SCHEDULING AND CONTROL WITH MACHINE LEARNING IN MANUFACTURING SYSTEMS

by

Sungbum Jun

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

School of Industrial Engineering

West Lafayette, Indiana

August 2020

THE PURDUE UNIVERSITY GRADUATE SCHOOL

STATEMENT OF COMMITTEE APPROVAL

Dr. Seokcheon Lee, Chair

School of Industrial Engineering

Dr. Yuehwern Yih

School of Industrial Engineering

Dr. Hua Cai

School of Industrial Engineering /
Division of Environmental and Ecological Engineering

Dr. Hyonho Chun

Department of Mathematical Sciences,
Korea Advanced Institute of Science and Technology

Approved by

Dr. Abhijit Deshmukh

Head of the Graduate Program

ACKNOWLEDGMENTS

This dissertation could not be accomplished the help, support, and guidance of many people during my Ph.D. study. First and foremost, I would like to deliver my sincere appreciation to my academic advisor, Dr. Seokcheon Lee. Without his academical, financial and personal supports, this work could not be completed. I would also like to thank my committee members, Dr. Yuehwern Yih, Dr. Hua Cai, and Dr. Hyonho Chun for their guidance and supervision for this work. I would like to express my gratitude to Dr. Jinwoo Park and Dr. Tae-woo Chang who gave me the opportunity to do research.

I also appreciate my colleagues in the Distributed Control Lab (DC Lab), Rahul, Chul-Hun, Patchara, Ho-Young, Ashutosh, Mohamad, Chiyoung, Zekun, Cansu, Jonggwang, Francisco, Ibrahim, Juan, Yonggab, and many others who always helped me to resolve academic and personal issues. Also, my special thanks go to Shinyoung Jun for her support and encouragement.

Last but not least, I am extremely grateful to my family (Changwoo Jun, Youngsook Jeong, and Kyungmi Jun) for their support and encouragement to pursue my education in Purdue.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABM	Agent-Based Modeling
ACO	Ant Colony Optimization
AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
CP	Constraint Programming
DR	Dispatching Rule
DT	Decision Tree
EDD	Earliest Due Date
FJSP	Flexible Job Shop Scheduling Problem
GA	Genetic Algorithm
HGA	Hybrid Genetic Algorithm
MA	Memetic Algorithm
MILP	Mixed-Integer Linear Programming
ML	Machine Learning
MSO	Model for Sequencing Operations
MTM	Model for Tie-breaking between alternative Machines
NP-hard	Non-deterministic Polynomial-time Hardness
NS	Neighborhood Search
PDP	Pickup and Delivery Problem
PSO	Particle Swarm Optimization
RANFORS	Random Forest for Obtaining Rules for Scheduling
SA	Simulated Annealing
SMSP	Single-Machine Scheduling Problem
SPT	Shortest Processing Time
VRP	Vehicle Routing Problem

ABSTRACT

Numerous optimization problems in production systems can be considered as decision-making processes that determine the best allocation of resources to tasks over time to optimize one or more objectives in concert with big data. Among the optimization problems, production scheduling and routing of robots for material handling are becoming more important due to their impacts on system performance. However, the development of efficient algorithms for scheduling or routing faces several challenges. While the scheduling and vehicle routing problems can be solved by mathematical models such as mixed-integer linear programming to find optimal solutions to small-sized problems, they are not applicable to larger problems due to the nature of NP-hard problems. Thus, further research on machine learning applications to those problems is a significant step towards increasing the possibilities and potentialities of field application. In order to create truly intelligent systems, new frameworks for scheduling and routing are proposed to utilize machine learning (ML) techniques. First, the dynamic single-machine scheduling problem for minimization of total weighted tardiness is addressed. In order to solve the problem more efficiently, a decision-tree-based approach called Generation of Rules Automatically with Feature construction and Tree-based learning (GRAFT) is designed to extract dispatching rules from existing or good schedules. In addition to the single-machine scheduling problem, the flexible job-shop scheduling problem with release times for minimizing the total weighted tardiness is analyzed. As a ML-based solution approach, a random-forest-based approach called Random Forest for Obtaining Rules for Scheduling (RANFORS) is developed to solve the problem by generating dispatching rules automatically. Finally, an optimization problem for routing of autonomous robots for minimizing total tardiness of transportation requests is analyzed by decomposing it into three sub-problems. In order to solve the sub-problems, a comprehensive framework with consideration of conflicts between routes is proposed. Especially to the sub-problem for vehicle routing, a new local search algorithm called COntextual-Bandit-based Adaptive Local search with Tree-based regression (COBALT) that incorporates the contextual bandit into operator selection is developed. The findings from my research contribute to suggesting a guidance to practitioners for the applications of ML to scheduling and control problems, and ultimately to lead the implementation of smart factories.

CHAPTER 1. INTRODUCTION

The application of the word ‘smart’ has been extended from electronic devices to various other systems especially due to the rapid development of Internet of Things (IoT). The basic concept of IoT is that data from machines, sensors, people, and more are stored through networks and integrated into smart systems to make better decisions. In recent years, the concept of smart factories and Industry 4.0 with IoT and wireless sensor network have been widely spread to optimize various operations in a factory.

Especially, beyond traditional automation, smart factories connect all machines to be more intelligent and flexible systems by learning and adapting to changes from the shop floor. In addition, autonomous robots for material handling in factories are growing quickly and even small and medium-sized factories also can take more advantages of robots due to their flexibility. The convergence of wireless sensor network and big data with machine learning is accelerating the adoption of these technologies and providing numerous business opportunities in manufacturing environments.

On the other hand, manufacturing systems also face a variety of new optimization problems, which can be expressed as decision-making processes that determine the best allocation of resources to tasks over time in order to optimize one or more objectives. These problems cause new challenges compared to traditional optimization problems.

The first challenge is the tremendous size of data. Sensors, machines and a wide range of devices generate massive amounts of structured and unstructured data, which require a new approach to capture underlying knowledge. However, manufacturing environments are often too complex to consider all important attributes because the requisite information comes from multiple sources and sensors and much of the underlying logics of the operation might be implicit and challenging to capture intuitively.

Another challenge is a set of new constraints for manufacturing systems. It is crucial to identify various constraints such as the limited battery life and consider them in a solution approach. For example, in scheduling problems, a traditional job shop scheduling problem rarely exists. In practice, there might be a variety of machines with different abilities placed in parallel at stages to increase capacity and balance the workload.

The last challenge is the computational complexity. Although exact approaches such as a mixed-integer linear programming and dynamic programming can find an optimal solution, they are often impractical because of the extremely long calculation time for large problems. In case of heuristics, nature-inspired algorithms such as genetic algorithm can be applied to more complex problems, but the execution time and solution quality vary with the design of the algorithm. The following subchapters describe the detailed characteristics for two major problems in manufacturing systems: production scheduling and vehicle routing problems.

1.1 Production Scheduling

According to the characteristics of manufacturing systems, production scheduling problems can be formulated from the simple problem (single machine) to the highly complicated one (parallel machines for each process). Single-machine scheduling problems (SMSPs) have been extensively studied due to their applicability in various manufacturing environments and impact on factory-floor performance. Moreover, in line with the trends toward mass-customization and customer-focused manufacturing, consideration of due dates, job priorities, and dynamic arrivals is necessary in order to meet the demands of customers in a timely manner. An example of a single-machine scheduling problem with due dates, arrival times, and weights is the testing process of wafers in semiconductors (Chou et al. 2005).

Furthermore, in order to increase productivity and flexibility, many factories have one or more alternative machines or workers for each process. The traditional job shop scheduling problem (JSP) is not suitable for such production environments because some processes need to employ parallel machines. This type of problem can be defined as a flexible job shop scheduling problem (FJSP) as shown in Figure 1.1 (Pinedo 2012). In the figure, O_i represents the ordered set of operations of job i and M_j means the set of alternative machines on which operation j can be processed. Also, t_{ijk} presents the processing time of operation j of job i on machine k.

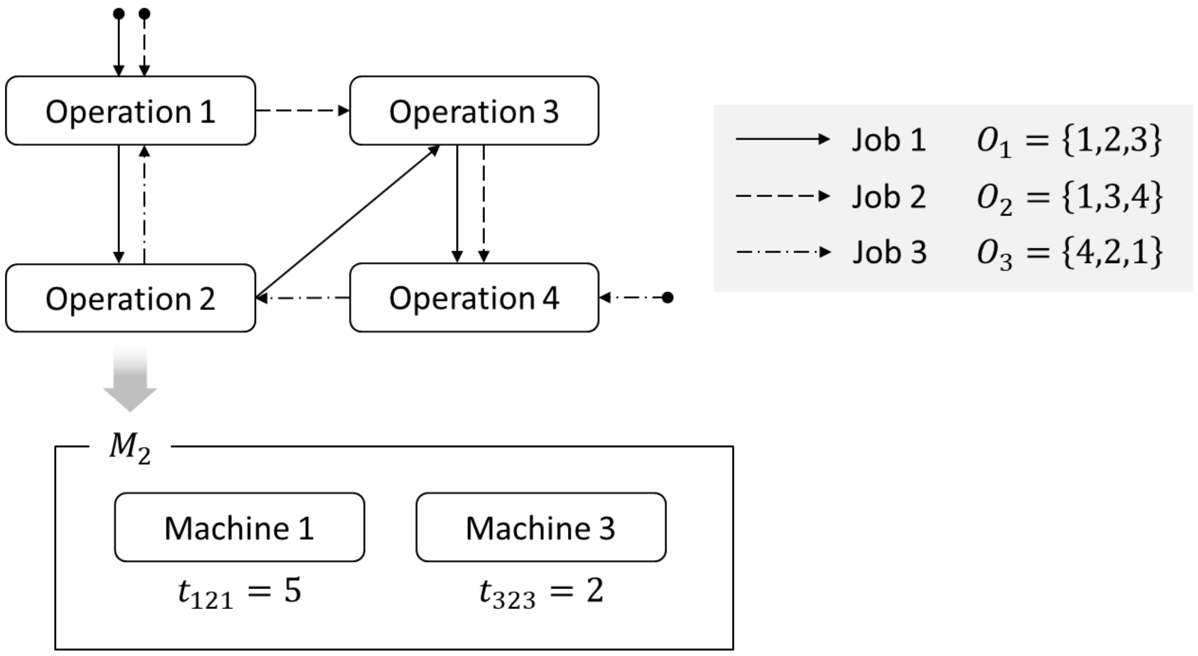


Figure 1.1 Summary of Flexible Job Shop Scheduling

1.2 Logistics with Autonomous Robots

Table 1.1 Comparison between AGVs and AMRs

	AGV	AMR
Paths	Fixed	Flexible
Frequency of finding paths	Sporadic (when installing lanes)	Frequent (when changes occur)
Conflict resolution by detouring	Not available	Available
Conflict resolution system	Centralized traffic control system	Decentralized collision detection and resolution by sensors
Additional features	Inductive power transfer, automatic battery replacement system	Data collection, shelf or cart attachment, robot arms for self-fulfillment

Traditionally, automated guided vehicles (AGVs) have been widely used in various fields related to material handling such as manufacturing and warehouse management. A major disadvantage of AGVs is that the route between two locations is fixed and there is less flexibility for changing of routes due to the fact that AGVs are guided by tape, wire, and magnetic tracks. Additionally, due to their high investment costs, AGVs have been used mainly in high-volume manufacturing operations. However, unlike AGVs, automated mobile robots (AMRs) can move freely without any sort of guidance such as tape or wire by using embedded sensors (e.g. light detection and ranging, cameras). While AGVs currently have a huge role in production logistics, their utilization in material handling is growing quickly as more and more manufacturers consider semi-autonomous or fully autonomous vehicles offering scalability and versatility as well as lower costs. The major differences between AGVs and AMRs are summarized in Table 1.1.

