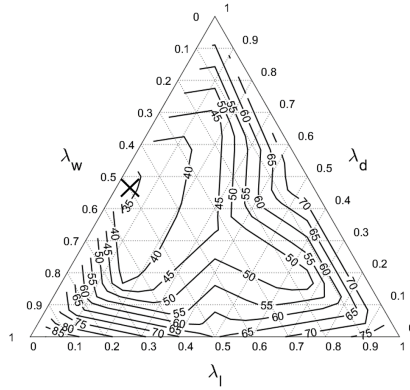
In terms of *ALT*, the X symbol of each dataset was observed at different locations as in RTRD. A few locations where the performance of RTLDs was lower than that of SVR were revealed according to the values of  $\lambda_w$  and  $\lambda_l$ . Surprisingly, the minimum value in Figure 6.16c was yielded on the mid-left side of the triangle ( $\lambda_w = 0.5$  and



(a) *AWT*



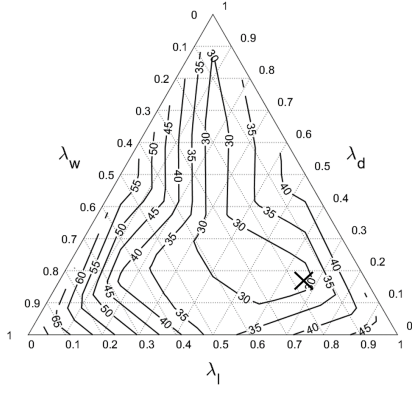
(b) *AIT*



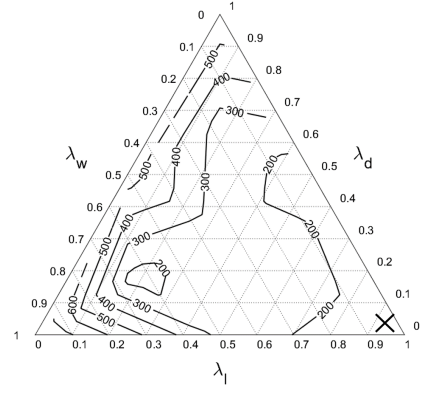
(c) *ALT*

Figure 6.16: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 9.

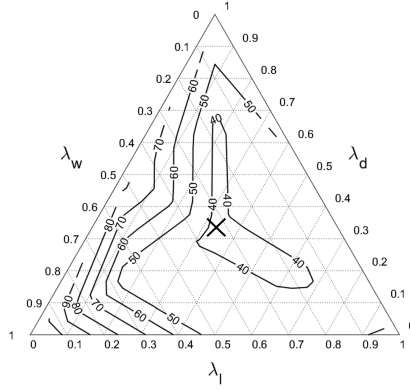
$\lambda_d = 0.5$ ). This result may be explained by the fact that dataset 9 has the larger number of operations per resource than other datasets. Because of this, in dataset 9, the waiting time increases greatly in order to reduce the idle time. As shown in the experiments on RTRD, the combination of findings implies the importance of



(a) *AWT*



(b) *AIT*



(c) *ALT*

Figure 6.17: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 12.

setting weight values according to the characteristics of the dataset.

### 6.3.3 Robustness test

For the purpose of evaluating the robustness in performances of the proposed dispatchers, we applied the RTDs generated by training data of each dataset to the

real-time dispatching phase of the dispatching problems that belong to different datasets. Table 6.5 shows statistically significant differences in *ALT* between RTRDs trained in different datasets by using the *t*-test at the 5% level presented. Each cell presents the result of applying dispatching methods to the dataset corresponding to the column which contains the cell. For example, for the first row, the problems of datasets 2 to 12 were solved with the use of the RTRD generated from the dataset 1. The calculated performances were then compared to those of the RTRDs generated from datasets 2 to 12.

In detail, the bottom value in the Table 6.5 indicates the corresponding *p*-value. The shaded cells signify that there is no statistically significant difference in performance between the two RTRDs corresponding to each cell. The top value in each cell means the *ALT* of RTRD generated from the dataset corresponding to the row divided that of SVR marked in percentage. Specifically, values less than 100 indicate that the RTRD generated from the dataset corresponding to the row outperforms SVR, and values greater than 100 with an asterisk (\*) mean the opposite.

As specified in Table 6.5, the RTRDs generated from datasets 11 and 12 show the best performance in terms of robustness, and RTRDs generated from datasets 4 to 7 were robust for one dataset only. Surprisingly, although the RTRDs generated from datasets 5 and 6 did not achieve robustness successfully, they outperformed the others generated from datasets and SVR in terms of *ALT*. These results did not provide any obvious trends in terms of robustness between datasets. It is difficult to explain the results, suggesting that there might be other important characteristics in classifying problems except for the criteria considered in this thesis.

Table 6.5: Statistically significant differences in *ALT* between RTRDs trained in different datasets.

Dataset No.	Dataset No.											
	1	2	3	4	5	6	7	8	9	10	11	12
1	76.12	75.91	71.6	57.43	63.79	65.18	84.31	78.41	73.14	71.26	78.55	60.27
	-	5.25E-13	2.38E-02	4.22E-07	2.06E-03	3.83E-01	5.87E-07	7.81E-03	9.78E-01	8.81E-04	1.04E-08	7.13E-01
2	105.67*	104.8*	87.5	72.9	89.59	80.99	113.92*	99.47	82.97	95.65	99.54	69.13
	1.05E-10	-	7.12E-04	5.11E-01	6.71E-20	5.00E-10	3.22E-01	4.65E-05	2.45E-05	3.81E-02	9.95E-01	4.27E-08
3	98.64	87.27	78.35	57.62	67.84	67.09	125.82*	102.71*	78.16	87.85	86.81	63.69
	1.22E-05	4.71E-06	-	1.10E-08	1.77E-05	8.39E-01	3.60E-03	1.12E-06	4.30E-02	5.89E-01	4.79E-04	1.51E-01
4	111.52*	106.92*	91.77	75.19	75.38	73.15	124.38*	107.86*	82.81	108.92*	101.45*	61.56
	3.53E-13	5.96E-01	2.96E-05	-	2.83E-12	1.41E-03	1.73E-03	1.89E-09	1.17E-04	6.09E-06	5.71E-01	6.60E-01
5	72.32	68.06	58.66	52.67	57.51	57.58	77.04	70.76	57.53	65.3	68.62	46.64
	3.54E-01	4.46E-17	2.64E-12	2.93E-11	-	1.39E-05	3.31E-10	2.77E-06	5.18E-10	3.76E-06	2.47E-13	1.89E-13
6	86.44	84.12	73.88	67.4	71.51	66.55	89.52	76.91	66.38	72.72	74.31	53.1
	7.34E-03	2.37E-07	6.12E-02	6.10E-03	5.87E-11	-	3.99E-06	5.97E-04	4.58E-04	2.00E-04	1.34E-17	4.00E-10
7	115.42*	108.95*	93.98	85.18	94.45	85.32	108.84*	104.45*	86.42	98.07	109.12*	71.29
	5.21E-17	2.52E-01	2.28E-08	6.14E-04	1.03E-37	5.85E-17	-	2.27E-08	1.02E-09	1.35E-04	1.68E-04	1.60E-13
8	91.57	90.55	89.6	66.52	73.93	78.52	93.3	86.72	76.23	81.85	84.57	57.61
	1.58E-04	8.87E-05	6.08E-04	2.03E-03	5.99E-16	4.11E-05	5.52E-05	-	1.33E-01	2.62E-01	1.02E-08	6.18E-03
9	98.69	94.61	81.70	67.95	76.22	72.75	100.52*	90.59	73.21	73.29	82.27	57.16
	2.30E-08	3.75E-03	1.90E-01	9.36E-03	1.70E-16	7.37E-04	3.75E-02	1.36E-01	-	1.58E-04	1.27E-10	1.19E-03
10	99.93	114.98*	116.83*	79.10	100.36*	100.26*	91.26	104.09*	107.66	85.47	96.52	78.09
	3.30E-07	1.20E-02	8.39E-15	1.91E-01	1.41E-20	6.58E-12	7.08E-05	2.35E-05	6.48E-13	-	4.03E-01	3.24E-08
11	100.47*	88.78	81.00	76.48	89.09	74.54	95.18	82.50	73.63	92.66	99.56	78.18
	5.81E-09	2.76E-05	3.64E-01	6.70E-01	1.44E-16	2.99E-02	7.53E-04	9.34E-02	8.81E-01	3.49E-02	-	3.91E-04
12	111.19*	103.99*	95.73	79.48	90.39	80.66	104.09*	98.09	84.84	89.70	91.49	60.83
	3.00E-14	8.22E-01	3.04E-07	1.45E-01	2.68E-23	4.79E-07	2.41E-01	7.24E-05	7.18E-06	2.19E-01	2.85E-03	-

Table 6.6 contains the results when the same test as in Table 6.5 was carried out on RTLDs, and the meaning of each value in this table is the same as Table 6.5. Unlike the results for RTRD, RTLD generated from dataset 7 showed robustness in terms of *ALT* for most datasets. Although RTLD generated from dataset 1 failed to achieve robustness, it excelled SVR and RTLDs generated from each dataset except datasets 2 and 12.

Although RTLD generated from dataset 6 achieved robustness for datasets 7, 8, and 9, it resulted in lower performances for datasets 10 and 11 than SVR. However, except for these cases, the values of *ALT* yielded in all other cases were smaller than those of SVR. Furthermore, the number of shaded cells in Table 6.6 is 28% greater than that of the shaded cells in Table 5.1. These results show that RTLD was superior to RTRD in terms of both performance and robustness.

Table 6.6: Statistically significant differences in *ALT* between RTLDs trained in different datasets.

Dataset No.	Dataset No.											
	1	2	3	4	5	6	7	8	9	10	11	12
1	53.86	51.41	39.41	46.61	48.63	42.27	60.02	49.80	35.31	61.01	61.27	39.11
	-	1.96E-02	3.43E-02	1.49E-14	1.84E-04	5.80E-31	1.68E-02	1.19E-02	1.11E-04	5.51E-03	9.19E-04	2.39E-01
2	52.10	46.98	40.62	50.87	50.30	44.36	56.40	45.59	31.95	61.93	57.03	33.98
	3.37E-01	-	3.12E-01	3.19E-13	1.64E-03	3.67E-30	6.14E-05	7.64E-06	1.10E-06	1.26E-02	3.99E-12	1.22E-06
3	50.07	51.51	42.15	52.19	59.46	50.10	70.74	50.00	37.27	65.97	61.88	34.65
	3.71E-02	6.74E-03	-	9.24E-13	7.59E-01	5.29E-25	2.20E-01	3.62E-02	1.76E-03	6.00E-01	1.89E-03	4.39E-05
4	69.18	68.92	55.86	95.20	91.35	71.48	76.71	64.19	45.08	93.74	73.38	42.16
	7.58E-05	1.05E-06	1.11E-03	-	2.92E-06	1.18E-01	2.91E-02	2.06E-02	8.67E-01	3.39E-06	7.82E-02	3.88E-02
5	62.03	68.16	67.89	54.72	60.36	69.58	61.44	53.29	41.30	89.31	88.49	53.99
	1.38E-02	2.73E-10	7.04E-10	7.77E-12	-	6.34E-03	3.93E-02	6.97E-01	1.20E-01	1.28E-11	1.62E-10	1.58E-12
6	71.02	63.08	55.22	85.13	90.85	79.64	68.29	55.52	43.30	111.17*	104.12*	58.36
	2.09E-07	1.55E-05	3.54E-05	4.53E-02	9.42E-12	-	6.81E-01	5.72E-01	4.29E-01	2.88E-27	7.83E-15	3.31E-14
7	51.26	48.38	42.57	49.04	54.37	47.23	66.93	55.01	38.38	68.58	65.32	36.27
	1.05E-01	4.08E-01	7.66E-01	8.24E-14	5.71E-02	1.07E-27	-	6.15E-01	5.01E-03	4.33E-01	3.36E-01	2.07E-02
8	47.53	43.70	35.47	51.80	49.05	40.85	67.01	54.21	36.38	80.93	86.24	46.13
	1.75E-04	3.36E-02	2.20E-06	6.74E-13	3.46E-04	1.10E-31	9.79E-01	-	5.56E-04	4.25E-06	1.69E-17	1.12E-12
9	49.02	50.08	39.89	54.40	62.61	52.58	84.78	58.47	45.59	71.79	69.16	41.04
	1.33E-02	1.15E-01	1.82E-01	5.17E-12	4.69E-01	1.78E-23	4.45E-06	5.17E-02	-	2.42E-02	1.42E-01	2.51E-03
10	69.29	57.12	48.53	57.83	52.87	42.14	65.71	48.48	36.83	67.02	61.10	37.06
	2.39E-05	9.98E-04	2.64E-02	2.39E-10	4.11E-02	2.29E-32	6.75E-01	8.82E-03	1.92E-03	-	2.32E-04	3.06E-01
11	55.12	53.46	44.40	53.75	56.97	56.26	56.33	45.81	38.59	70.49	66.60	42.13
	4.67E-01	3.04E-04	1.22E-01	3.09E-12	2.67E-01	2.94E-19	1.79E-05	1.01E-06	1.07E-02	1.13E-01	-	9.62E-05
12	68.94	67.61	58.58	52.76	53.97	47.27	73.51	57.13	43.45	71.64	65.62	38.06
	3.49E-06	1.16E-07	2.48E-06	3.26E-12	1.18E-01	6.92E-16	8.60E-02	2.81E-01	4.52E-01	3.78E-02	5.89E-01	-

## Chapter 7

### Conclusions

#### 7.1 Summary and contributions

RML is the manufacturing line where parts can make several visits to the same stage before the parts complete all operations assigned. Recently, with the emergence of semiconductor manufacturing and thin film transistor-liquid crystal display (LCD) manufacturing lines, RMLs causes wide concern in both academia and industry. Due to the frequently re-entrant parts between multiple stages in RMLs, it is challenging to achieve both goals of reducing the flow time and increasing the utilization of resources at the same time.

In order to decrease the flow time without loss in resource utilization of the bottleneck stage for real-world RMLs, this thesis proposes a novel approach for DNN based RTDs. First, the DEBS and monitoring tool were implemented to generate training data and evaluate the performance of dispatching decisions. In addition to imitating real-world RMLs, DEBS is in charge of generating simulation logs to be converted into the training data for RTDs. The monitoring tool was designed to display a variety of information about RML at the time each dispatching decision being made. It provides not only the functionality of the existing Gantt chart, but also the ability to show various performance indicators over time.



Second, the thesis proposes two DNN based RTDs with different decision-making processes, called RTRD and RTLD. Whenever a dispatching decision is required, RTRD determines the best among the existing dispatching rules while RTLD calculates scores on all candidate parts and assigns the part with the highest score to a resource.

To learn an efficient dispatching policy in RMLs considering intentional delays, we employed DNN which has the ability to capture complex non-linear dynamics. In the training phase, in order to obtain training data used for RTDs, all dispatching decisions of each training problem are executed randomly. The performances of decisions are then measured by the score generator, and the scored simulation logs are used by a learning algorithm to train DNNs embedded in RTDs.

The experimental results demonstrate that the proposed RTDs are successful in decreasing flow time and increasing the utilization of bottleneck resources at the same time. RTLD outperforms the existing dispatching methods in terms of the average loss time for all datasets considered. Meanwhile, through the weight adjustment experiments and robustness test, we confirmed the direction in which the proposed approaches could be further developed.

This thesis has made contributions as follows. First, it developed the monitoring tool that ensures the ability to investigate each dispatching decision, which is independent from the programming language used to implement the simulator. Second, a proposal was made regarding a novel method for generating training data for DNNs without a verified solver. Lastly, the proposed RTDs can perform effective dispatching decisions considering intentional delays in RMLs.

## 7.2 Limitations and future research

Although satisfactory results were obtained through this thesis, there exist some room for further improvement. First, RTRD is required to reduce the time parts spend in the re-entrant stocker while RTLD is supposed to improve in terms of decreasing computation time. If the concept of conflicting lots is introduced into RTRD, the performance of RTRD is highly likely to be improved. However, this attempt is less likely to be successful due to the nature of the RTRD's decision-making manner of not being able to specify a lot.

Second, we plan to investigate a reinforcement learning (RL) algorithm which is capable of finding a policy that maximizes global rewards in order to enable more intelligent dispatching decisions that minimize both the waiting and idle time. The dispatching rules representing the action vector of RTRD can be used to define the action space of the Q-learning which is the most representative of the RL. However, it is essential to design an immediate reward between the current state and next state, since the preference scores indicating the value of dispatching decisions are calculated only after all simulations are completed.

Lastly, further studies are needed to improve the robustness of the proposed dispatchers to the type of datasets such as the numbers of job types, resources, and operations. Furthermore, it is necessary to investigate a learning method for determining the weights between the flow time and resource utilization according to the characteristics of RMLs.

Meanwhile, the proposed method is applicable to the resource allocation problem in a cloud computing environment where it is required to efficiently assign jobs of different complexities to computing resources. The goal of the cloud computing

environment is similar to the objective function addressed by RTDs in that the completion time of each job is minimized and the utilization of computing resources is maximized. Therefore, if the computation time of RTDs is reduced and the suitable network structure for the cloud computing environment is studied, the proposed method will be able to achieve superior performance even in the new environment.

# Appendices

# Appendix A

## Performance comparison results

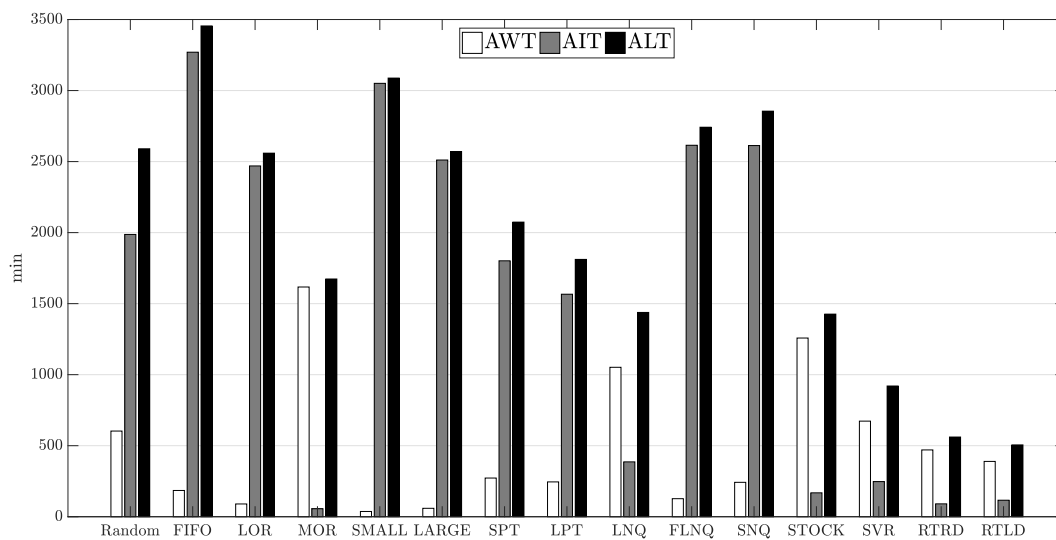


Figure A.1: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 1.

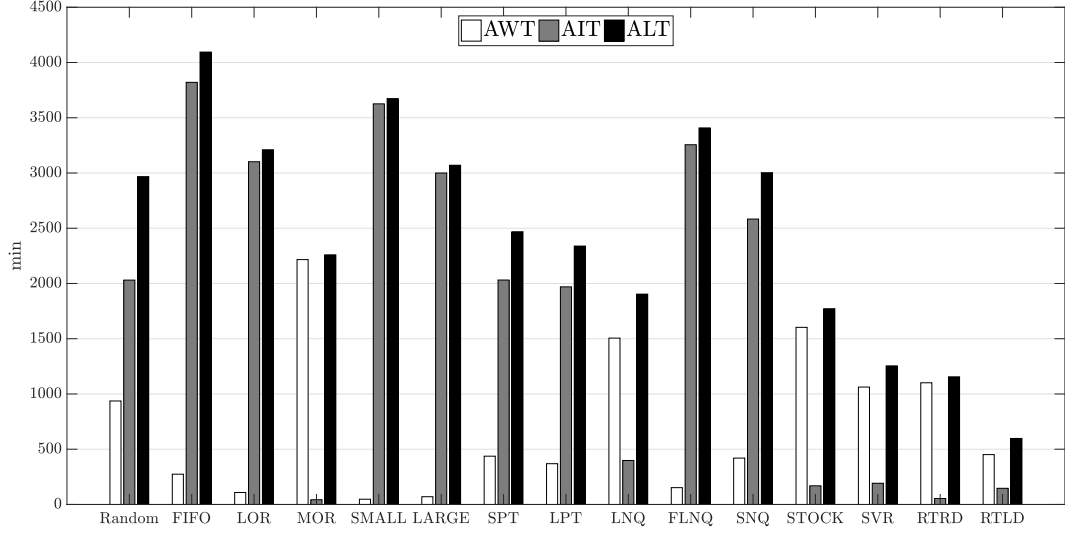


Figure A.2: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 2.

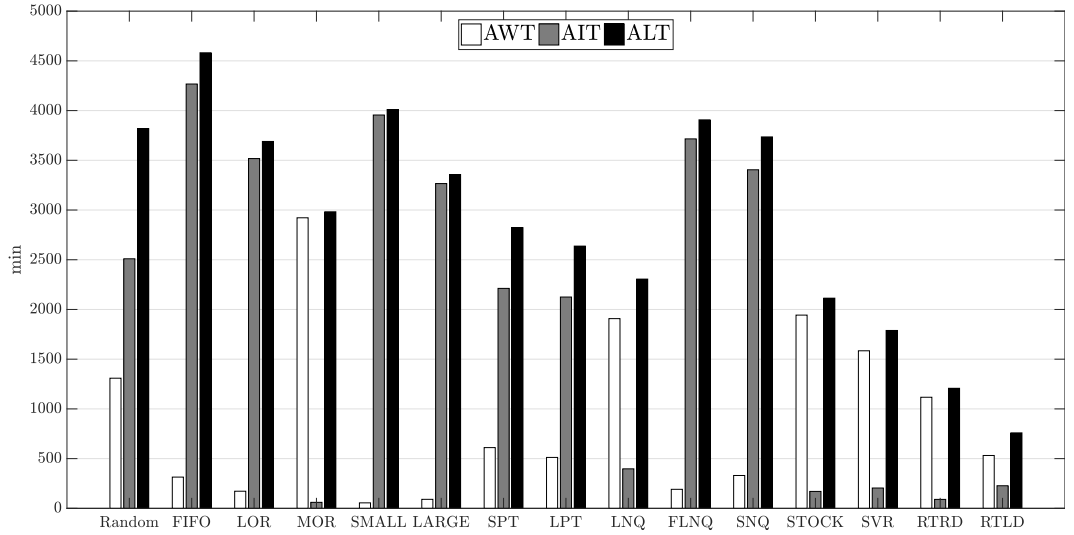


Figure A.3: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 3.

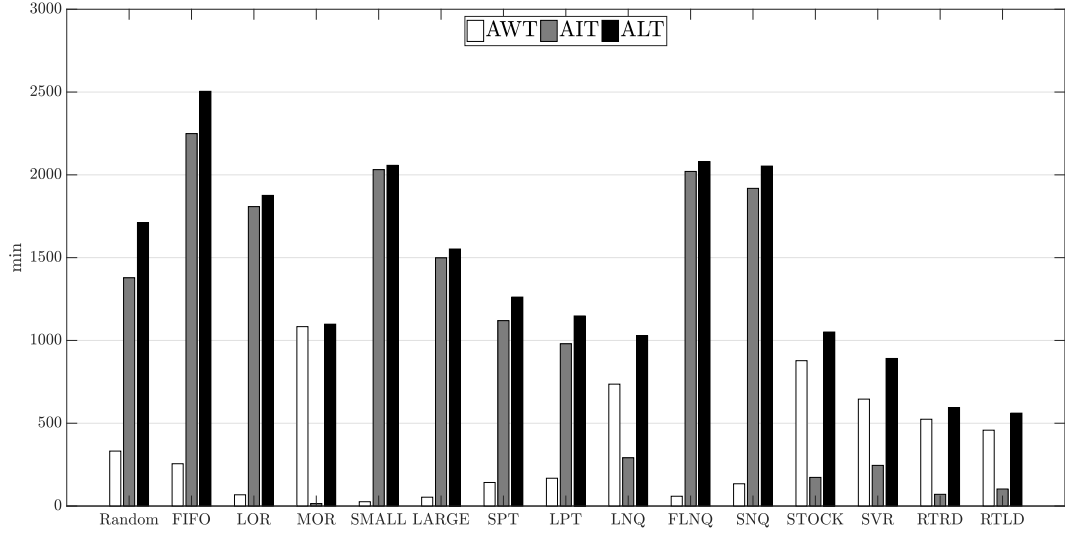


Figure A.4: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 4.

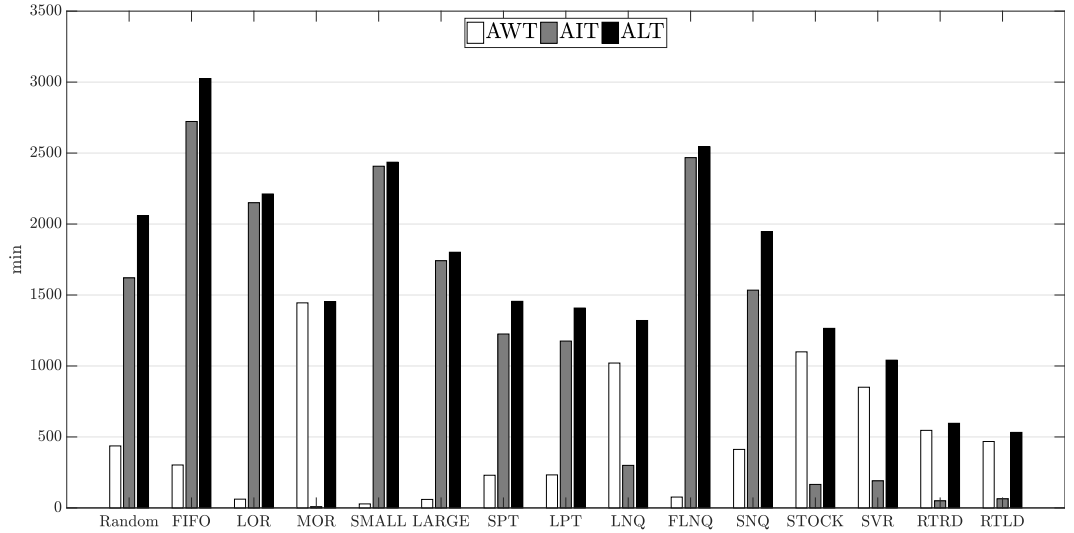


Figure A.5: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 5.

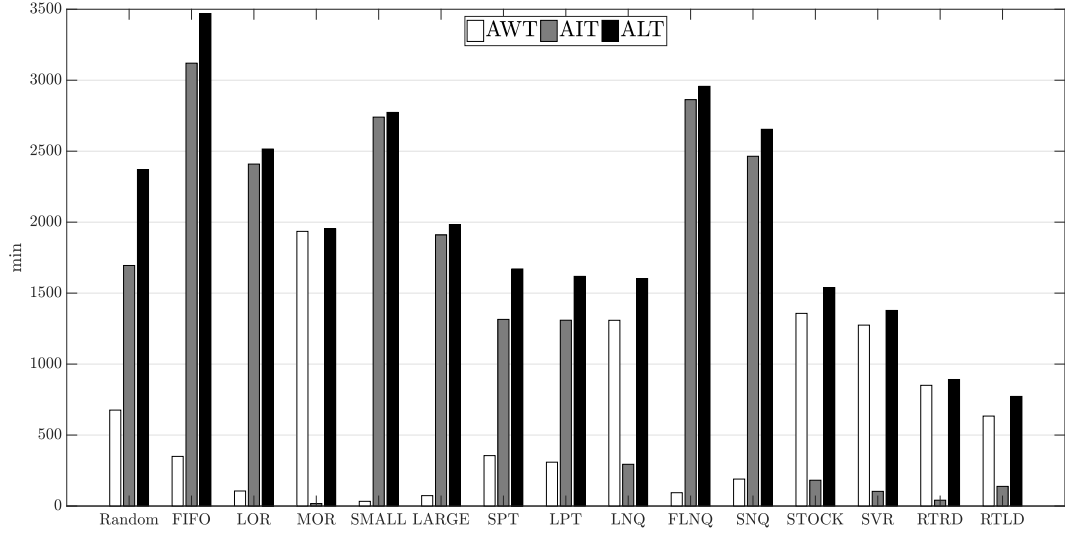


Figure A.6: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 6.