As shown in the above table, AMRs have a wide array of advantages compared with traditional AGVs. First, due to their autonomous ability, AMRs can easily avoid collisions and resolve conflicts between vehicles in a decentralized way. Also, with attachment of additional modules such as robot arms, AMRs can be widely used to perform additional tasks during transportation. Based on these great potentials, even small- and medium-sized factories can take advantage of AMRs as compared with expensive AGVs with their high investment cost and low versatility (Material Handling Institute 2018).

The major problem for AMRs is to identify a set of optimal routes, with consideration of conflicts, whereby a fleet of vehicles can serve given transportation requests. This problem can be categorized into three sub-problems: path finding (PF), vehicle routing (VR), and conflict resolution (CR). The goal of the path finding problem is to find the shortest path between two locations. In the case of the vehicle routing problem, the objective is to determine the best allocation of pickup and delivery operations with consideration of the paths calculated in the path finding problem. Finally, in terms of conflict resolution, the main purpose is to minimize collisions or delays based on the vehicle routing and path finding solutions.

1.3 Research Problems and Contributions

Whereas the scheduling problems such as SMSP and FJSP can be solved by mathematical models such as mixed-integer linear programming (MILP) to find optimal solutions to small-sized problems, they are not applicable to practical (larger) problems, due to the nature of NP-hard problems for which they are scaled up, and the computational time incurred consequently. Furthermore, much of the underlying logistics of actual operations are implicit and challenging to capture intuitively, because smart systems are often too complex to understand all important attributes, due especially to the massive data from multiple sources and sensors that they have to handle.

In the case of logistics with AMRs in factories, many hurdles such as the computational complexity of three sub-problems must be cleared in order to make the best use of AMRs. All of these sub-problems are not independent, but rather are closely related to each other. For example, in order to find the best path of an AMR, the required distance between two locations as well as the routes of the other vehicles must be considered. Also, the expected delivery time largely depends on delays caused by conflicts between AMRs.

These interdependencies among the sub-problems render AMR optimization problems more complicated. First of all, finding the shortest paths and routing AMRs based on them are NP-hard. This implies that, in the worst case, most algorithms will likely have great difficulties finding the optimal solution within a reasonable time. Also, even if the optimal schedule of AMRs with shortest paths are known, there are possibilities for situations in which two or more AMRs encounter each other in the same grid, which cause unnecessary delays and affect schedule performance thereby.

Thus, there is a significant need for research that can help systems make better decisions in extracting underlying decisions while considering various environments. Due particularly to the recent development of machine learning (ML) techniques, huge volumes of data can be transmitted through wireless networks to make systems more intelligent, understand complex decisions, and optimize them.

In order to utilize a tremendous amount of data, this dissertation aims at: 1) discovering both explicit and implicit knowledge from existing schedules and 2) proposing a comprehensive framework for AMRs. The contributions of this dissertation are:

- Development of an understandable ML method to extract both explicit and implicit knowledge as a set of rules from good schedules
- Improvement of decision-tree-based models with discretization and feature extraction by using the evolutionary process

- Implementation of a comprehensive framework for controlling AMRs efficiently with consideration of conflicts between routes
- Integration of reinforcement learning and local search selection in vehicle routing
- Development of an agent-based model with distributed decision-making for conflict resolution

1.4 Organization of Dissertation

The remaining chapters of this dissertation are organized as following sequences. In Chapter 2, the previous literature relevant to production scheduling and logistics is reviewed. Chapter 3 presents for the SMSP for minimization of total weighted tardiness by learning of dispatching rules (DRs) from schedules. In addition to the SMSP, the more complicated problem, FJSP, and an automated approach for discovering DRs are presented in Chapter 4. In Chapter 5, a new framework for controlling AMRs in production logistics is proposed. Finally, Chapter 6 concludes this dissertation and summarizes the plan for further research directions.

CHAPTER 2. LITERATURE REVIEW

In this chapter, the previous literature is reviewed in three categories: single-machine scheduling problem (SMSP), flexible job shop scheduling problem (FJSP), and control of material handling vehicles. Each of the sub-chapters is related to the research problem to be addressed in this dissertation.

2.1 Single-Machine Scheduling Problem (SMSP)

Single-machine scheduling problems (SMSPs) can be categorized into two types, offline and online (Pinedo 2012). In offline SMSPs, all information about jobs to be processed such as the number of jobs, processing times and release dates are known at the beginning. In contrast, in online SMSPs, the problem data are not known a priori and the detailed attributes of a given job are identified only once the job is released.

In terms of the objective functions, in a more customer-oriented and highly competitive environment, tardiness-related objective functions are becoming more and more crucial with respect to overall customer satisfaction. Moreover, the weight of an individual job is often used to determine the priorities of customer orders in production scheduling. However, most single-machine scheduling problems with dynamic arrivals, weight, and tardiness-related objective functions have been shown to be NP-hard. Lawler (1977) showed that $1||\sum w_j T_j$ is strongly NP-hard, which implies that $1|r_j|\sum w_j T_j$ is also strongly NP-hard.

Traditionally, exact approaches such as branch-and-bound algorithms and dynamic programming have been used for SSMSPs to produce optimal solutions. Akturk and Ozdemir (2000) proposed a branch-and-bound algorithm with dominance properties for $1|r_j|\sum w_j T_j$. In the case of online SMSPs, Liu et al. (2009) showed that there is no online algorithm with a finite competitive ratio for an online version of $1|r_j|\sum w_j T_j$ with $\sum w_j T_j > 0$. The competitive ratio of an algorithm is defined as the ρ value where the objective function value of a schedule obtained from the algorithm is no more than ρ times and no smaller value of ρ can be established for this algorithm (Anderson and Potts 2004).

However, although exact approaches can guarantee optimality, they are inapplicable to large or complex problems, due to their extremely long computation time. To deal with these issues more practically, heuristic algorithms also have been used to find the optimal or near-optimal schedule in a reasonable time. The heuristic algorithms can be classified into two types: constructive and improvement.

DRs, notably, which are a type of constructive heuristics, have been actively researched for the past several decades owing to their computational efficiency, ease of implementation, and utility for producing good solutions for specific problems. DRs based on the priority function; the job with the highest priority value is selected from a list of jobs and assigned to a machine every time the machine completes a job. When calculating the priorities of jobs, multiple attributes can be combined as a “composite” attribute for improved performance (Jayamohan and Rajendran 2004).

In order to find the best priority function for DRs automatically, genetic programming (GP) has been widely used to synthesize raw attributes into a combined priority function. Jakobović and Budin (2006) proposed a GP-based approach for a dynamic SMSP with minimization of the total weighted tardiness. The role of GP is to find the most suitable priority function by using widely used criteria such as slack, number of remaining jobs, and waiting times. Recent applications of GP have utilized the relationships among more problem-specific attributes, such as the difference between operation’s due dates, that are relevant to scheduling decisions (Zhou et al. 2019).

In the case of other types of constructive heuristics, Chand et al. (1997) introduced rolling horizon procedures (RHP) for $1|r_j|\sum C_j$, which is a dynamic scheduling problem decomposed into a series of smaller sub-problems of the same type. The RHP provides a compromise between exact solution methods and myopic DRs by changing the degree of forward visibility. The computational results showed that RHP is better than DRs in some cases and worse in other cases because the impact of the local objectives on the global one can be ignored (Wang et al. 2005). Chou et al. (2005) also proposed a heuristic algorithm, this one with two phases, that uses a rolling horizon and non-delay concept; this means that machine idleness needs to be avoided on the shop floor and that machines will keep running as long as there is work in process (WIP) in queue.

Unlike constructive heuristics, improvement heuristics start from a given set of initial solutions and exploit the iterative improvement. Congram et al. (2002) introduced a new neighborhood search technique for $1||\sum w_j T_j$ that uses dynamic programming to search an exponential-sized neighborhood in polynomial time. Holthaus and Rajendran (2005) proposed a fast ant-colony optimization (ACO) algorithm for $1||\sum w_j T_j$ based on the use of pheromone trails laid as a medium for communication and feedback among ants. Cakar and Koker (2015) proposed a neuro-hybrid system based on particle swarm optimization (PSO) that improves the obtained solution by using genetic algorithm (GA) and simulated annealing (SA) together as a hybrid system to solve $1|r_j|\sum w_j T_j$.

2.2 Flexible Job Shop Scheduling Problem (FJSP)

An FJSP can be divided into two sub-problems, a routing problem and a sequencing problem. The objective of the routing problem is to assign each operation to an alternative machine and can be considered as a parallel machine problem. The sequencing problem is for sequencing assigned operations to machines and is equivalent to the classical job shop scheduling problem. These two sub-problems have been shown to be NP-hard (Garey, Johnson, and Sethi 1976).

Exact approaches based on mathematical modelling have been used to ensure better performance than other heuristic methods in terms of finding optimal solutions to small- and moderate-sized FJSPs. Fattahi, Mehrabad, and Jolai (2007) used a sequence-position variable-based model to formulate an FJSP. Özgüven, Özbakır, and Yavuz (2010) developed a MILP model to solve FJSPs by using precedence variables. Demir and İşleyen (2013) examined the performances of mathematical models and showed that the MILP model with precedence variables has the least computation time for almost all optimally solved test problems.