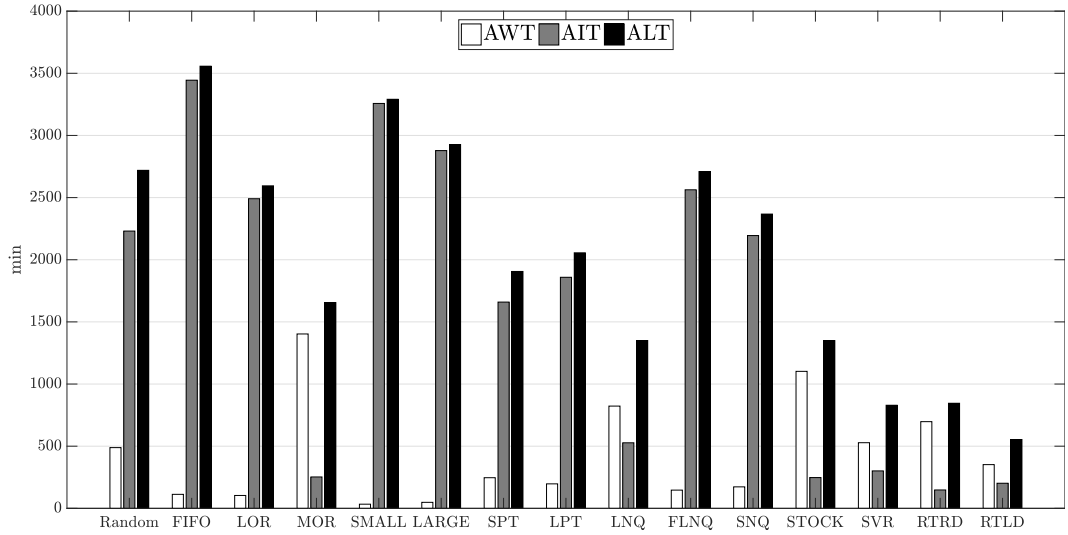


Figure A.7: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 7.



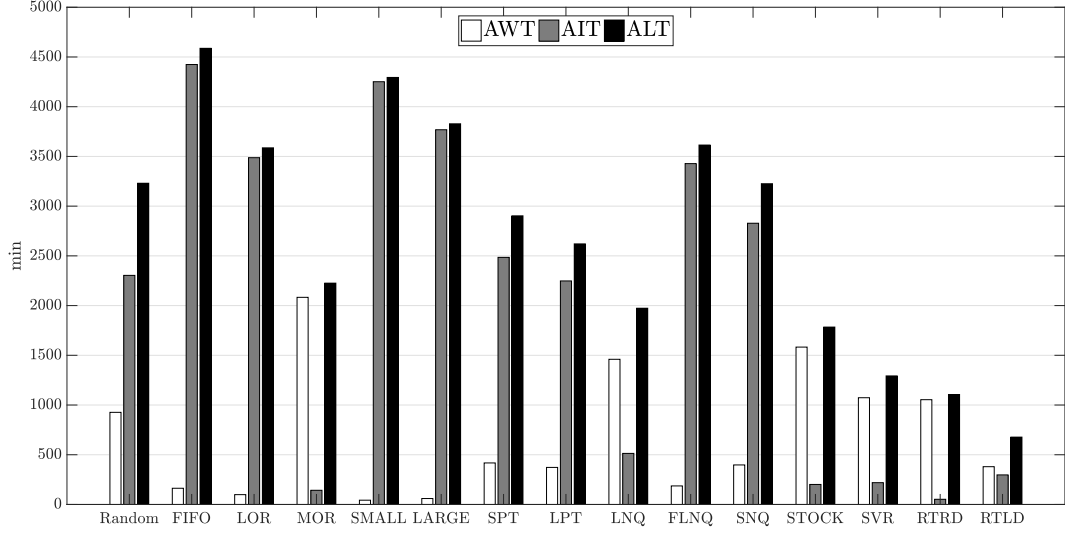


Figure A.8: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 8.

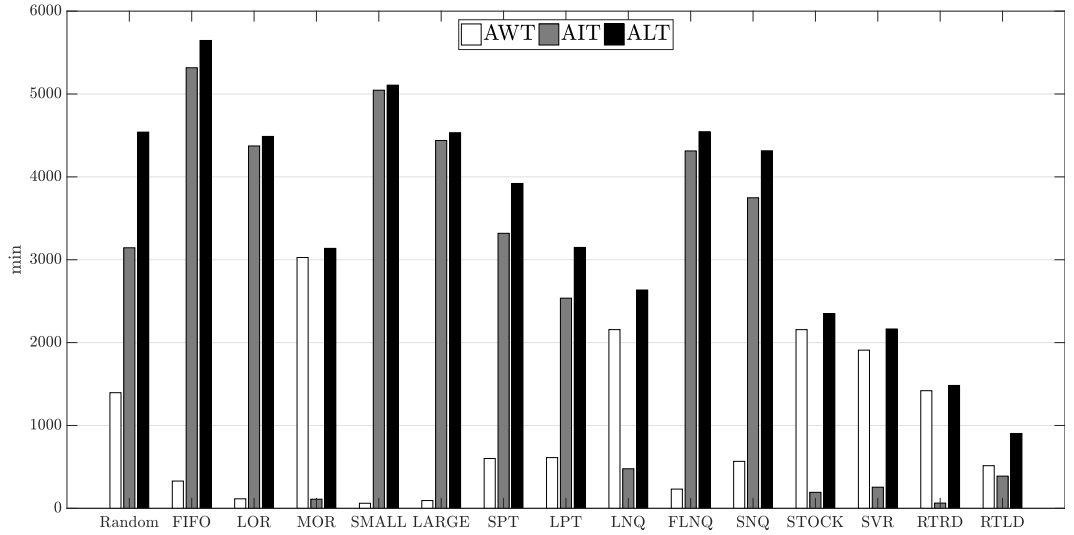


Figure A.9: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 9.

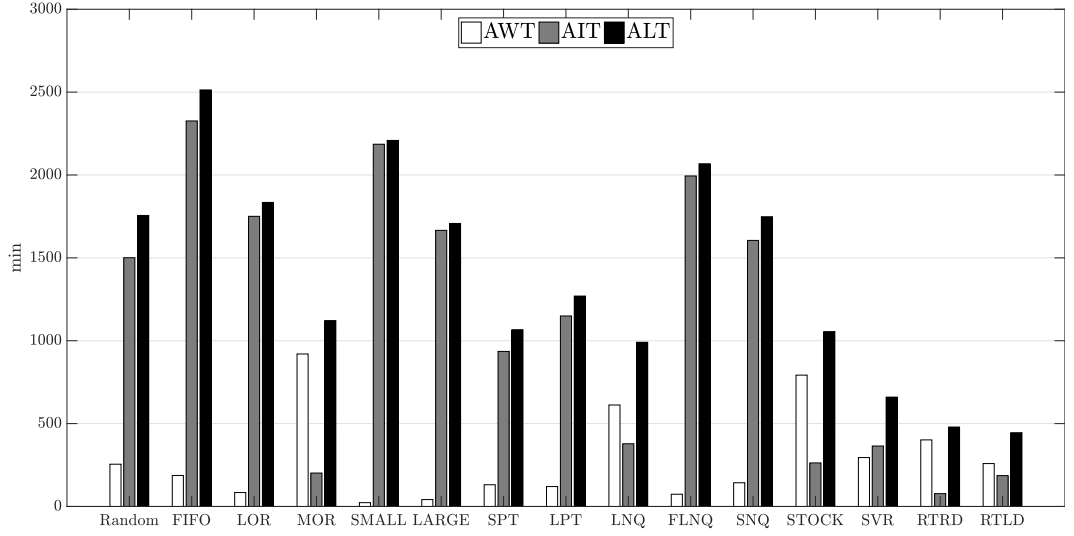


Figure A.10: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 10.

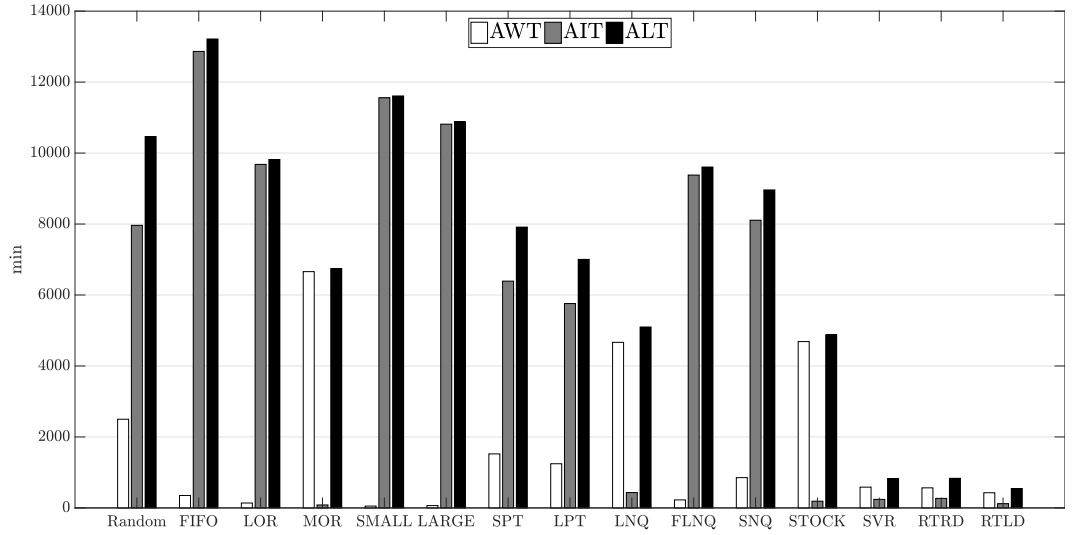


Figure A.11: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 11.

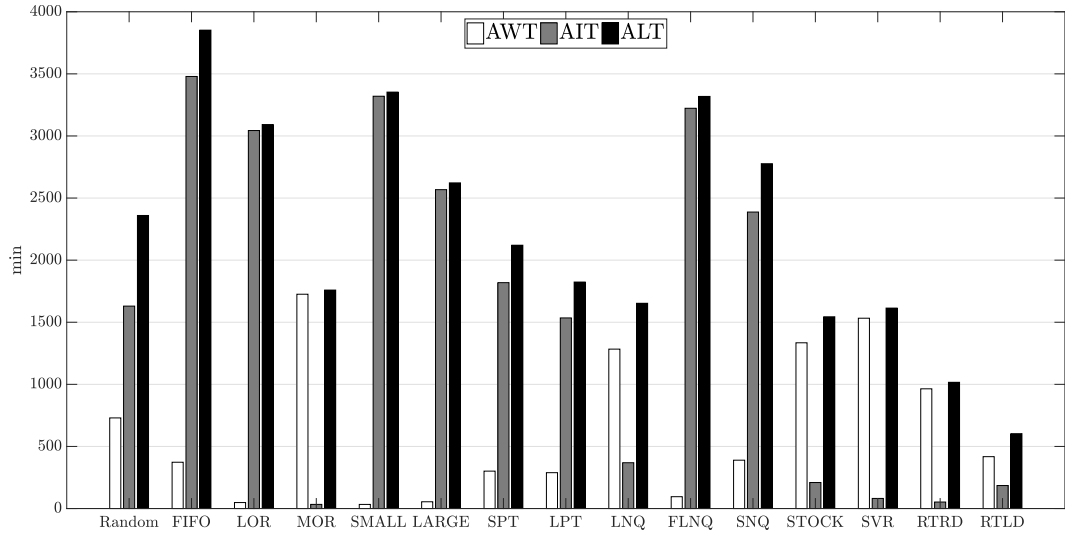
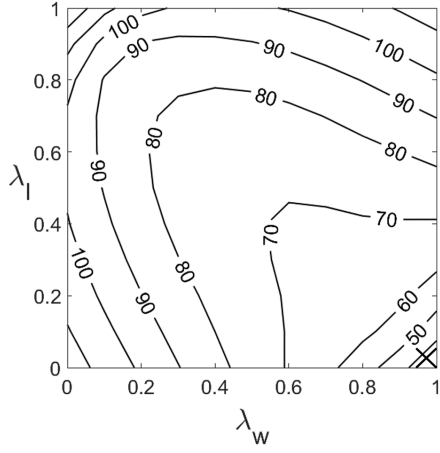


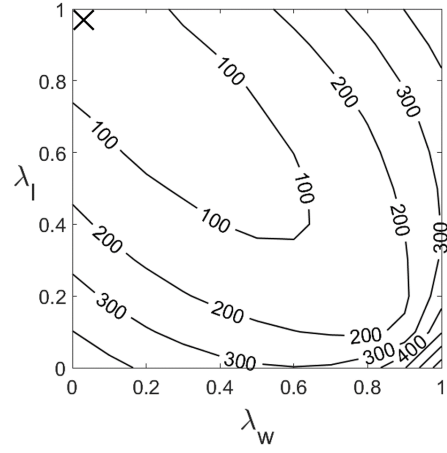
Figure A.12: *AWT*, *AIT*, and *ALT* results of the proposed dispatchers and the existing methods for dataset 12.