However, although exact approaches based on mathematical modelling have been developed, they are inapplicable to large or complex problems. To deal with FJSPs more practically, various dispatching rules (DRs) have been proposed. In general, DRs can be distinguished by their input attributes, such as due dates and processing times, and various attributes of jobs are often combined in practice (Bergmann et al. 2015). To determine which job precedes others, additional attributes such as slack time can be constructed as input data so that DRs can produce more efficient schedules.

Heuristic algorithms also have been used to find the optimal or near-optimal schedule in a reasonable time. Brandimarte (1993) proposed a hierarchical algorithm based on tabu search for FJSPs, where an initial population is generated by using DRs (SPT, EDD, MWRT, ATC, etc.) with fixed routing decisions. Yazdani, Amiri, and Zandieh (2010) proposed a parallel variable neighborhood search (PVNS) algorithm for an FJSP with the objective of minimizing the makespan. The parallelization in this algorithm can enhance the exploration of the search space.

Of the various meta-heuristic algorithms, GAs are considered successful in dealing with FJSPs, as reflected by the growing number of related studies (Çaliş and Bulkan 2015).

Constraint programming (CP) is another problem-solving paradigm that has been widely applied to scheduling, especially flexible and extensible scheduling systems. A key idea of CP is that constraints can be used to reduce the computation time required to solve FJSPs. In CP, constraints are used to check for validity and inconsistencies, remove values from the domains, and deduce new constraints (Baptiste, Le Pape, and Nuijten 2012). Ham and Cakici (2016) tested three MILP models and their CP model to minimize the makespan of an FJSP with parallel batch processing machines; in the results, the proposed CP outperformed all three MIP models.

2.3 Control of Material Handling Vehicles

The major problem for material handling vehicles such as AGVs and AMRs is to identify a set of optimal routes, with consideration of conflicts, whereby a fleet of vehicles can serve given transportation requests. This problem can be categorized into three sub-problems: path finding (PF), vehicle routing (VR), and conflict resolution (CR). This chapter provides a detailed overview of the research streams focusing on three sub-problems (see Table 2.1).

2.3.1 Path Finding (PF)

In general, the goal of PF problems is to find the best path between two locations whereby the sum of the costs of edges is minimized. A map or workspace (e.g. a 2D grid map) can be formulated as a graph G = (V, E) wherein the vertices (V) represent locations and the edges (E) represent segments of road, which are weighted by the time, length, or any cost needed to travel. When finding the shortest path for AMRs, consideration of obstacles is necessary, because it affects the feasibility of paths and the expected arrival time of a transportation request.

Table 2.1 Overview of research related to path finding, vehicle routing, and conflict resolution

Type	Algorithm	Reference
Path finding	Breadth-first search	Moore (1959)
	Dijkstra	Dijkstra (1959)
	A*	Hart et al. (1968)
	Hierarchical A*	Botea et al. (2004)
	FastMap	Cohena et al. (2018)
	D* Lite	Koenig and Likhachev (2002)
Multi-agent path finding	Conflict-based search	Sharon et al. (2015)
	Real-time heuristic search	Sigurdson et al. (2018)
	M*	Wagner and Choset (2011)
	Reinforcement learning	Godoy et al. (2015)
	Deep reinforcement learning	Long et al. (2018)
	Mixed integer programming	Schouwenaars et al. (2001)
	Sequential convex programming	Chen et al. (2015)
	Mixed integer linear programming (MILP)	Yu and LaValle (2016)
	Mixed integer nonlinear programming	Wang et al. (2019)
Vehicle routing	MILP	Savelsbergh and Sol (1995)
	MILP with two-index formulation	Furtado et al. (2017)
	Dynamic programming	Mahmoudi and Zhou (2016)
	Dispatching rules	Ho and Chien (2006); Ho and Liu (2009)
	Double-horizon-based heuristics	Mitrović-Minić et al. (2004)
	Adaptive large neighborhood search	Li et al. (2016)
	Hybrid large neighborhood search	Curtois et al. (2018)
	NSGA-II-based evolutionary algorithm	Phan and Suzuki (2016)
	Evolutionary algorithm with PSO and GA	Muñoz-Carpintero et al. (2015)
Conflict resolution	Collision avoidance by negotiation	Asama et al. (1991)
	Local Collision Avoidance	Guy et al. (2009)
	Structural Control Policy	Reveliotis (2000)
	Distributed control mechanism	Zheng et al. (2013); Chen et al. (2017)
	Behavior-based Multi-Robot Collision Avoidance	Sun et al. (2014)
	Zone Controller Agent (ZCA)	Srivastava et al. (2008)

As the most commonly used approach to finding the shortest path, A* algorithm proposed by Hart et al. (1968) has proven its applicability to a variety of path finding problems, especially in a discrete space (Yuan et al. 2016). Due to its versatility, many variations of A* algorithms such as Hierarchical A* also have been studied (Botea et al. 2004). Similarly, the D* algorithm comes from the term “Dynamic A*”, because the algorithm behaves like A* except that the arc costs can change as the algorithm runs (Koenig and Likhachev 2002). More recently, A* algorithm with preprocessing, called FastMap, inspired by data mining, was proposed to improve performance and shorten computation time (Cohena et al. 2018).

In addition to these algorithms, the concept of multi-agent path finding (MAPF) has been widely studied in robotics. As with the single-agent PF, the goal of MAPF is to find collision-free paths for multiple robots for a given problem with consideration of obstacles, but MAPF is an NP-hard problem even when approximating optimal solutions (Ma et al. 2019). In order to solve MAPF algorithms, various mathematical models (e.g. mixed integer programming, mixed integer linear or nonlinear programming, and Sequential convex programming) have been proposed (Schouwenaars et al. 2001; Chen et al. 2017; Yu and LaValle 2016; Wang et al. 2019). On the other hand, due to MAPF’s NP-hard nature, various heuristics such as M* also have been studied in order to solve it within a reasonable time (Sharon et al. 2015; Sigurdson et al. 2018; Wagner and Choset 2011).

Some of the prior work based on centralized methods assumed that comprehensive knowledge about all agents is given for a central server in order to control their action. These centralized methods have a limitation in that, when scaling to large systems with many robots, their performance can be poor when task reassignments are required frequently. Thus, in more recent times, decentralized multi-robot collision avoidance with deep reinforcement learning has been studied (Godoy et al. 2015; Long et al. 2018). Meanwhile, in order to consider task allocation with MAPF, Liu et al. (2019) studied a Multi-Agent Pickup-and-Delivery (MAPD) problem with TA-Prioritized, which uses prioritized planning to plan collision-free paths on which the robots can serve all of their transportation requests, and determines the sequence of transportation requests for each agent.

2.3.2 Vehicle Routing (VR)

The vehicle routing problem (VRP) with AMRs is a combinatorial optimization problem entailing identification of a set of optimal routes (with consideration of recharging) whereby a fleet of robots can serve transportation requests. Especially, in material handling wherein parts or finished products must be collected from a pickup location and delivered to a paired delivery location, the VRP can be classified as a pickup and delivery problem (PDP).

Due to the need for optimal solutions to the PDP, exact approaches such as mathematical models have been proposed. Savelsbergh and Sol (1995) formulated a mathematical model for the general PDP as well as a dynamic version of the model for re-optimization at a specific point in order to consider newly arriving requests. Also, Furtado et al. (2017) proposed a new mixed integer linear programming (MILP) that allows for the assignment of vehicles to routes explicitly in two-index flow formulations. Mahmoudi and Zhou (2016) had earlier proposed a dynamic-programming-based approach to find optimal solutions for the PDP with a single vehicle.

However, because of the NP-hard nature of the PDP, these exact approaches are applicable only to small-sized problems. Thus, heuristic algorithms are widely studied to find good solutions more quickly. Among heuristics, dispatching rules (DRs) are widely used in practice, particularly in the case of PDPs for AGVs, owing to their short computation times. Ho and Chien (2006) studied the control problem of multiple-load AGVs and proposed DRs for two sub-problems: task-determination and delivery-dispatching. Ho and Liu (2009) addressed the performance of pickup-DRs and load-selection rules for multiple-load AGVs based on a detailed flow chart for control of multiple-load AGVs. In terms of constructive heuristics, Mitrović-Minić et al. (2004) proposed double-horizon-based heuristics for the dynamic PDP-TW wherein future transportation requests are not known a priori.

With regard to metaheuristic algorithms, evolutionary algorithms such as adaptive large neighborhood search (ALNS), genetic algorithm (GA), and particle swarm optimization (PSO) are widely used to solve various PDPs. Especially, hybrid approaches that incorporate a GA into local search methods have been investigated in order to explore possible solutions (Muñoz-Carpintero et al. 2015; Phan and Suzuki 2016). In addition to hybridization, neighborhood-search-based algorithms using a given set of search operators have been studied as well (Li et al. 2016; Curtois et al. 2018).

2.3.3 Conflict Resolution (CR)

The purpose of conflict resolution (CR) is to detect expected conflicts between routes and to resolve those conflicts efficiently. Especially for AGV systems, Reveliotis (2000) specified two types of conflicts (collisions and deadlocks) as well as behaviors such as re-routing and backtracking. However, similarly to PF, the centralized approaches might be inappropriate for solving CR problems within a reasonable time, because behaviors such as re-routing are highly involved with MAPF. Thus, to pursue better scalability, an early attempt to resolve conflicts between AMRs in a decentralized way was presented by Asama et al. (1991). They proposed an intelligent robot system named ACTRESS (ACTor-based Robots and Equipment Synthetic system) that incorporates dynamic path planning into decentralized collision avoidance with negotiations between robots. Researchers have proposed distributed approaches based on message-passing schemes, which resolve local (e.g. pairwise) conflicts without needing to form a joint optimization problem among all members of the team (Zheng et al. 2013; Chen et al. 2017). To solve conflicts locally, Sun et al. (2014) specified eight types of behaviors (e.g. WaitForGoThrough) for avoiding collisions and deadlocks.

In order to implement decentralized concepts, agent-based modelling (ABM) has been applied to robots, especially AGV or AMR systems in material handling. Simulation tools based on ABMs such as AnyLogic and FlexSim are widely used to model the dynamics of crowds and robots in large-scale transportation, logistics, and warehousing models (Abar et al. 2017). For example, Srivastava et al. (2008) proposed a collaborative architecture with rules associated with each agent for finding the conflict-free minimum time motion planning of AGVs that are required to navigate unidirectional and bidirectional flow path network. Guy et al. (2009) also presented a local multi-agent collision avoidance algorithm for real-time simulations.

2.4 Machine Learning Applications in Scheduling and Routing

Although the above-mentioned studies have proposed various approaches for dealing with production scheduling and logistics, two issues persist in practice. The first issue is the computation time. Although exact approaches, such as MILP, can guarantee the optimal solution, they are often impractical due to their relatively long computation times for large problems. Moreover, while heuristic approaches can be applied to large problems, the computation time and quality of the solution vary with algorithm design, because their results remain limited owing to their reliance on randomized natural selection and recombination (Reynolds 1994). Thus, there is a significant need for ML-based application for minimization of computation time and human intervention.

2.4.1 Production Scheduling

More recently, the increasing computational complexity of scheduling problems and computing power have fostered considerable interest in machine-learning applications for scheduling problems. Especially, inductive learning such as DT-based algorithms is an emerging area of research and application, due specifically to its interpretability. To solve a SMSP with a weighted maximum lateness objective, Olafsson and Li (2010) proposed a method to generate a set of rules from optimal solutions using the C4.5 algorithm, proposed by Quinlan (2014). The training data was generated from all pair-wise comparisons between jobs, and additionally, four derivative attributes based on the raw attributes were added. The results showed that the proposed method can find better rules compared with the application of data mining algorithms to the training data. Shahzad and Mebarki (2016) also presented an approach for extraction of DRs using the C4.5 algorithm with pre-defined predictors (e.g. number of jobs in system, average remaining time until due dates). To verify the performance, they also conducted experiments with problem instances for minimization of maximum lateness and compared the results with those for well-known DRs such as COVERT, ATC, and slack. In terms of generating schedules to be learned, Zahmani and Atmani (2018) proposed an approach for $1||\sum w_j T_j$ based on DTs with the C4.5 algorithm from best schedules generated by a hybrid GA. The aim of the approach is to mimic the behavior of best schedules from the hybrid GA while retaining DRs' advantages such as short processing time and reactivity to dynamic scheduling.

In addition to the DT-based algorithms, other classification algorithms have also been applied to discover rules from given schedules. Bergmann et al. (2017) presented an approach for approximation of DRs through various machine-learning techniques, including the naïve Bayes classifier, classification and regression trees (CART), k-nearest neighbors (kNN), and artificial neural networks (ANN). In terms of ensemble learning methods based on DTs, Jun et al. (2019) introduced a random-forest-based approach called Random Forest for Obtaining Rules for Scheduling (RANFORS) to extract DRs from the best schedules. In order to improve the robustness of DT by overcoming overfitting problems, RANFORS applied random forest (RF) with the discretization technique that partitions continuous attributes into a given number of classes.

In order to improve the performance of learning algorithms, construction of new attributes based on the scheduling domain knowledge has been proposed. Li and Olafsson (2005) showed the benefits of constructed attributes that combines raw attributes for simultaneous minimization of the number of nodes in the DTs and maximization of accuracy. Based on their previous work, Li and Olafsson (2010) proposed a two-phase approach for learning DRs with best scheduling practices, but four constructed attributes based on comparisons such as 'Job1 release earlier' and 'Job1 weight higher' had been added manually to improve the accuracy.

More recently, Shahzad and Mebarki (2016) applied new attributes called predictors (e.g. the number of jobs in the system, the percentage of jobs with relatively longer processing times, and the relative tightness ratio). Jun et al. (2019) applied random forests for learning DRs with discretization of continuous attributes as well as constructed attributes. In both of the above-noted studies, constructed attributes, as predefined from the domain knowledge, were supplemented to the training data.

2.4.2 Vehicle Routing

With applications of well-known ML models such as neural networks (NNs), data-driven solution approaches for modern smart delivery systems are an emerging area of research and application, due specifically to the abilities of learning from experience (Nalepa 2020). In the case of supervised learning, Chen et al. (2013) proposed a framework to mimic current dispatch processes and generate solutions for the PDP by learning from historical data. In the paper, the decision-tree algorithm called Logistic Regression Tree with Unbiased Selection (LOTUS) was used. Also, Arnau et al. (2018) proposed a learnheuristic-based approach that integrates a multiple linear regression model within a metaheuristic framework.

In addition to the above supervised learning, Nazari et al. (2018) proposed an end-to-end framework for solving the vehicle routing problem (VRP) using reinforcement learning. In this approach, they trained a single policy model that finds near-optimal solutions for a broad range of problem instances of similar size. In the case of autonomous taxis and ridesharing vehicles, reinforcement-learning-based algorithms with decentralized or distributed learning for PDPs were broadly studied (Rahili et al. 2018; Shi et al. 2019).

CHAPTER 3. SINGLE MACHINE SCHEDULING PROBLEM

A part of this chapter is published in *International Journal of Production Research*:

Jun, S., & Lee, S. (2020). Learning dispatching rules for single machine scheduling with dynamic arrivals based on decision trees and feature construction. *International Journal of Production Research*, DOI: 10.1080/00207543.2020.1741716.

3.1 Introduction

In solving the dynamic single-machine scheduling problems (SMSPs), dispatching rules (DRs) have played an important role; in fact, their development has drawn great attention from researchers and practitioners. Although other solution approaches such as mathematical models and heuristics might outperform them, DRs are frequently used in practice due to their ease of implementation and quick computation time (Tay and Ho 2008).

The further development of more and more efficient DRs necessitates close investigation of the important attributes that potentially affect the performance. However, due to the increasing complexity of manufacturing environments, it is often difficult to consider all attributes of the shop floor. Furthermore, much of the underlying logics of a schedule might be implicit, and, as such, challenging to capture intuitively. Thus, further research on learning of efficient DRs from previous or good schedules by consideration of various attributes is a significant step towards increasing the possibilities and potentialities of field application.

The remainder of this chapter is organized as follows: Chapter 3.2 defines the SMSP with the aim of minimizing the total weighted tardiness, and Chapter 3.3 presents its solution: the proposed DT-based approach with feature construction. Chapter 3.4 discusses the experiments conducted to verify the validity of the proposed approach. Finally, Chapter 3.5 summarizes conclusions and contributions.

3.2 Problem Definition

Using the well-known three-field notation (Pinedo 2012), the dynamic SMSP for minimization of the total weighted tardiness can be denoted by $1\left|online\text{-}time, r_j\right|\sum w_j T_j$, and even its offline version is NP-hard in the strong sense (Lawler 1977). The basic assumptions followed in this paper are as follows:

- The shop floor has one machine and the machine cannot process more than one job simultaneously.
- Each job has four attributes, processing time p_j, release date r_j, due date d_j, and weight w_j that represents the importance of job j.
- n jobs are released over time and are processed once on the machine without preemption.
- The attributes of a job are unknown in advance unless the job is currently available at the machine.
- The objective is to determine a sequence of jobs on the machine in order to minimize the total weighted tardiness ($\sum w_j T_j$), which is defined as $\sum w_j \cdot max(0, C_j - d_j)$, where C_j is the completion time of job j.

In order to find the optimal or best feasible solutions for learning data, we implemented a mathematical model for SMSPs under offline setting wherein release dates are known at the beginning. The mathematical model for the offline SMSP for minimization of the total weighted tardiness proposed by Chou et al. (2005) is presented.

3.2.1 Indices and sets

i, j	jobs ($i, j \in N$)

3.2.2 Parameters

p_i	the processing time of job i
d_i	the due date of job i
r_i	the release date of job i
w_i	the weight of job i
M	a large number

3.2.3 Decision variables

X_{ij}	If job j is scheduled after job i, $X_{ij} = 1$; otherwise $X_{ij} = 0$.
C_i	the completion time of job i
T_i	the tardiness of job i ($T_i = Max(0,\ C_i - d_i)$)

3.2.4 Mathematical Formulation

$$Min \quad F = \sum_{i \in N} w_i \cdot T_i$$

$$s.t. \quad X_{ij} + X_{ji} = 1 \quad \forall i, j \in N, i \neq j \tag{1}$$

$$C_i \geq r_i + p_i \quad \forall i \in N \tag{2}$$

$$C_i - C_j + \mathrm{M} \cdot X_{ij} \geq p_i \quad \forall i, j \in N, i \neq j \tag{3}$$

$$C_i - d_i \leq T_i \quad \forall i \in N \tag{4}$$

and

$$X_{ij} \in \{0,1\} \quad \forall i, j \in N, i \neq j$$

$$C_i \geq 0, T_i \geq 0 \ \forall i \in N$$

Constraint (1) represents the sequence relationship between any two jobs. Constraint (2) guarantees that the completion time of a job should be greater than the sum of its processing time and the current time. Constraint (3) determines the completion time based on X_{ij} by using a large number M. Constraint (4) determines the tardiness of jobs.

3.3 Proposed Methodology

In this subchapter, a new approach called Generation of Rules Automatically with Feature construction and Tree-based learning (GRAFT) is proposed. GRAFT proceeds in two phases: learning DRs from schedules and improving the DRs with feature construction based on genetic programming. The objective of the first phase is to find efficient DRs based on given schedules, which can be obtained from previous operation or solution approaches. With the application of the first phase, scheduling knowledge of why one job is dispatched ahead of another can be extracted without any information about the objective function.