## Appendix B

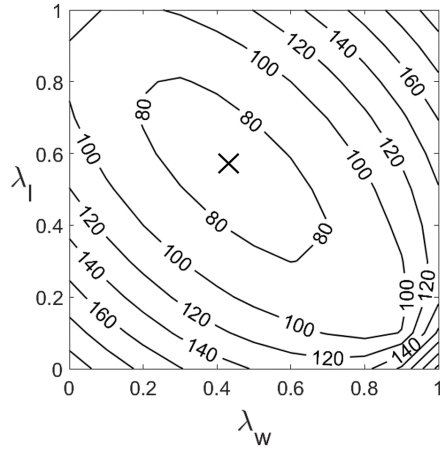
Performance contour of RTRD with respect to  $\lambda_w$   
and  $\lambda_l$



(a) *AWT*

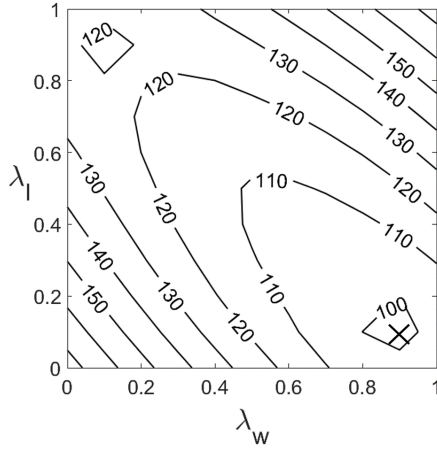


(b) *AIT*

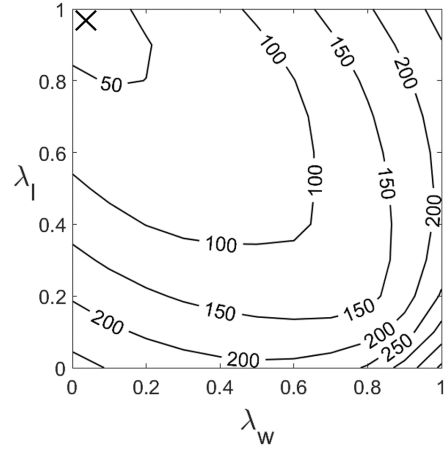


(c) *ALT*

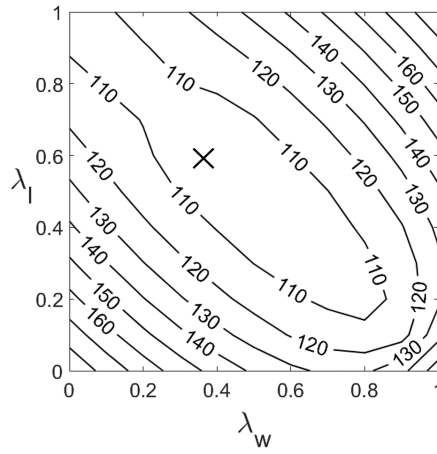
Figure B.1: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 1.



(a) *AWT*

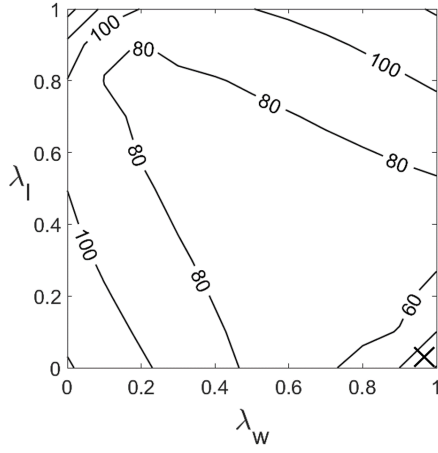


(b) *AIT*

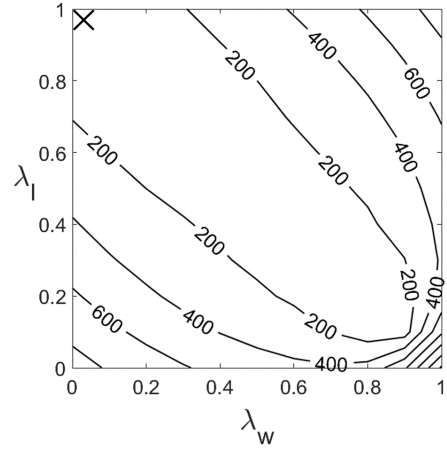


(c) *ALT*

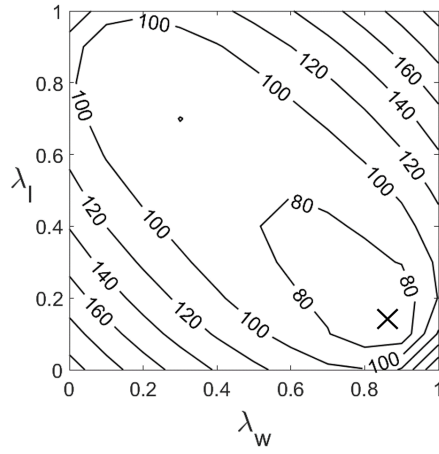
Figure B.2: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 2.



(a) *AWT*



(b) *AIT*



(c) *ALT*

Figure B.3: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 3.

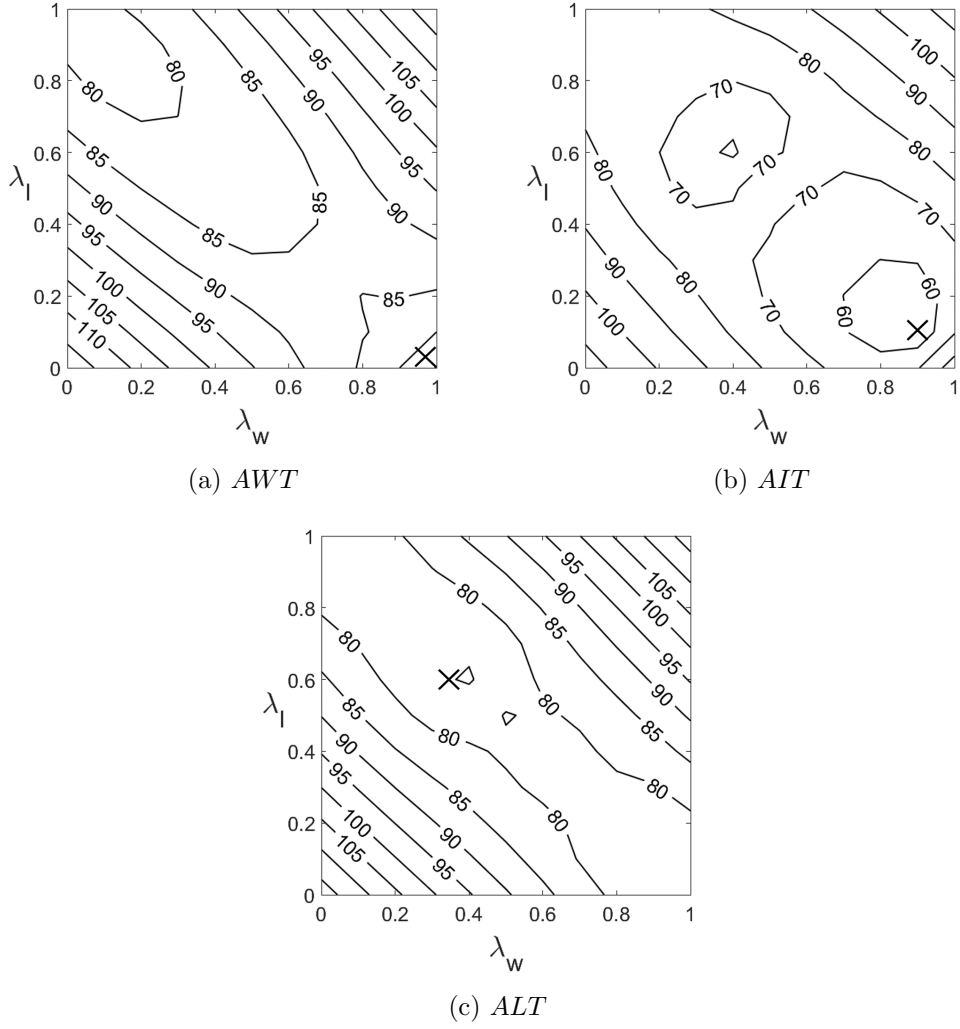
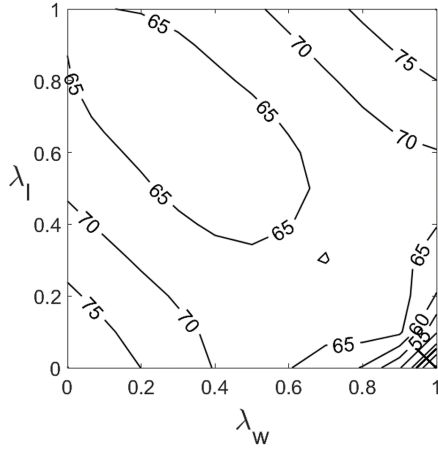
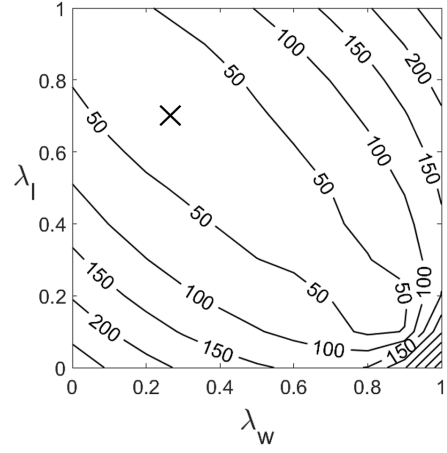


Figure B.4: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 4.

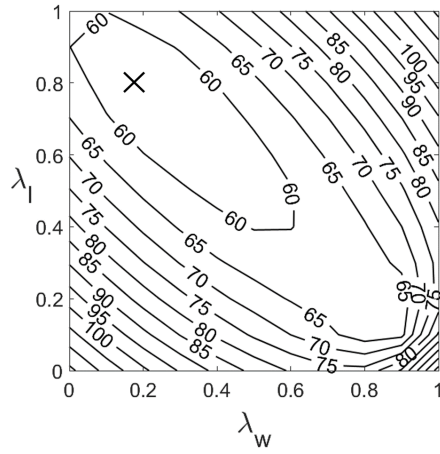




(a) *AWT*

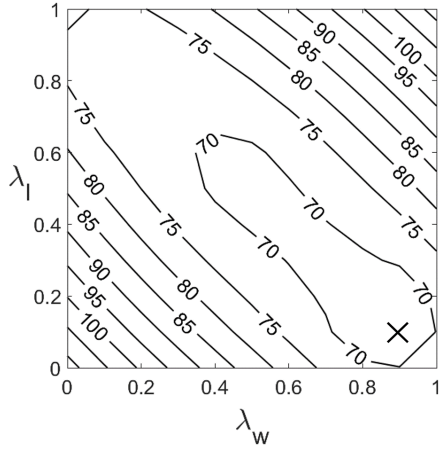


(b) *AIT*

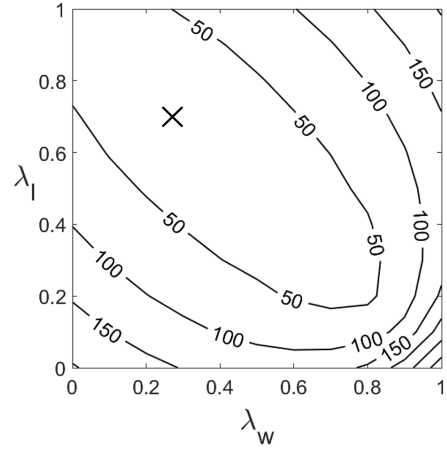


(c) *ALT*

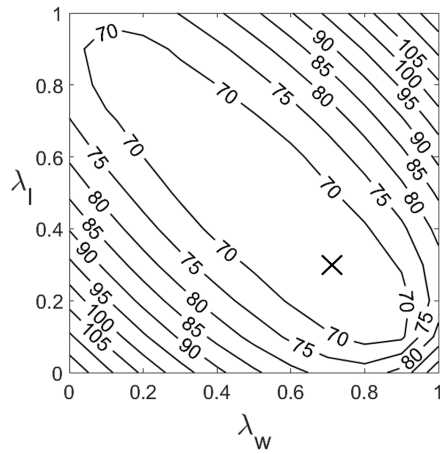
Figure B.5: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 5.



(a) *AWT*

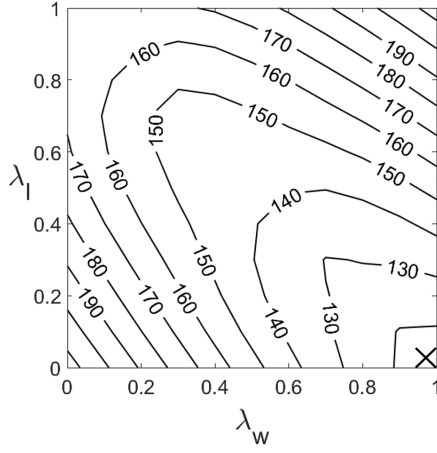


(b) *AIT*

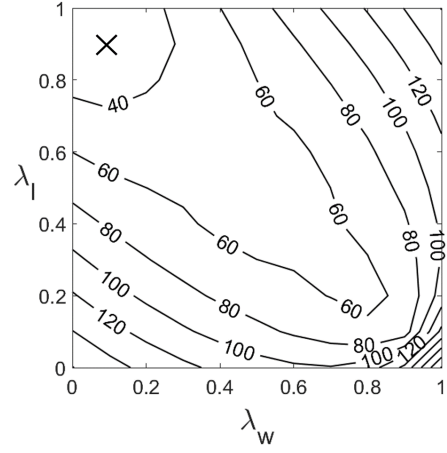


(c) *ALT*

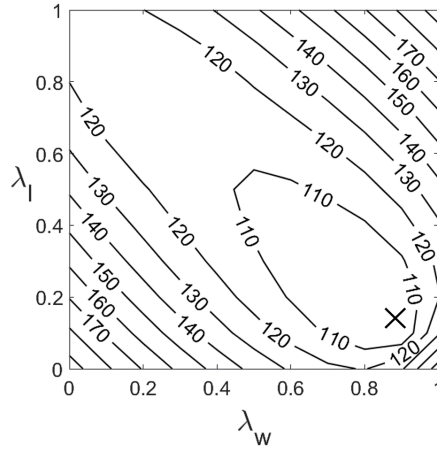
Figure B.6: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 6.



(a) *AWT*

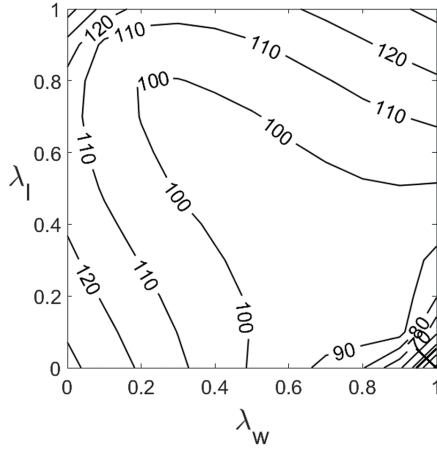


(b) *AIT*

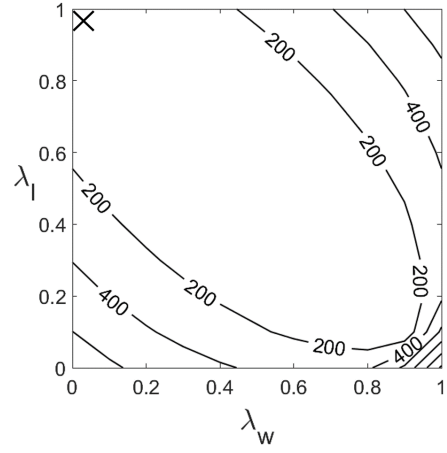


(c) *ALT*

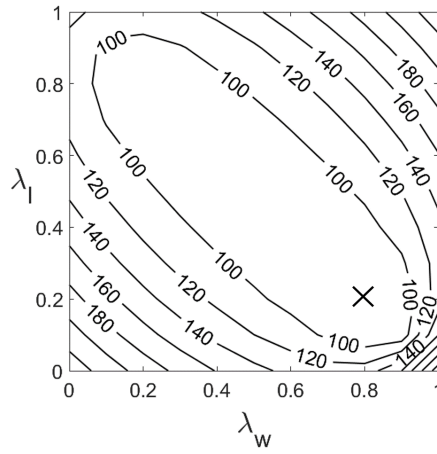
Figure B.7: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 7.



(a) *AWT*

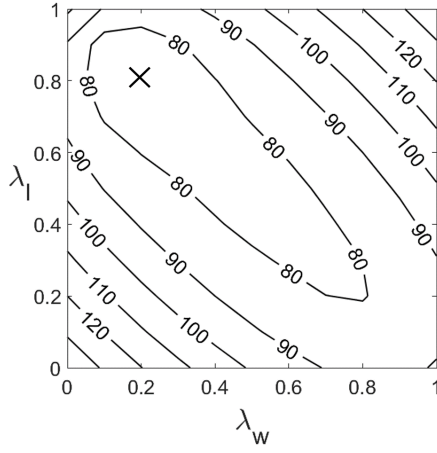


(b) *AIT*

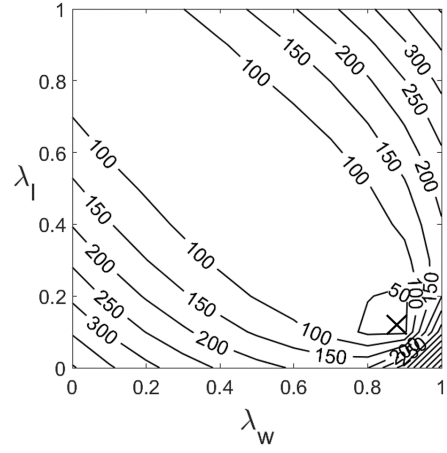


(c) *ALT*

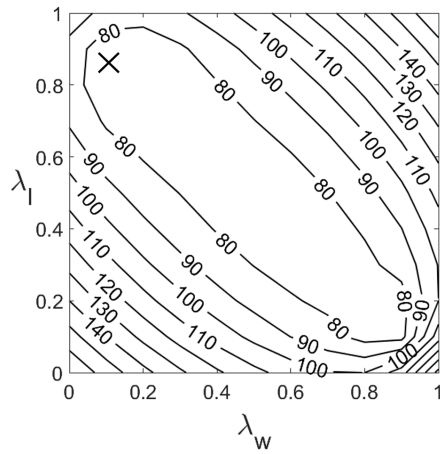
Figure B.8: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 8.



(a) *AWT*

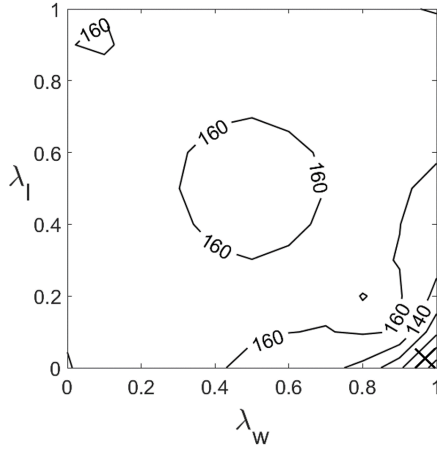


(b) *AIT*

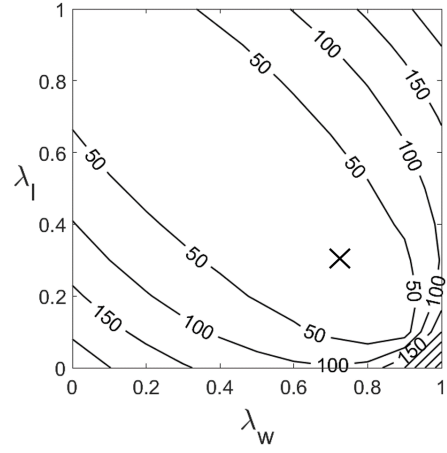


(c) *ALT*

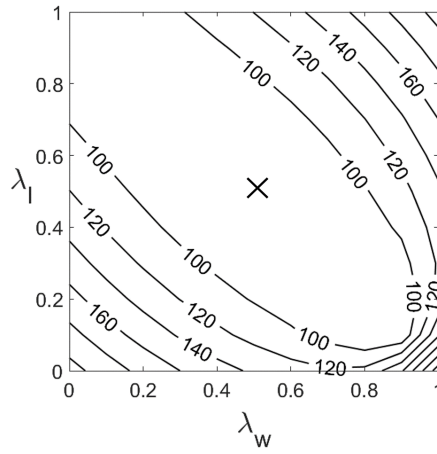
Figure B.9: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 9.



(a) *AWT*

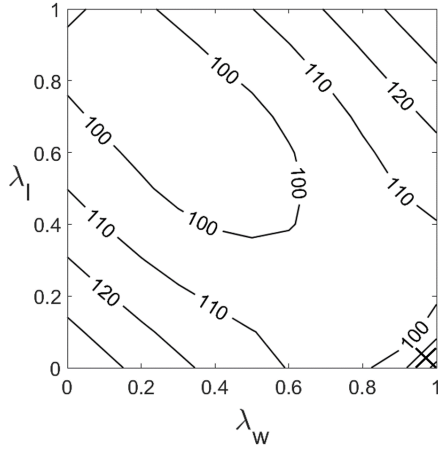


(b) *AIT*

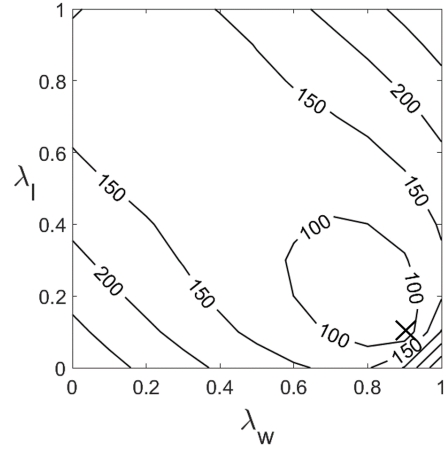


(c) *ALT*

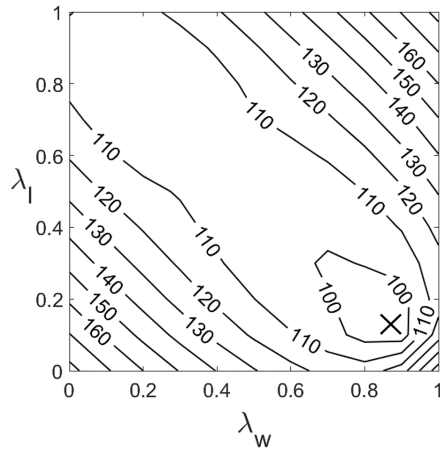
Figure B.10: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 10.



(a) *AWT*

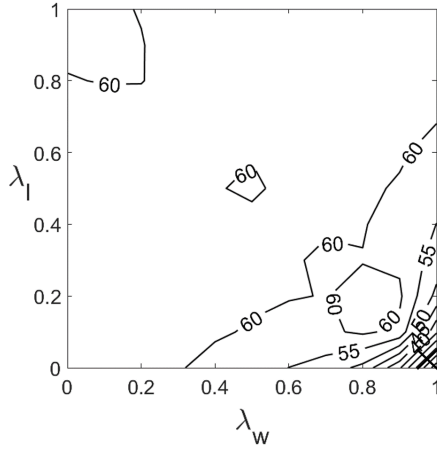


(b) *AIT*

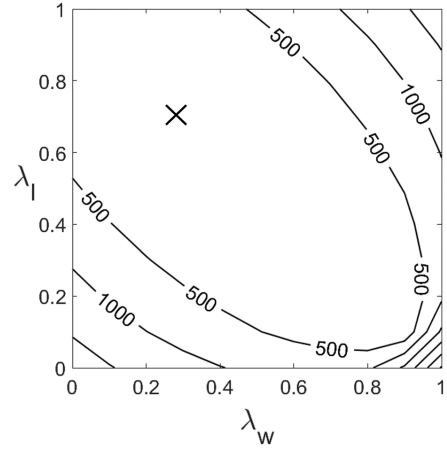


(c) *ALT*

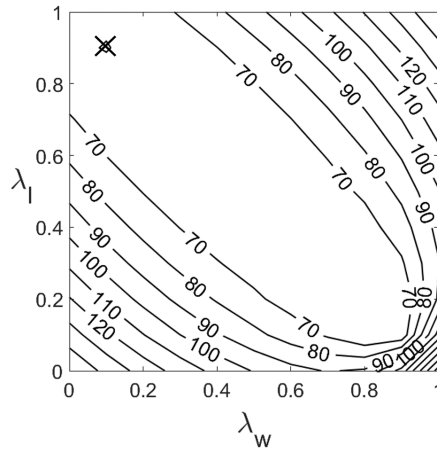
Figure B.11: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 11.



(a) *AWT*



(b) *AIT*



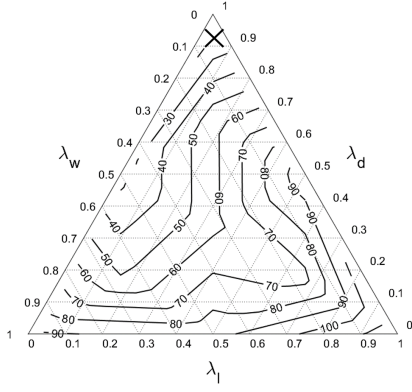
(c) *ALT*

Figure B.12: Performances of RTRD against SVR depending on  $\lambda_w$  and  $\lambda_l$  in dataset 12.

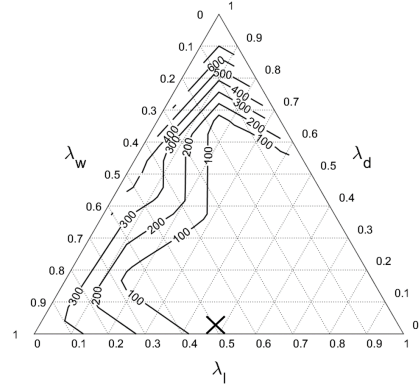


## Appendix C

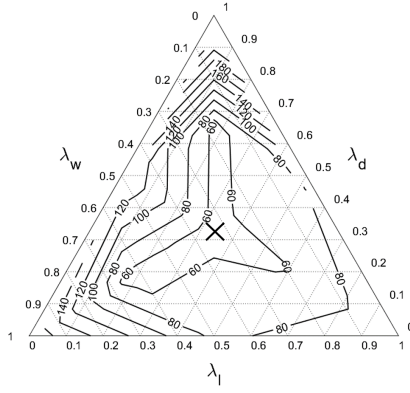
Performance contour of RTLD with respect to  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$



(a) *AWT*

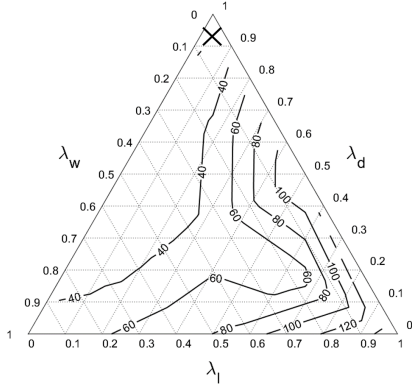


(b) *AIT*

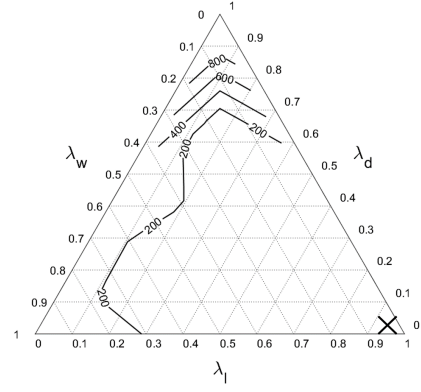


(c) *ALT*

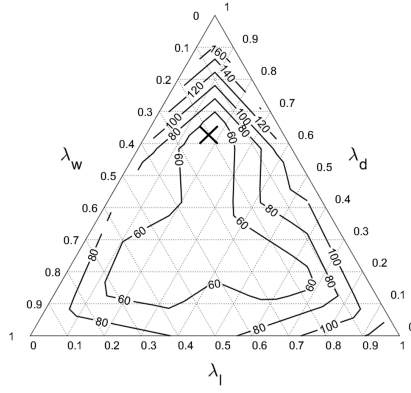
Figure C.1: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 1.