The second phase is designed for improvement of the performance of DRs with feature construction based on a set of problem instances for training. The goal of this phase is to discover a new attribute by combining various attributes and to use it to generate improved DTs with the combinations of existing attributes. The distinguishing feature of GRAFT is that it has three modules (retrieval of scheduling decisions, construction of attributes by comparison, and feature construction) for improvement of DTs with the aim of better performance and robustness. The proposed approach can be summarized as follows (see Figure 3.1):

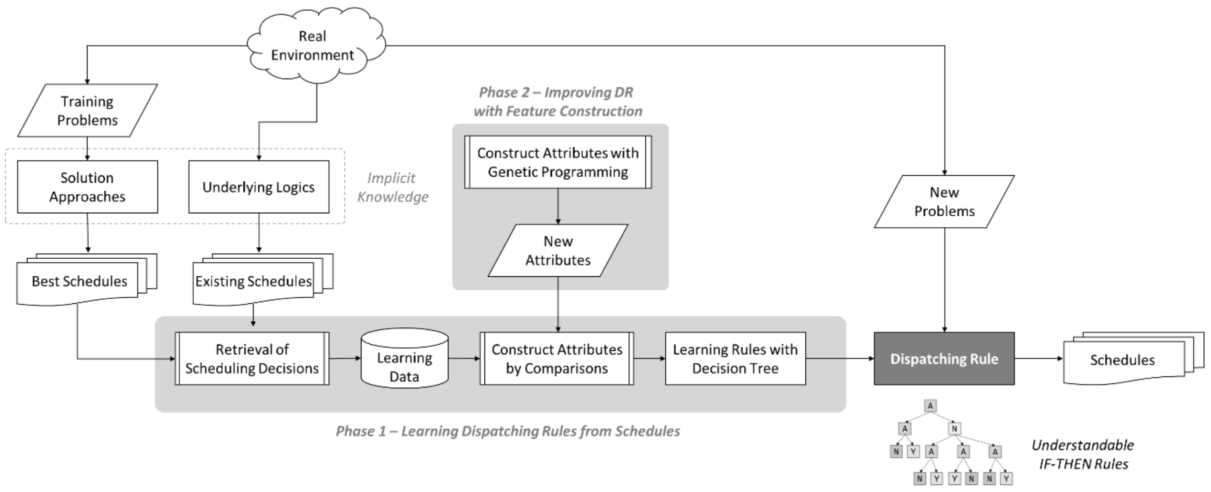

Real Environment
Training Problems
Solution Approaches
Underlying Logics
Implicit Knowledge
Best Schedules
Existing Schedules
Phase 2 – Improving DR with Feature Construction
Construct Attributes with Genetic Programming
New Attributes
New Problems
Retrieval of Scheduling Decisions
Learning Data
Construct Attributes by Comparisons
Learning Rules with Decision Tree
Dispatching Rule
Schedules
Phase 1 – Learning Dispatching Rules from Schedules
Understandable IF-THEN Rules

Figure 3.1 Overall GRAFT framework

3.3.1 Learning Dispatching Rules from Schedules

3.3.1.1 Retrieval Procedures of Scheduling Decisions (RPSD)

Based on the RHP with the non-delay concept proposed by Chou et al. (2005), the retrieval procedures of scheduling decisions (RPSD) was implemented. According to the non-delay concept, when the machine finishes the process of one job, the next job should be chosen immediately among jobs in queue. RPSD finds which jobs have arrived by means of the rolling of a timer. After that, every job in queue is compared to the 'selected' job at timer to retrieve why the selected job was dispatched ahead of another 'candidate' job with consideration of various attributes. Based on a schedule list and its problem instance, a training dataset (Training) is generated as shown in Figure 3.

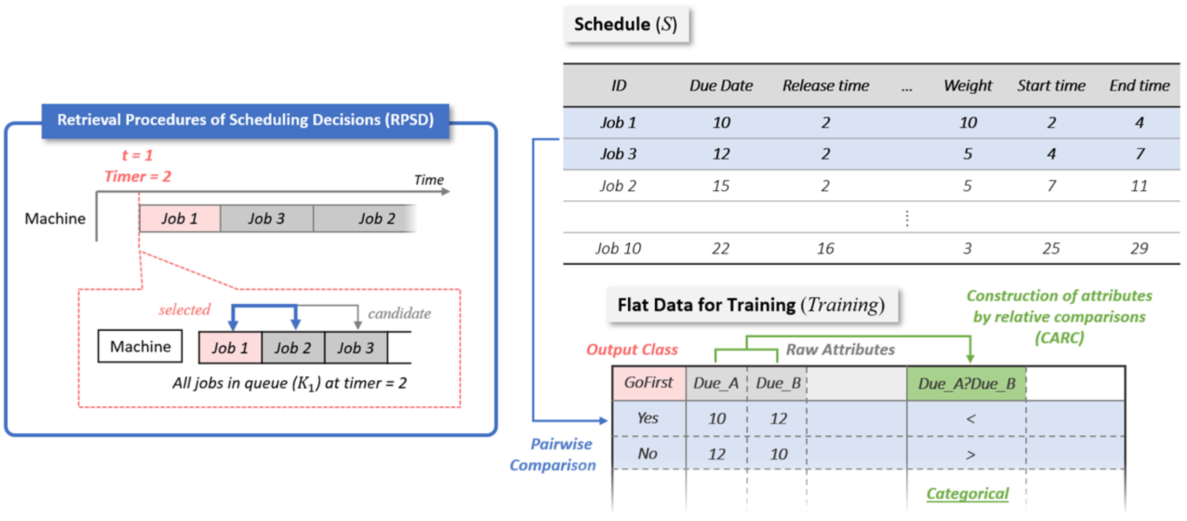

Schedule (S)
ID | Due Date | Release time | ... | Weight | Start time | End time
Job 1 | 10 | 2 | 10 | 2 | 4
Job 3 | 12 | 2 | 5 | 4 | 7
Job 2 | 15 | 2 | 5 | 7 | 11
Job 10 | 22 | 16 | 3 | 25 | 29
Retrieval Procedures of Scheduling Decisions (RPSD)
t = 1
Timer = 2
Time
Machine
Job 1
Job 3
Job 2
selected
candidate
All jobs in queue (K_1) at timer = 2
Pairwise Comparison
Flat Data for Training (Training)
Construction of attributes by relative comparisons (CARC)
Output Class
Raw Attributes
GoFirst | Due_A | Due_B | Due_A?Due_B
Yes | 10 | 12 | <
No | 12 | 10 | >
Categorical

Figure 3.2 Retrieval of scheduling decisions and construction of new attributes from all pair-wise comparisons between jobs

Algorithm 1 – RPSD

```
**Input**: Set of scheduled jobs ($S$)
**Output**: Flat data for training ($Training$)
$timer = 0$
**for** ($t = 1$ to number of jobs in $S$) {
selected ← t th job
timer ← the start time of selected
find a list of jobs ($K_t$) that arrived and are available at timer
**if** (size of $K_t = 1$) **then** { go to the next iteration }
**else** {
**for each** (job candidate in $K_t$ – selected){
call AppendDecision (selected, candidate, Yes)
call AppendDecision (candidate, selected, No)
} **end for**
  } **end if**
} **end for**
**return** Training

**Function** AppendDecision (Job A, Job B, GoAFirst){
add raw attributes
append a row with values of the output class GoAFirst and all attributes to Training
**return**
}
**End Function**
```

Based on a given schedule, RPSD disintegrates it into a series of iterations, which are distinguished by decision points (t). Each iteration involves three steps: identifying available jobs (K_t) at the current decision point t, finding the other candidate jobs, and converting the scheduling decision into flat data with attributes. The detailed procedures of the RPSD are presented in the following Algorithm 1.

3.3.1.2 Construction of Attributes by Relative Comparisons (CARC)

Olafsson and Li (2010), in order to extract underlying scheduling decisions more effectively, introduced four additional attributes with two classes (≤ or >) that indicate whether the first job in the row is released earlier, due earlier, and has lower processing time or higher weight. In order to improve the performance and robustness of DT-based algorithms, the concept with consideration of feature interactions by comparing the same attributes for two jobs is extended more specifically. For example, the comparison of due dates between two jobs (d_A and d_B) can generate a constructed attribute with three classes ($d_A < d_B$, $d_A = d_B$, $d_A > d_B$).

As shown in Table 3.1, commonly used attributes are included based on the information about jobs and the shop floor at a specific moment (Nguyen 2016). This CARC process is also applied to newly generated attributes by feature construction in Chapter 3.3.2.

Table 3.1 Raw attributes and constructed attributes with CARC

Source	Attribute Type (Number of attributes)	Attribute Description
Sequencing jobs	Raw attributes (8)	Due date of J1, release date of J1, weight of J1, processing time of J1, due date of J2, release date of J2, weight of J2, processing time of J2
	Categorical constructed attributes (4)	Comparison between due dates, comparison between release dates, comparison between weights, comparison between processing times
Shop floor	Raw attributes (5)	Current time, number of jobs in queue, utilization of the machine, total waiting time in queue, sum of processing times of jobs in queue

3.3.1.3 Learning Dispatching Rules with Decision-tree-based Algorithm

In order to constitute a DR by mimicking behaviours from the training data, a DT is generated by using the C4.5 algorithm, due specifically to its interpretability. Based on this DT, a DR determines the next job to be processed at each decision point similarly to RPSD but in reverse order. First, jobs that can be processed at a *timer* are sorted by the output classes of the DT. For example, when an output class '*GoFirst*' of the DT is '*Yes*', it means that Job 1 tends to precede Job 2; thus, Job 1 is selected. By comparing all output classes for jobs in queue at t, the next job to be processed is selected. Based on the assignment of all jobs to the machine, a detailed schedule and its performance are determined. The detailed procedures of the DR with DT are presented in the following Algorithm 2.

Algorithm 2 – DR with DT

```
Input: $DT$ learned by the training data
Output: Set of scheduled jobs ($S$), Total weighted tardiness ($wT$)
$timer = 0, wT = 0, S = \emptyset$
while (all jobs are processed) {
find a list of jobs ($K$) in queue at $timer$
if ($K = \emptyset$) then {increase $timer$}
else if ($size\ of\ K = 1$) { $timer \leftarrow$ sum of $timer$ and the processing time of the job in $K$
add the job in $K$ to $S$ and $wT \leftarrow wT + max(0, timer -$ the due date of the job in $K$) }
else { $selected$ = the first job in $K$
for ($i = 2\ to\ size\ of\ K$){
if ($Decide\_DT(selected,$ $i$th job in $K$) is 'No') { $selected$ = $i$th job in $K$ }} end if
$timer \leftarrow$ sum of $timer$ and the processing time of $selected$
add $selected$ to S and $wT \leftarrow wT + max(0, timer -$ the due date of $selected$)
  } end if
} end while

Function $Decide\_DT$(Job $A$, Job $B$){
add new attributes by comparing two jobs
return the output from the $DT$}
End Function
```

3.3.2 Improving Learned DRs with Feature Construction by Genetic Programming

In this section, a new approach for improvement of a learned DR in the previous phase with feature construction by GP (FCGP) is presented. The purpose of GP is to discover hidden relationships between attributes inferring new composite attributes. By incorporating new attributes generated into a DT, FCGP can improve its performance while keeping the interpretability with compact sizes. For example, Smith and Bull (2005) showed that the hybridization of GP and C4.5 outperforms the standard C4.5 and that the constructed features helped DT to achieve smaller error rates with much smaller sizes. The overall FCGP can be summarized as shown in Figure 3.3.