(a) *AWT*

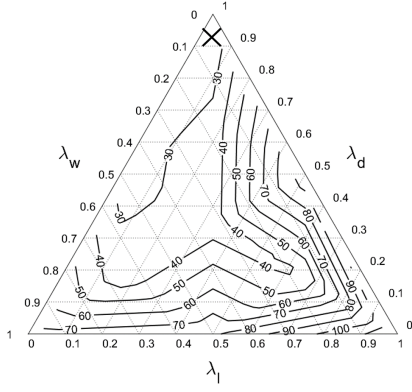


(b) *AIT*

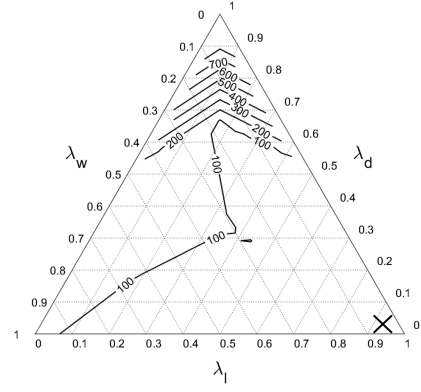


(c) *ALT*

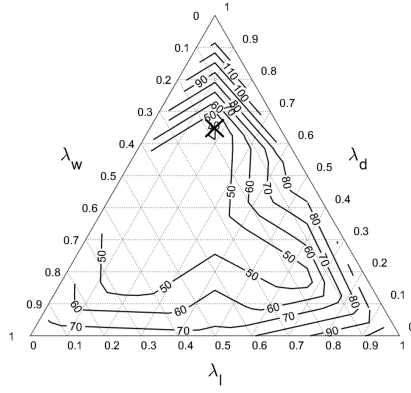
Figure C.2: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 2.



(a) *AWT*

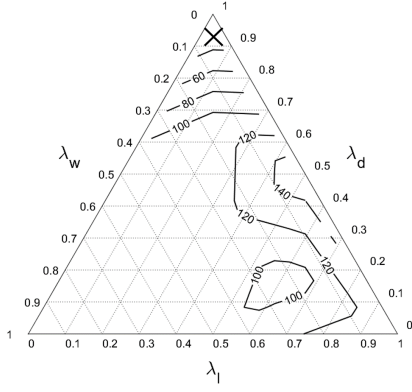


(b) *AIT*

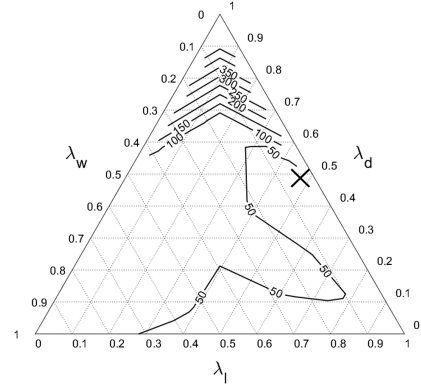


(c) *ALT*

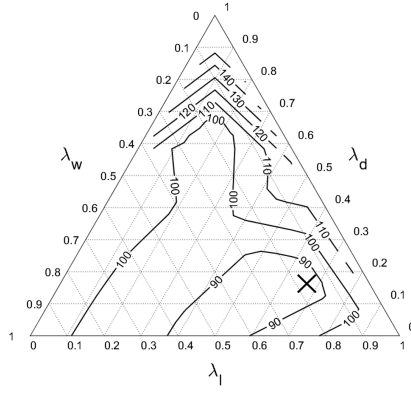
Figure C.3: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 3.



(a) *AWT*

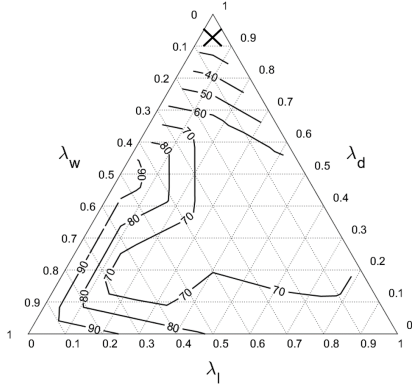


(b) *AIT*

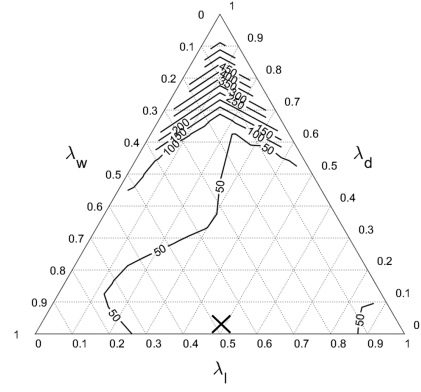


(c) *ALT*

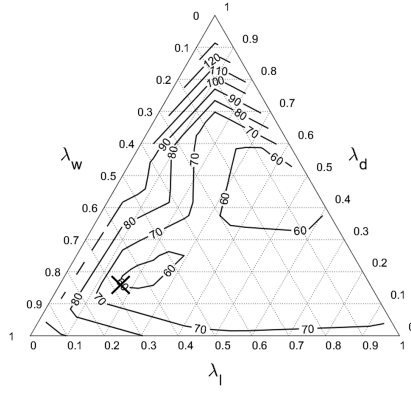
Figure C.4: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 4.



(a) *AWT*

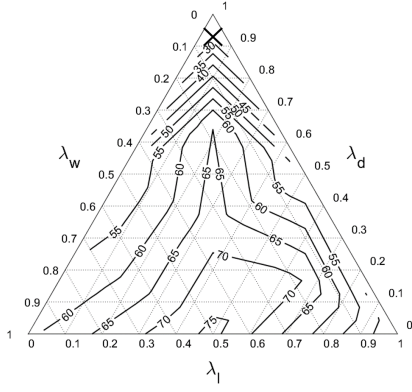


(b) *AIT*

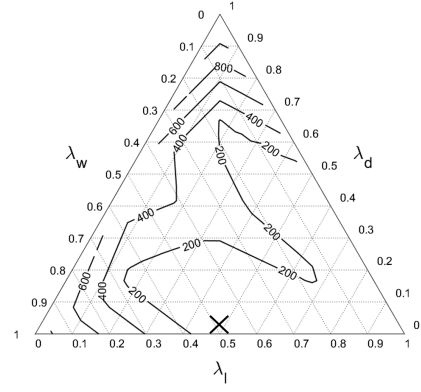


(c) *ALT*

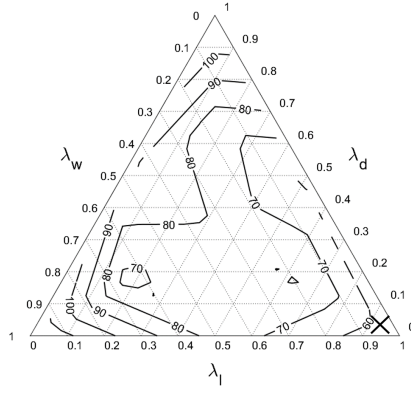
Figure C.5: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 5.



(a) *AWT*

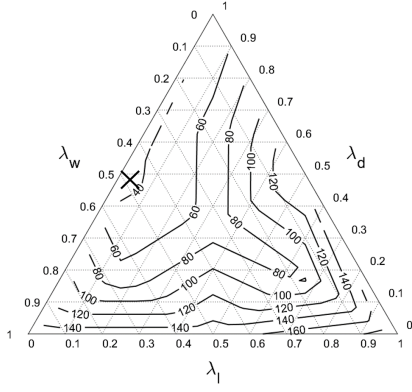


(b) *AIT*

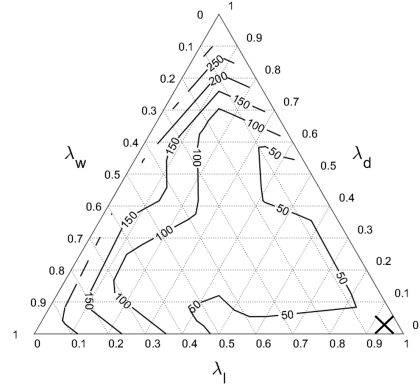


(c) *ALT*

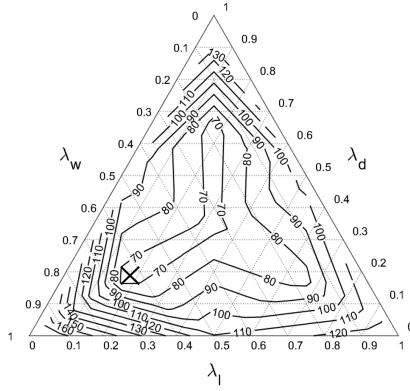
Figure C.6: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 6.



(a) *AWT*



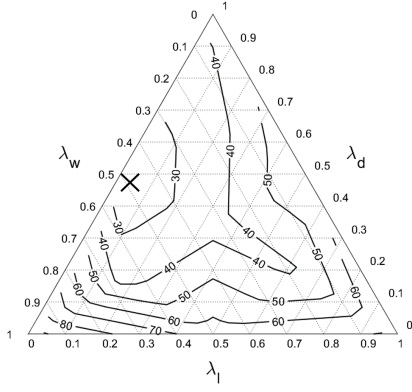
(b) *AIT*



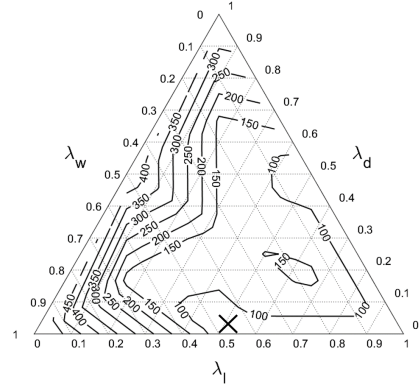
(c) *ALT*

Figure C.7: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 7.

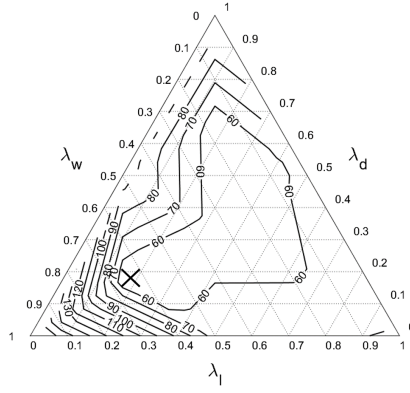




(a) *AWT*

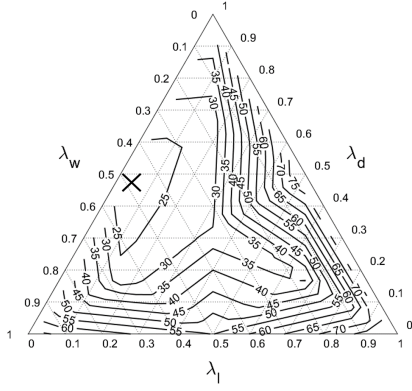


(b) *AIT*

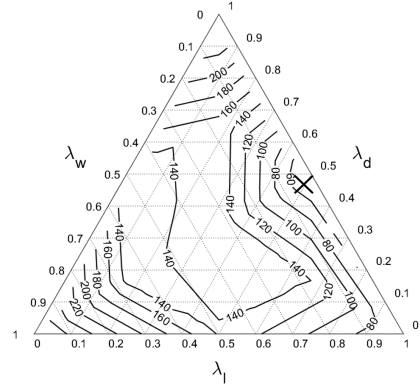


(c) *ALT*

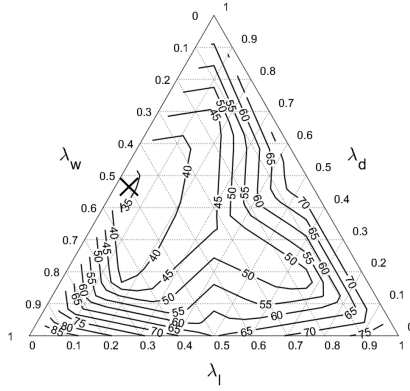
Figure C.8: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 8.



(a) *AWT*

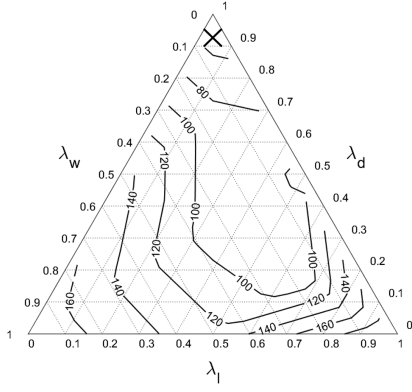


(b) *AIT*

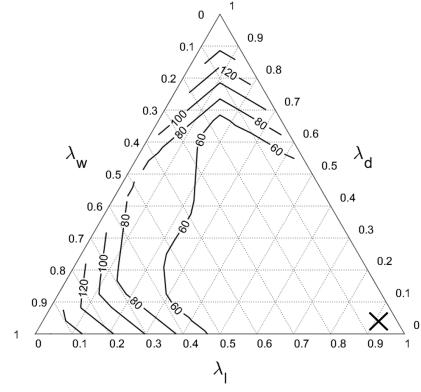


(c) *ALT*

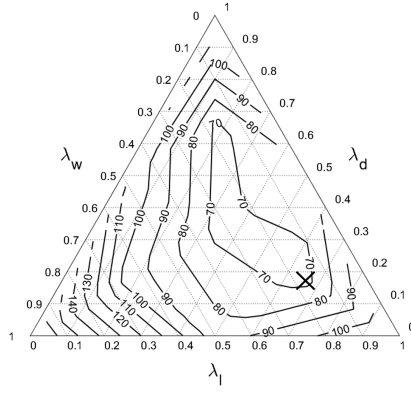
Figure C.9: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 9.



(a) *AWT*

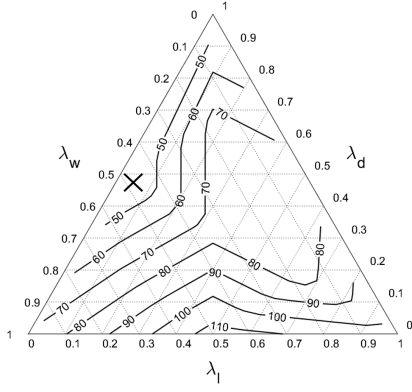


(b) *AIT*

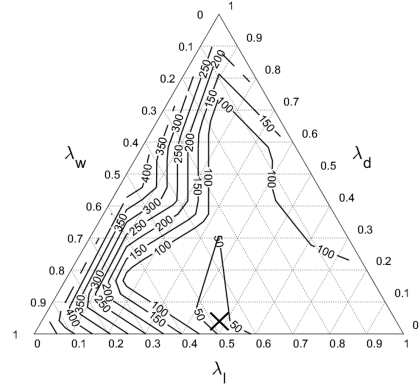


(c) *ALT*

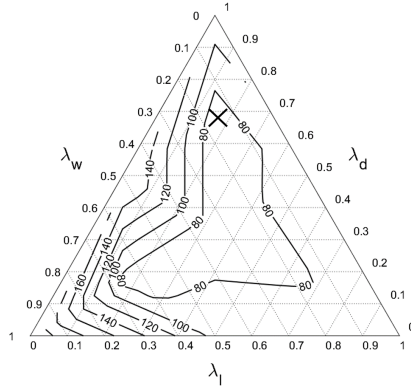
Figure C.10: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 10.