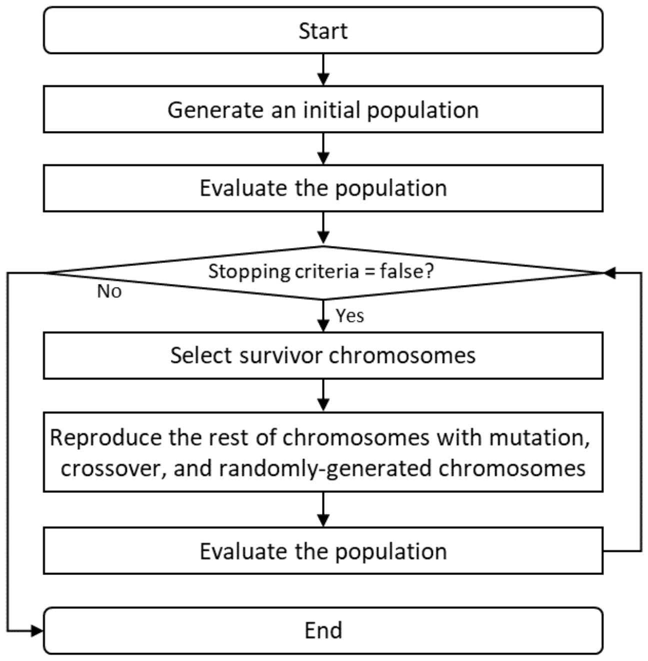

Start
Generate an initial population
Evaluate the population
Stopping criteria = false?
No
Yes
Select survivor chromosomes
Reproduce the rest of chromosomes with mutation, crossover, and randomly-generated chromosomes
Evaluate the population
End

Figure 3.3 Overall FCGP framework

3.3.2.1 Chromosome Representation

To represent the expression for constructing a new attribute, the postfix notation for chromosome representation proposed by Dabhi and Vij (2011) is used. Unlike the infix notation, the postfix notation changes the position of the operators by moving them towards the right of the operands. In the case of postfix notation, stack can be used to transform a chromosome into an expression as shown in the below steps.

Step 1 Set the current gene to the start gene of a chromosome.

Step 2 Identify the type of the current gene. If the current gene represents an operand, go to Step 3; otherwise go to Step 4.

Step 3 Push the operand that represents an attribute on the stack and go to Step 5.

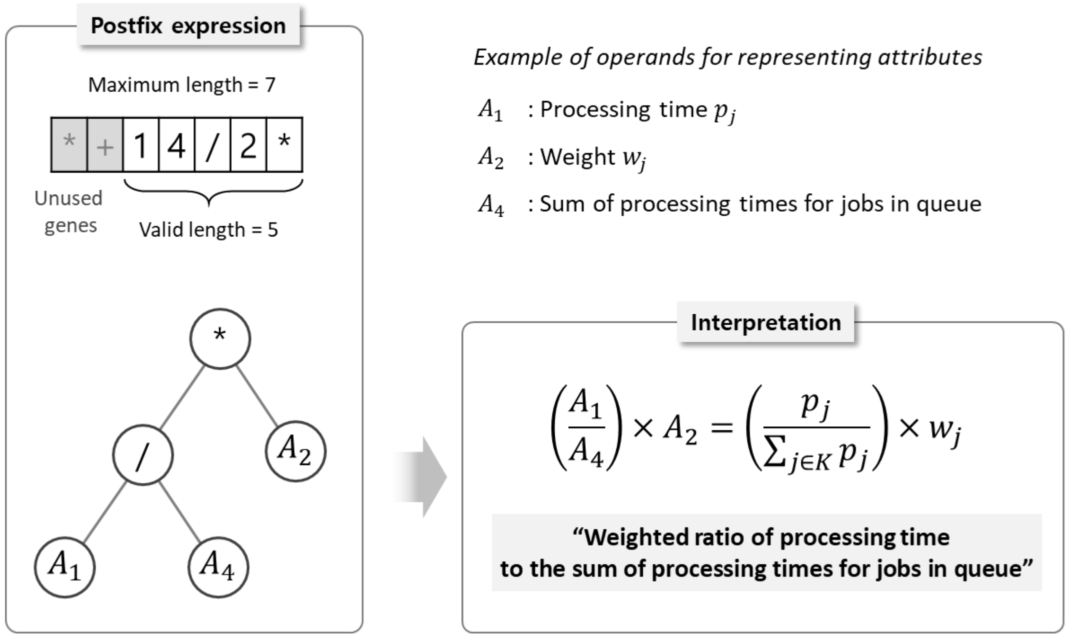
Step 4 Check whether there is the required number of operands in the stack. If the number of operands in the stack is not sufficient for the operator, mark the gene as 'unused' and go to Step 5; Otherwise, pop operands from the stack, combine operands and the operator into an intermediate expression, and push it on the stack.

Step 5 Set the next gene to the current gene and go to Step 2 until the current gene is the last gene of the chromosome.

Based on the previous literature on GP for scheduling (Nguyen et al. 2017; Zhou et al. 2019), the operators and operands used in this dissertation are described in Table 3.2.

Table 3.2 Operands and operators for genetic programming

Type of gene	Description
Operand (12)	Due date, release date, weight, processing time, number of jobs, sum of processing times of all jobs, current time, number of remaining jobs, number of jobs in queue, utilization of the machine, total waiting time in queue, sum of processing times of jobs in queue
Operator (4)	$+, -, \times, \div$



Postfix expression
Maximum length = 7
* + 1 4 / 2 *
Unused genes
Valid length = 5
*
/
A_2
A_1
A_4
Example of operands for representing attributes
A_1 : Processing time p_j
A_2 : Weight w_j
A_4 : Sum of processing times for jobs in queue
Interpretation
$\left(\frac{A_1}{A_4}\right) \times A_2 = \left(\frac{p_j}{\sum_{j \in K} p_j}\right) \times w_j$
"Weighted ratio of processing time to the sum of processing times for jobs in queue"

Figure 3.4 Chromosome representation and decoding procedure with postfix method

Figure 3.4 represents a chromosome in postfix notation, maximum and valid lengths, and the tree representation of the chromosome. The maximum and minimum lengths are specified by users. The valid length of a chromosome is defined as the number of used genes and it should be equal to or greater than the minimum length. Even if the maximum length is fixed, expressions of chromosomes can have different sizes and structures according to the valid length. For example, the simplest expression can be only one attribute (i.e. when all genes of a chromosome except one operand are operators) and the longest expression can be a chromosome that uses all the genes are used. In order to maintain the understandability of new attributes generated by genetic programming, the four basic operators along with the limited maximum length of a chromosome were used.

Chromosomes in the initial population are generated randomly, and the random generation process for each chromosome is repeated until the valid length of the chromosome is equal to or greater than the minimum length.

3.3.2.2 Reproduction

A new population for the next generation is generated by reproduction with two operators (crossover and mutation) based on the chromosomes of the current population. To change the expression of a single chromosome, the mutation operator selects a gene randomly in the chromosome and reassigns it to another value. In the case of crossover, a one-point crossover is applied because it strongly preserves the good characteristics of two chromosomes. Examples of the implementation of the mutation and crossover operator are shown in Figures 3.5 and 3.6, respectively.

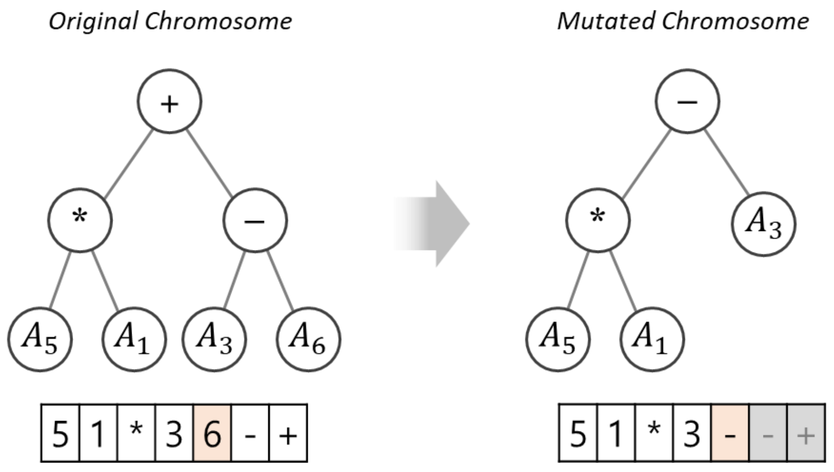


Figure 3.5 Illustration of mutation operation

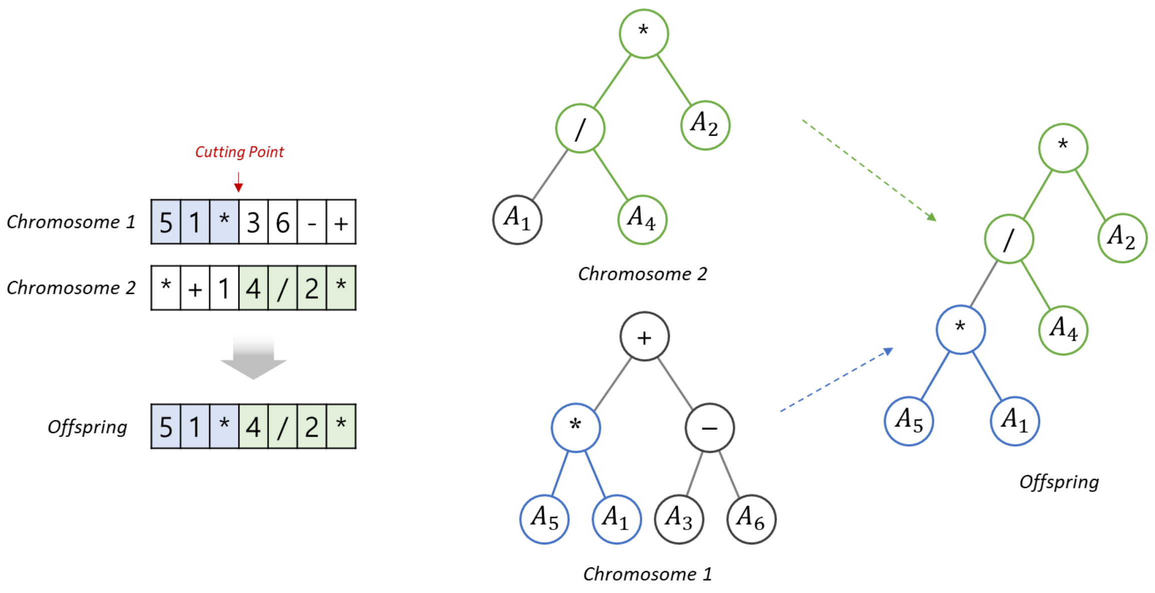


Figure 3.6 Illustration of crossover operation

3.3.2.3 Evaluation and Selection

After the generation of chromosomes according to the given number of populations, the chromosomes are evaluated by the total weighted tardiness. To evaluate the performance of a chromosome, the chromosome is converted into an expression by postfix notation, and a new attribute constructed by the expression is added to the original flat data for training. Based on a DT from the updated training data (*DT_GP*), a new DR is generated. The detailed procedures of the DR with *DT_GP* are shown in Algorithm 3.

Algorithm 3 – DR with DT and FCGP

```
Input: DT_GP learned by the training data
Output: Set of scheduled jobs (S), Total weighted tardiness (wT)
$timer = 0, wT = 0, S = \emptyset$
while (all jobs are processed) {
find a list of jobs (K) in queue at timer
if ($K = \emptyset$) then {increase $timer$}
else if ($size\ of\ K = 1$) { timer ← sum of timer and the processing time of the job in K
add the job in K to S and wT ← wT + max(0, timer − the due date of the job in K) }
else {
selected = the first job in K
for ($i = 2\ to\ size\ of\ K$){
```

```
if (Decide_DT_GP(selected, $i$th job in $K$) is ‘No’) { selected = $i$th job in $K$ } end if
} end for
timer ← sum of timer and the processing time of selected
add selected to $S$ and $wT$ ← $wT$ + max(0, timer − the due date of selected)
  } end if
} end while
return $S$ and $wT$

Function Decide_DT_GP(Job $A$, Job $B$, Chromosome){
add new attributes by comparing two jobs
add new attributes by comparing between attributes generated by Chromosome of $A$
and $B$.
return the output from the DT_GP}
End Function
```

The evaluation of a chromosome is determined by the fitness function, which is defined as the average total weighted tardiness of problem instances for training, which is calculated by the DR with *DT_GP*. The termination criterion is the number of generations and the selection of surviving chromosomes is determined by the tournament selection method to preserve the desirable characteristics of chromosomes for the next generation.

3.4 Experimental Results

In this section, the training data from the best solutions for each training problem was generated and simulation experiments on similar and different types of problem instances were performed to compare the performance and robustness of the existing DRs.

3.4.1 Generation of Problem Instances and Training Data

Problem instances were randomly generated by using parameters from the method proposed by Chu (1992) and Akturk and Ozdemir (2000). Each problem instance is generated at random from four uniform distributions of w_j, r_j, p_j, and $d_j - (r_j + p_j)$. The distribution of p_j and w_j is always between 1 and 10. The distribution of r_j was generated from a uniform distribution ranging from 0 to $\alpha \sum p_j$, where four different α values [0.0, 0.5, 1.0, 1.5] were used. In the case of due dates, instead of generating them directly, slack times ($d_j - (r_j + p_j)$) were generated from a uniform distribution between 0 and $\beta \sum p_j$ where three different β values [0.05, 0.25, 0.5] were used. To generate 120 problems for each size, a total of 12 problem instances based on combinations of α and β were considered, and 10 replications were taken for each combination.

To solve the defined MILP in Chapter 3.2, IBM ILOG CPLEX optimizer version 12.7.1 with default settings was used and the run time was limited to 3,600 seconds. Also, in order to reduce the searching space and improve the solution quality, four properties proposed by Chou et al. (2005) were applied. To obtain the training dataset for learning, the optimal or best feasible solutions were obtained under offline setting wherein release dates are known at the beginning. To generate training problem instances (48 problems), 12 problem instances based on combinations of α and β were considered and 4 replications were taken for each combination. The average performances of solutions for the offline SMSPs obtained by solving MILPs are compared in Table 3.3.

Table 3.3 Results summary of optimal or best feasible solutions for offline SMSPs

	Offline MILP
Average $\sum w_i T_i$	1712.97
Average CPU time (s)	1800.53

As shown in Figure 3.1, the test set of problem instances was used as a training set, and the best solutions for the training set of problems were transformed into the training data. The training data contained 17440 instances, where each class refers to a scheduling decision derived by RPSD as described in Chapter 3.3.1.1.

3.4.2 Comparison among Different Dispatching Rules Generated by Learning Algorithms for Training Data

All of the algorithms were coded in C#, and the experiments were run on an Intel Xeon E5-1620 3.6 GHz processor with 16 GB of RAM. To compare the performances of the different learning techniques, four classification algorithms (C4.5 without FCGP, k-nearest neighbors (kNN), random forest (RF), and artificial neural network (ANN)) were also implemented. The detailed experimental parameters of the learning algorithms are listed in Table 3.4.

Table 3.4 Parameters of learning algorithms and GRAFT

Algorithm	Parameter	Value
C4.5	Maximum height of trees	3
k-nearest neighbors (kNN)	Number of neighbours (*k*)	3
Random forest (RF)	Maximum height of trees	10
	Number of trees	75
	Sample ratio	0.8
Artificial neural network (ANN)	Training algorithm	Levenberg–Marquardt
	Number of hidden neurons	50
	Number of epochs	20
	Weight initialization	Nguyen-Widrow
	Activation function	Bipolar sigmoid
GRAFT	Population size	100
	Number of generations	10
	Mutation rate	0.3
	Crossover rate	0.3
	Tournament size	5
	Maximum length	10

The results for each algorithm are summarized in Table 3.5 below. To verify the performances of the learning algorithms for the training data, the average total weighted tardiness was compared on 48 problem instances of the training set, the computation time for training, and the classification accuracy, as shown in the table. In terms of the average computation times for scheduling, all of the DRs could find solutions for the training problem instances in less than 1 second.

In terms of the average total weighted tardiness for the training problems, two DT-based algorithms (C4.5 and GRAFT) could find better DRs with FCGP than could the other algorithms, whereas the other algorithms' classification accuracies were higher than the DT-based algorithms'. One possible explanation for this result is that the DT-based algorithms are designed to discover general 'rules'; thus, they appear to find more generalized DRs at the expense of accuracy. Two DTs are shown in Figure 10, with new FCGP-generated attributes highlighted in yellow.

Table 3.5 Average performances of DRs generated by learning algorithms for training data

	Accuracy (%)	Average $\sum wT$ for training set	Computation time for training (s)
C4.5	85.21	2840.56	5.42
ANN	92.69	3784.46	171.62
RF	94.84	3189.1	183.54
kNN	98.28	2976.5	4.01
GRAFT	93.27	2602.77	15309.88

3.4.3 Comparison among Algorithms for Similar Types of Problem Instances with Training Data

To validate the competitive performance of a DR learned by GRAFT, six well-known DRs: (SPT, WSPT, EDD, Slack, ATC, and COVERT) were implemented (Jouglet et al. 2008). In addition to the average total weighted tardinesses, the relative deviation index (RDI) was applied to compare the relative performances of the DRs with HGA. The RDI of the kth experiment was calculated as shown in Equation (5) (Akhshabi, Tavakkoli-Moghaddam, and Rahnamay-Roodposhti 2014):

$$RDI_k = \frac{F_k - Min_k}{Max_k - Min_k} \times 100 \tag{5}$$

where F_k is the total weighted tardiness obtained for the kth experiment, and Max_k and Min_k are the best and worst solutions in the kth experiment, respectively.

The average total weighted tardiness, computation times, and RDIs of the DRs and MILP over 120 problem instances for each α and β are compared in Table 3.6. Also, the average RDIs for each α and the best RDIs for each test set are summarised in Figure 3.7. The figure shows the comparison between DRs with different α levels determining the range of release dates (r_j). For example, if α is zero, all the jobs are ready to be processed at time 0 ($r_j = 0$); otherwise, each job j arrive at r_j, which is randomly generated from 0 to $\alpha \sum p_j$.

Table 3.6 Average performances of DRs for similar problem instances with training set

	Average $\sum wT$	Average CPU time (s)	Average RDI (%)
Offline-MILP	1746.09	1904.48	0
SPT	13037.98	< 1	82.38
WSPT	4990.08	< 1	29.76
EDD	3516.27	< 1	25.53
Slack	3850.46	< 1	31.54
ATC	3127.18	< 1	18.88
COVERT	10504.49	< 1	68.29
C4.5	2810.49	< 1	14.83
ANN	3651.03	< 1	30.22
RF	3215.33	< 1	22.35
kNN	2984.92	< 1	17.87
GRAFT	**2560.53**	< 1	**9.91**

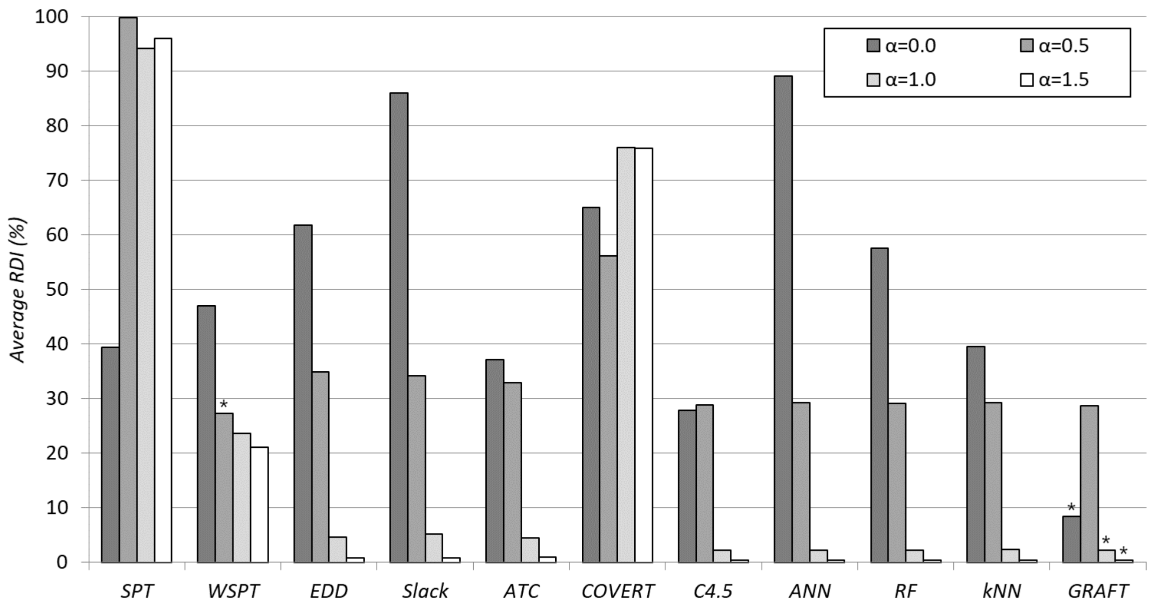
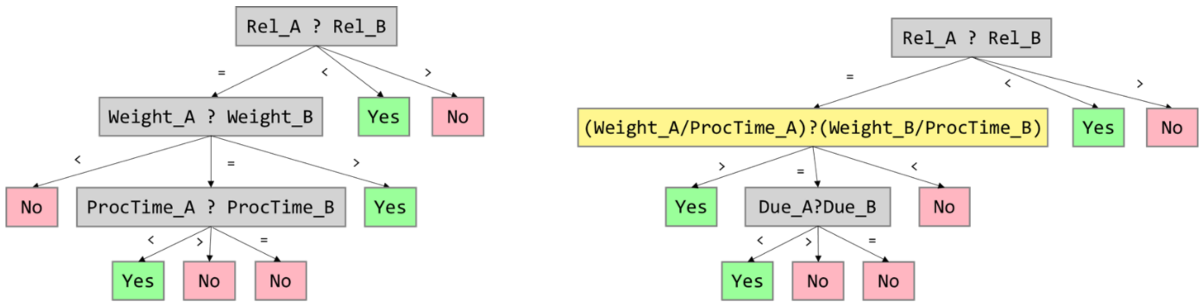