(a) *AWT*

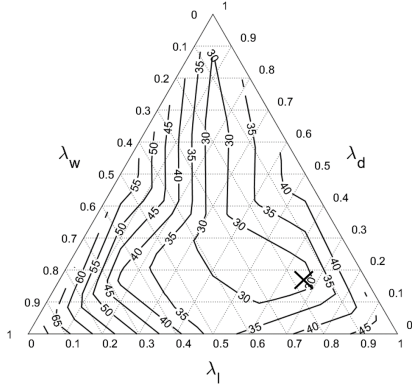


(b) *AIT*

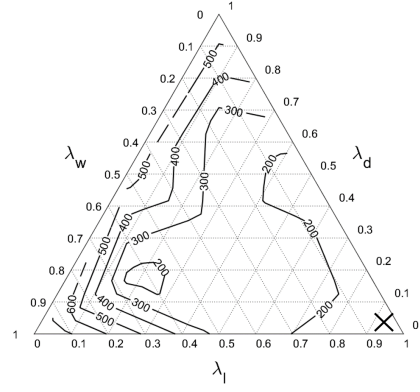


(c) *ALT*

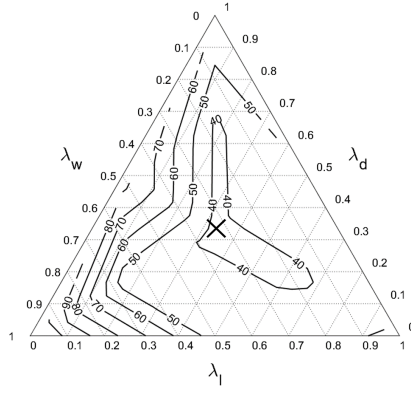
Figure C.11: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 11.



(a) *AWT*



(b) *AIT*



(c) *ALT*

Figure C.12: Performances of RTLD against SVR depending on  $\lambda_w$ ,  $\lambda_l$ , and  $\lambda_d$  in dataset 12.

## Bibliography

- [1] I. Sindicic, S. Bogdan, and T. Petrovic, “Resource allocation in free-choice multiple reentrant manufacturing systems based on machine-job incidence matrix,” *IEEE Transactions on Industrial Informatics*, vol. 7, no. 1, pp. 105–114, Feb. 2011.
- [2] J. Zhang and X. Wang, “Multi-agent-based hierarchical collaborative scheduling in re-entrant manufacturing systems,” *International Journal of Production Research*, vol. 54, no. 23, pp. 7043–7059, Dec. 2016.
- [3] H.-S. Choi, J.-S. Kim, and D.-H. Lee, “Real-time scheduling for reentrant hybrid flow shops: A decision tree based mechanism and its application to a TFT-LCD line,” *Expert Systems with Applications*, vol. 38, no. 4, pp. 3514–3521, Apr. 2011.
- [4] J. A. Ramirez-Hernandez and E. Fernandez, “Control of a re-entrant line manufacturing model with a reinforcement learning approach,” in *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, Dec. 2007, pp. 330–335.
- [5] J. C. Chen, K. H. Chen, J. J. Wu, and C. W. Chen, “A study of the flexible job shop scheduling problem with parallel machines and reentrant process,” *The*

- International Journal of Advanced Manufacturing Technology*, vol. 39, no. 3, pp. 344–354, Oct. 2008.
- [6] P. R. Kumar, “Re-entrant lines,” *Queueing Systems*, vol. 13, no. 1, pp. 87–110, Mar. 1993.
- [7] Y. Narahari and L. M. Khan, “Performance analysis of scheduling policies in re-entrant manufacturing systems,” *Computers & Operations Research*, vol. 23, no. 1, pp. 37–51, 1996.
- [8] M. C. Gomes, A. P. Barbosa-Póvoa, and A. Q. Novais, “Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach,” *International Journal of Production Research*, vol. 51, no. 17, pp. 5120–5141, Sep. 2013.
- [9] Y. H. Han and J. Y. Choi, “A GSPN-Based Approach to Stacked Chips Scheduling Problem,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 1, pp. 4–12, Feb. 2010.
- [10] F. He, D. Armbruster, M. Herty, and M. Dong, “Feedback control for priority rules in re-entrant semiconductor manufacturing,” *Applied Mathematical Modelling*, vol. 39, no. 16, pp. 4655–4664, Aug. 2015.
- [11] J. C. Tyan, T. C. Du, J. C. Chen, and I. H. Chang, “Multiple response optimization in a fully automated FAB: an integrated tool and vehicle dispatching strategy,” *Computers & Industrial Engineering*, vol. 46, no. 1, pp. 121–139, Mar. 2004.

- [12] K. Sourirajan and R. Uzsoy, “Hybrid decomposition heuristics for solving large-scale scheduling problems in semiconductor wafer fabrication,” *Journal of Scheduling*, vol. 10, no. 1, pp. 41–65, Feb. 2007.
- [13] H. Liu, Z. Jiang, and R. Y. K. Fung, “The infrastructure of the timed EOPNs-based multiple-objective real-time scheduling system for 300 mm wafer fab,” *International Journal of Production Research*, vol. 45, no. 21, pp. 5017–5056, Nov. 2007.
- [14] H. Zhang, Z. Jiang, and C. Guo, “Simulation-based optimization of dispatching rules for semiconductor wafer fabrication system scheduling by the response surface methodology,” *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 1, pp. 110–121, Mar. 2009.
- [15] C.-H. Tsai, Y.-M. Feng, and R.-K. Li, “A hybrid dispatching rules in wafer fabrication factories,” *International journal of the computer, the internet and management*, vol. 11, no. 1, pp. 64–72, 2003.
- [16] B. Lee, Y. H. Lee, T. Yang, and J. Ignisio, “A due-date based production control policy using WIP balance for implementation in semiconductor fabrications,” *International Journal of Production Research*, vol. 46, no. 20, pp. 5515–5529, Oct. 2008.
- [17] J. Huh, I. Park, S. Lim, B. Paeng, J. Park, and K. Kim, “Learning to dispatch operations with intentional delay for re-entrant multiple-chip product assembly lines,” *Sustainability*, vol. 10, no. 11, pp. 4123–4143, 2018.



- [18] W. L. Pearn, Y. T. Tai, and J. H. Lee, "Statistical approach for cycle time estimation in semiconductor packaging factories," *IEEE Transactions on Electronics Packaging Manufacturing*, vol. 32, no. 3, pp. 198–205, Jul. 2009.
- [19] M. Liu and C. Wu, "Genetic algorithm using sequence rule chain for multi-objective optimization in re-entrant micro-electronic production line," *Robotics and Computer-Integrated Manufacturing*, vol. 20, no. 3, pp. 225–236, Jun. 2004.
- [20] H. Jang, T. Y. Jung, K. Yeh, and J. H. Lee, "A model predictive control approach for fab-wide scheduling in semiconductor manufacturing facilities," *IFAC Proceedings Volumes*, vol. 46, no. 24, pp. 493–498, Sep. 2013.
- [21] F. D. Vargas-Villamil and D. E. Rivera, "A model predictive control approach for real-time optimization of reentrant manufacturing lines," *Computers in Industry*, vol. 45, no. 1, pp. 45–57, May 2001.
- [22] L. Danping and C. K. M. Lee, "A review of the research methodology for the re-entrant scheduling problem," *International Journal of Production Research*, vol. 49, no. 8, pp. 2221–2242, Apr. 2011.
- [23] J. A. Ramirez-Hernandez and E. Fernandez, "Optimal job releasing and sequencing for a reentrant manufacturing line with finite capacity buffers," in *Proceedings of the 45th IEEE Conference on Decision and Control*, Feb. 2006, pp. 6654–6659.
- [24] F. Dugardin, L. Amodeo, and F. Yalaoui, "Multiobjective scheduling of a reentrant hybrid flowshop," in *2009 International Conference on Computers Industrial Engineering*, Jul. 2009, pp. 193–195.

- [25] F. D. Vargas-Villamil, D. E. Rivera, and K. G. Kempf, “A hierarchical approach to production control of reentrant semiconductor manufacturing lines,” *IEEE Transactions on Control Systems Technology*, vol. 11, no. 4, pp. 578–587, Jul. 2003.
- [26] P. Y. Mok, “A decision support system for the production control of a semiconductor packaging assembly line,” *Expert Systems with Applications*, vol. 36, no. 3, pp. 4423–4430, Apr. 2009.
- [27] Y. H. Lee and T. Kim, “Manufacturing cycle time reduction using balance control in the semiconductor fabrication line,” *Production Planning & Control*, vol. 13, no. 6, pp. 529–540, Aug. 2002.
- [28] F. D. Vargas-Villamil and D. E. Rivera, “Multilayer optimization and scheduling using model predictive control: application to reentrant semiconductor manufacturing lines,” *Computers & Chemical Engineering*, vol. 24, no. 8, pp. 2009–2021, Sep. 2000.
- [29] J. T. Lin and C.-M. Chen, “Simulation optimization with GA and OCBA for semiconductor back-end assembly scheduling,” in *2015 International Conference on Industrial Engineering and Operations Management (IEOM)*, Mar. 2015, pp. 1–8.
- [30] Y.-H. Kang, S.-S. Kim, and H. J. Shin, “A scheduling algorithm for the reentrant shop: an application in semiconductor manufacture,” *The International Journal of Advanced Manufacturing Technology*, vol. 35, no. 5, pp. 566–574, Dec. 2007.

- [31] B. S. Chung, J. Lim, I. B. Park, J. Park, M. Seo, and J. Seo, "Setup change scheduling for semiconductor packaging facilities using a genetic algorithm with an operator recommender," *IEEE Transactions on Semiconductor Manufacturing*, vol. 27, no. 3, pp. 377–387, Aug. 2014.
- [32] C. Pickardt, J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness," in *Proceedings of the 2010 Winter Simulation Conference*, Dec. 2010, pp. 2504–2515.
- [33] L. Li, Z. Sun, M. Zhou, and F. Qiao, "Adaptive dispatching rule for semiconductor wafer fabrication facility," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 2, pp. 354–364, Apr. 2013.
- [34] J. Lim, M. J. Chae, Y. Yang, I. B. Park, J. Lee, and J. Park, "Fast scheduling of semiconductor manufacturing facilities using case-based reasoning," *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, pp. 22–32, Feb. 2016.
- [35] K. Appleton-Day and L. Shao, "Real-time dispatch gets real-time results in AMD's fab 25," in *1997 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop ASMC 97 Proceedings*, Sep. 1997, pp. 444–447.
- [36] J. J. Kanet, "Tactically delayed versus non-delay scheduling: An experimental investigation," *European Journal of Operational Research*, vol. 24, no. 1, pp. 99–105, Jan. 1986.

- [37] J. J. Kanet and V. Sridharan, “Scheduling with inserted idle time: Problem taxonomy and literature review,” *Operations Research*, vol. 48, no. 1, pp. 99–110, 2000.
- [38] O. Gholami and Y. N. Sotskov, “A neural network algorithm for servicing jobs with sequential and parallel machines,” *Automation and Remote Control*, vol. 75, no. 7, pp. 1203–1220, Jul. 2014.
- [39] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.
- [40] P. Thomas and A. Thomas, “Multilayer perceptron for simulation models reduction: Application to a sawmill workshop,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 4, pp. 646–657, Jun. 2011.
- [41] W. Mouelhi-Chibani and H. Pierreval, “Training a neural network to select dispatching rules in real time,” *Computers & Industrial Engineering*, vol. 58, no. 2, pp. 249–256, Mar. 2010.
- [42] P. Priore, D. de la Fuente, J. Puente, and J. Parreño, “A comparison of machine-learning algorithms for dynamic scheduling of flexible manufacturing systems,” *Engineering Applications of Artificial Intelligence*, vol. 19, no. 3, pp. 247–255, Apr. 2006.
- [43] A. K. Gupta and A. I. Sivakumar, “Job shop scheduling techniques in semiconductor manufacturing,” *The International Journal of Advanced Manufacturing Technology*, vol. 27, no. 11, pp. 1163–1169, Feb. 2006.

- [44] G. R. Weckman, C. V. Ganduri, and D. A. Koonce, “A neural network job-shop scheduler,” *Journal of Intelligent Manufacturing*, vol. 19, no. 2, pp. 191–201, Apr. 2008.
- [45] F. Zhou, C. Wu, and C. Yu, “Dynamic dispatching for re-entrant production lines — A deep learning approach,” in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Aug. 2017, pp. 1026–1031.
- [46] Z. Hammami, W. Mouelhi, and L. Ben Said, “On-line self-adaptive framework for tailoring a neural-agent learning model addressing dynamic real-time scheduling problems,” *Journal of Manufacturing Systems*, vol. 45, pp. 97–108, Oct. 2017.
- [47] D. Golmohammadi, “A neural network decision-making model for job-shop scheduling,” *International Journal of Production Research*, vol. 51, no. 17, pp. 5142–5157, Sep. 2013.
- [48] D.-Y. Liao and C.-N. Wang, “A neural-network approach to delivery time estimation for 300mm automatic material handling operations,” in *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*, vol. 2, Nov. 2003, pp. 1073–1079 Vol.2.
- [49] J. Domaschke, S. Brown, J. Robinson, and F. Leibl, “Effective implementation of cycle time reduction strategies for semiconductor back-end manufacturing,” in *Proceedings of the 30th Conference on Winter Simulation*, ser. WSC ’98. IEEE Computer Society Press, 1998, pp. 985–992.

- [50] A. I. Sivakumar and C. S. Chong, “A simulation based analysis of cycle time distribution, and throughput in semiconductor backend manufacturing,” *Computers in Industry*, vol. 45, no. 1, pp. 59–78, May 2001.
- [51] S. Werner, S. Horn, G. Weigert, and T. Jähnig, “Simulation based scheduling system in a semiconductor backend facility,” in *Proceedings of the 38th Conference on Winter Simulation*, ser. WSC ’06. Winter Simulation Conference, 2006, pp. 1741–1748.
- [52] H. Zhang, Z. Jiang, and C. Guo, “Simulation based real-time scheduling method for dispatching and rework control of semiconductor manufacturing system,” in *2007 IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2007, pp. 2901–2905.
- [53] Y.-F. Hung and I.-R. Chen, “A simulation study of dispatch rules for reducing flow times in semiconductor wafer fabrication,” *Production Planning & Control*, vol. 9, no. 7, pp. 714–722, Jan. 1998.
- [54] J. Potoradi, O. S. Boon, and S. J. Mason, “Using simulation-based scheduling to maximize demand fulfillment in a semiconductor assembly facility,” in *Proceedings of the Winter Simulation Conference*, vol. 2, Dec. 2002, pp. 1857–1861.
- [55] M. Y. H. Low, K. W. Lye, P. Lendermann, S. J. Turner, R. T. W. Chim, and S. H. Leo, “An agent-based approach for managing symbiotic simulation of semiconductor assembly and test operation,” in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS ’05. ACM, 2005, pp. 85–92.

- [56] T. J. Chua, T. X. Cai, and X. F. Yin, “A heuristic approach for scheduling multi-chip packages for semiconductor backend assembly,” in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, Sep. 2007, pp. 1024–1030.
- [57] Y.-F. Hung and C.-B. Chang, “Dispatching rules using flow time predictions for semiconductor wafer fabrications,” *Journal of the Chinese Institute of Industrial Engineers*, vol. 19, no. 1, pp. 67–75, Jan. 2002.
- [58] Y. Li, J. Li, J. Yao, and Y. Ni, “Development of an integrated real time dispatching system: A case study at a semiconductor assembly and test factory,” *Journal of Manufacturing Technology Management*, vol. 25, no. 7, pp. 980–997, Aug. 2014.
- [59] H.-S. Min and Y. Yih, “Selection of dispatching rules on multiple dispatching decision points in real-time scheduling of a semiconductor wafer fabrication system,” *International Journal of Production Research*, vol. 41, no. 16, pp. 3921–3941, Jan. 2003.
- [60] Y. F. Lee, Z. B. Jiang, and H. R. Liu, “Multiple-objective scheduling and real-time dispatching for the semiconductor manufacturing system,” *Computers & Operations Research*, vol. 36, no. 3, pp. 866–884, Mar. 2009.
- [61] S. J. Lee and T. E. Lee, “Scheduling a multi-chip package assembly line with reentrant processes and unrelated parallel machines,” in *2008 Winter Simulation Conference*, Dec. 2008, pp. 2286–2291.

- [62] D. M. Chiang, R.-S. Guo, and F.-Y. Pai, “Improved customer satisfaction with a hybrid dispatching rule in semiconductor back-end factories,” *International Journal of Production Research*, vol. 46, no. 17, pp. 4903–4923, Sep. 2008.
- [63] J. F. Bard, Z. Gao, R. Chacon, and J. Stuber, “Daily scheduling of multi-pass lots at assembly and test facilities,” *International Journal of Production Research*, vol. 51, no. 23, pp. 7047–7070, Nov. 2013.
- [64] F. Tovia, S. J. Mason, and B. Ramasami, “A scheduling heuristic for maximizing wirebonder throughput,” *IEEE Transactions on Electronics Packaging Manufacturing*, vol. 27, no. 2, pp. 145–150, Apr. 2004.
- [65] Y. Ma, F. Qiao, F. Zhao, and J. W. Sutherland, “Dynamic scheduling of a semiconductor production line based on a composite rule set,” *Applied Sciences*, vol. 7, no. 10, p. 1052, 2017.
- [66] J. T. Lin and C.-M. Chen, “Simulation optimization approach for hybrid flow shop scheduling problem in semiconductor back-end manufacturing,” *Simulation Modelling Practice and Theory*, vol. 51, pp. 100–114, Feb. 2015.
- [67] M. K. E. Adl, A. A. Rodriguez, and K. S. Tsakalis, “Hierarchical modeling and control of re-entrant semiconductor manufacturing facilities,” in *Proceedings of 35th IEEE Conference on Decision and Control*, vol. 2, Dec. 1996, pp. 1736–1742 vol.2.
- [68] Y. Deng, J. F. Bard, G. R. Chacon, and J. Stuber, “Scheduling back-end operations in semiconductor manufacturing,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 23, no. 2, pp. 210–220, May 2010.



- [69] M. Fu, R. Askin, J. Fowler, M. Haghnevis, N. Keng, J. S. Pettinato, and M. Zhang, “Batch production scheduling for semiconductor back-end operations,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 24, no. 2, pp. 249–260, May 2011.
- [70] J. A. Ramirez-Hernandez and E. Fernandez, “A case study in scheduling reentrant manufacturing lines: Optimal and simulation-based approaches,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec. 2005, pp. 2158–2163.
- [71] C.-C. Liu, H.-Y. Jin, Y. Tian, and H.-B. Yu, “Reinforcement learning approach to re-entrant manufacturing system scheduling,” in *2001 International Conferences on Info-Tech and Info-Net. Proceedings (Cat. No.01EX479)*, vol. 3, 2001, pp. 280–285 vol.3.
- [72] J. A. Ramirez-Hernandez and E. Fernandez, “An approximate dynamic programming approach for job releasing and sequencing in a reentrant manufacturing line,” in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Apr. 2007, pp. 201–208.
- [73] C.-H. Chen, J. Lin, E. Yücesan, and S. E. Chick, “Simulation budget allocation for further enhancing the efficiency of ordinal optimization,” *Discrete Event Dynamic Systems*, vol. 10, no. 3, pp. 251–270, 2000.
- [74] C. H. Dagli, *Artificial Neural Networks for Intelligent Manufacturing*. Springer Science & Business Media, Dec. 2012, google-Books-ID: K4ftCAAQBAJ.

- [75] P. Priore, A. Gómez, R. Pino, and R. Rosillo, “Dynamic scheduling of manufacturing systems using machine learning: An updated review,” *AI EDAM*, vol. 28, no. 01, pp. 83–97, 2014.
- [76] I. Moon, S. Lee, and H. Bae, “Genetic algorithms for job shop scheduling problems with alternative routings,” *International Journal of Production Research*, vol. 46, no. 10, pp. 2695–2705, May 2008.
- [77] A. El-Bouri and P. Shah, “A neural network for dispatching rule selection in a job shop,” *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3-4, pp. 342–349, Nov. 2006.
- [78] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, “Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations,” *Evolutionary Computation*, vol. 23, no. 2, pp. 249–277, Jun. 2014.
- [79] L. Tang, W. Liu, and J. Liu, “A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment,” *Journal of Intelligent Manufacturing*, vol. 16, no. 3, pp. 361–370, Jun. 2005.
- [80] R.-S. Guh, Y.-R. Shiue, and T.-Y. Tseng, “The study of real time scheduling by an intelligent multi-controller approach,” *International Journal of Production Research*, vol. 49, no. 10, pp. 2977–2997, May 2011.
- [81] Y.-R. Shiue and R.-S. Guh, “Learning-based multi-pass adaptive scheduling for a dynamic manufacturing cell environment,” *Robotics and Computer-Integrated Manufacturing*, vol. 22, no. 3, pp. 203–216, Jul. 2006.

- [82] K.-J. Wang, J. C. Chen, and Y.-S. Lin, “A hybrid knowledge discovery model using decision tree and neural network for selecting dispatching rules of a semiconductor final testing factory,” *Production Planning & Control*, vol. 16, no. 7, pp. 665–680, Oct. 2005.
- [83] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, “Automated Design of Production Scheduling Heuristics: A Review,” *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, Feb. 2016.
- [84] R. Linn and W. Zhang, “Hybrid flow shop scheduling: A survey,” *Computers & Industrial Engineering*, vol. 37, no. 1, pp. 57–61, Oct. 1999.
- [85] M. MONTAZERI and L. N. V. WASSENHOVE, “Analysis of scheduling rules for an FMS,” *International Journal of Production Research*, vol. 28, no. 4, pp. 785–802, Apr. 1990.
- [86] D. Quadt, “Simulation-based scheduling of parallel wire-bonders with limited clamp&paddles,” in *Proceedings of the 38th Conference on Winter Simulation*, ser. WSC '06. Winter Simulation Conference, 2006, pp. 1887–1892.
- [87] A. I. Sivakumar, “Optimization of a cycle time and utilization in semiconductor test manufacturing using simulation based, on-line, near-real-time scheduling system,” in *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1*, ser. WSC '99. ACM, 1999, pp. 727–735.

- [88] A. Negahban and J. S. Smith, “Simulation for manufacturing system design and operation: Literature review and analysis,” *Journal of Manufacturing Systems*, vol. 33, no. 2, pp. 241–261, Apr. 2014.
- [89] A. Varga, “Discrete event simulation system,” in *Proc. of the European Simulation Multiconference (ESM’2001)*, 2001.
- [90] E. Red, G. Jensen, D. French, and P. Weerakoon, “Multi-user architectures for computer-aided engineering collaboration,” in *2011 17th International Conference on Concurrent Enterprising*, Jun. 2011, pp. 1–10.
- [91] W. Han and M. Jochum, “Near real-time satellite data quality monitoring and control,” in *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, Jul. 2016, pp. 206–209.
- [92] A. Pritsker and K. Snyder, “Production scheduling using factor,” in *The Planning and Scheduling of Production Systems*. Springer, 1997, pp. 337–358.
- [93] S. H. Lee, F. Pena-Mora, and M. Park, “Dynamic planning and control methodology for strategic and operational construction project management,” *Automation in construction*, vol. 15, no. 1, pp. 84–97, 2006.
- [94] J. Jo, J. Huh, J. Park, B. Kim, and J. Seo, “LiveGantt: Interactively Visualizing a Large Manufacturing Schedule,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 12, pp. 2329–2338, Dec. 2014.
- [95] P. Xu, H. Mei, L. Ren, and W. Chen, “Vidx: Visual diagnostics of assembly line performance in smart factories,” *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 291–300, 2017.

- [96] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1, June 2013, p. 3.
- [97] R. Haupt, “A survey of priority rule-based scheduling,” *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3–16, Mar. 1989.
- [98] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.
- [99] J. Weston, O. Chapelle, V. Vapnik, A. Elisseeff, and B. Schölkopf, “Kernel dependency estimation,” in *Advances in neural information processing systems*, 2003, pp. 897–904.
- [100] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, p. 533, 1986.
- [101] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, “Hyper-heuristic evolution of dispatching rules: A comparison of rule representations,” *Evolutionary computation*, vol. 23, no. 2, pp. 249–277, 2015.
- [102] S. C. Sarin, A. Varadarajan, and L. Wang, “A survey of dispatching rules for operational control in wafer fabrication,” *Production Planning & Control*, vol. 22, no. 1, pp. 4–24, Jan. 2011.

## 국문초록

재유입 제조라인에서는 하나의 제품을 완성하기까지 부품이 다수의 공정을 여러 번 반복하는 과정이 반드시 발생한다. 반도체와 액정 디스플레이 제조공정의 등장으로 재유입 제조라인은 학계와 산업계의 주목을 모두 받게 되었다. 최근에 플래시 메모리와 액정 디스플레이가 내장된 소형 전자 기기의 수요가 증가함에 따라 관련 연구가 활발히 진행되고 있다.

본 논문에서는 재유입 제조라인의 병목 단계의 설비 가동률을 저하시키지 않으면서 플로우 타임을 감소시키기 위한 심층신경망 기반의 실시간 디스패처를 제안하고자 한다. 재유입 제조라인의 여러 단계에 반복하여 진입하는 부품들은 플로우 타임을 줄이고 설비 가동률을 향상시키는 두 가지 목표를 동시에 달성하는 것을 어렵게 만드는 주요 인이다. 구체적으로, 산출물의 양을 최대화하기 위하여 충분한 양의 재공을 공급하면 설비의 가동률을 높은 수준으로 유지할 수 있다. 하지만 재공의 양이 지나치게 많으면 다음 공정에서 부품들의 대기 시간이 길어지므로 부품들의 플로우 타임이 증가하게 된다.

논문에서 제안하는 새로운 방법론들은 다음과 같다. 첫째, 학습 데이터를 생성하고 디스패칭 의사결정의 성능을 평가하기 위해 이산 사건 기반 시뮬레이터와 시각화 도구가 개발되었다. 이산 사건 기반 시뮬레이터는 실제 현장의 재유입 제조라인을 모방하고 심층 신경망을 학습시키기 위한 데이터를 생성하는 역할을 담당한다. 시각화 도구는 디스패칭 의사결정이 수행될 때의 제조라인의 상태를 표현하는 기능과 시간의 경과에 따른 다양한 성능 지표의 변화를 관찰할 수 있는 기능을 함께 제공한다.

둘째, 서로 다른 의사결정 과정을 수행하는 두 종류의 심층신경망 기반 실시간 디스패처가 제안되었다. 학습 단계에서, 디스패처는 시뮬레이터에 의해 생성된 제조라인

데이터를 이용하여 디스패칭 의사결정이 요구되는 시점의 각 대안의 선호도를 학습한다. 그 후 실시간 디스패칭 단계에서, 디스패처는 의도적인 지연을 고려하여 디스패칭 의사결정을 수행한다. 이 때, 부품의 대기 시간을 줄이면서 병목 단계의 유휴 시간을 감소시킬 것으로 기대되는 대안이 높은 점수를 받게 된다.

본 논문의 기여와 실용적인 의미는 다음의 세 가지로 요약될 수 있다. 첫째, 사용자에게 디스패칭 의사결정 분석 기능을 제공하는 시각화 도구를 개발했다. 둘째, 제안된 방법론은 실제 현장의 재유입 제조라인을 모방하는 이산 사건 기반 시뮬레이터를 이용하여 심층신경망의 학습에 사용되는 데이터를 생성할 수 있다. 마지막으로, 제안된 실시간 디스패처는 재유입 제조라인의 의도적인 지연을 고려하여 병목 단계의 설비 가동률을 높이면서 플로우 타임을 감소시킬 수 있음이 성공적으로 검증되었다.

**주요어:** 재유입 제조라인, 실시간 디스패처, 의도적인 지연, 심층신경망, 플로우 타임, 설비 가동률, 시뮬레이터, 시각화 도구

**학번:** 2013-23211