Figure 3.7 Average RDIs of DRs for similar problem instances with different α values. The best RDIs for each α are noted by an asterisk (*).

Among the DRs, GRAFT performed the best in terms of the average total weighted tardiness for problem instances that were similar to the training sets. In terms of relative performance, the results showed that GRAFT still outperformed the other DRs in that it yielded the smallest average

RDI value for most cases. Thus, GRAFT has an advantage over the other DRs in terms of finding robust and good solutions for problems that are similar to the training problems.

To delineate the changes in tree structures by FCGP, two DTs, a DT generated by C4.5 only with CARC and another DT generated by the entire GRAFT, are illustrated as shown in Figure 3.8. The improved DT by GRAFT has a new attribute (highlighted in yellow), and the new branch with the attribute was 'grafted' to a DT for achieving higher performance relative to the DT without FCGP.



(a) Decision tree generated by C4.5 (b) Decision tree improved by FCGP

Figure 3.8 Illustration of DTs generated by C4.5 without FCGP and with FCGP

3.4.4 Comparison among Algorithms for Different Types of Problem Instances with Training Data

In order to check the robustness of performance for different problems with the training problems, two sets of larger problem instances (50 and 100 jobs) were generated. In the same way as for the training problems, 120 problem instances for each α and β were generated for each set. To generate DRs for larger problems, the same training data as in Chapter 3.4.3 was used for verification of the generalizability. The average total weighted tardiness, computation times, and RDIs of the DRs and MILP over 120 problem instances for each number of jobs are compared in Table 3.7. Also, the average RDIs for the respective test sets and the best RDIs for each test set are summarized in Figure 3.9.

The results indicated that the DR of GRAFT still produced the lowest average total weighted tardiness, as shown in Table 9 and Figure 10. The results also indicated that the gap of the average

RDIs between GRAFT and the other DRs continued to broaden when the size of the problem increased and the number of jobs in queue increased (the α values determining the range of release dates decreased).

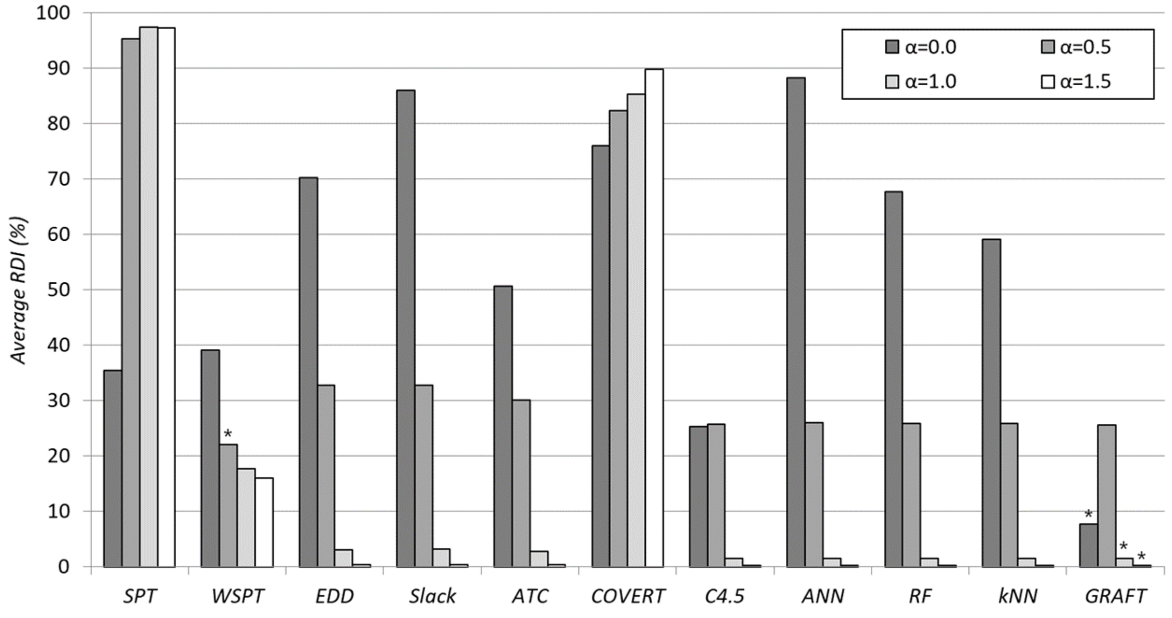
In order to show the incremental improvements of GRAFT in fixed times, the changes on the average RDIs for different computation times were checked by changing termination condition to the maximum computation time. Figure 3.10 shows that the average RDI of a new attribute discovered by GRAFT was gradually improved and converged into a certain level as the maximum computation time increased. In addition, for all of the different numbers of jobs and computation times, GRAFT outperformed the other algorithms in terms of the average RDI as well as total weighted tardiness. Thus, the results showed that GRAFT with a fixed time also provides better performance than does C4.5.

In summary, the simulation results demonstrated that GRAFT could offer good performance for a larger number of jobs compared with the others. In addition, GRAFT appears to find more generalized DRs with FCGP, because the average total weighted tardiness and RDI for different types of problems are smaller than for the other algorithms without FCGP. Thus, the DR by GRAFT, with its quicker computation speed, appears to be more applicable in terms of scalability for extremely large problems in the real world.

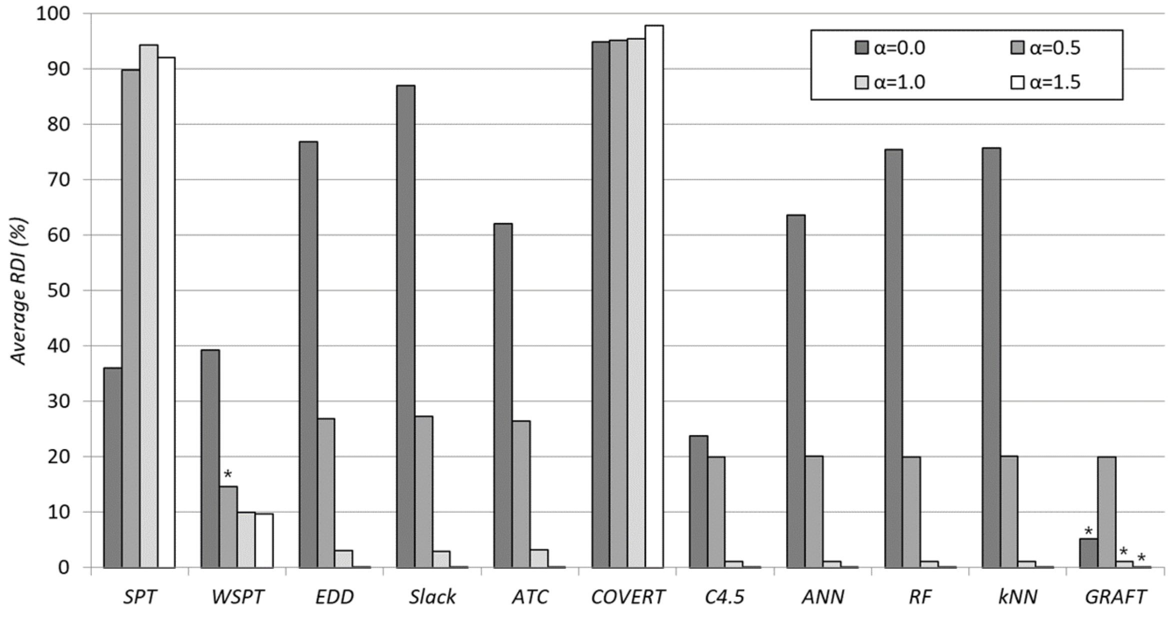
Table 3.7 Average performances of DRs over two test sets with different number of jobs

	50 jobs			100 jobs		
	Avg. $\sum wT$	Avg. CPU time (s)	Avg. RDI (%)	Avg. $\sum wT$	Avg. CPU time (s)	Avg. RDI (%)
Offline-MILP	4743.48	2156.27	0	19747.73	2185.43	0
SPT	39072.34	< 1	81.34	164333.9	< 1	78.05
WSPT	12346.18	< 1	23.72	42072.83	< 1	18.41
EDD	10143.41	< 1	26.61	41051.33	< 1	26.72
Slack	10867.92	< 1	30.64	42788.44	< 1	29.34
ATC	9096.47	< 1	20.97	38698.88	< 1	22.98
COVERT	37158.72	< 1	83.37	180240.28	< 1	95.87
C4.5	7625.59	< 1	13.22	29812.33	< 1	11.23
ANN	10235.93	< 1	29.02	35868.68	2.38	21.21
RF	9407.83	< 1	23.86	37793.68	2.32	24.15
kNN	9096.65	< 1	21.74	37838.95	3.34	24.23
GRAFT	**6917.74**	< 1	**8.79**	**26962.43**	< 1	**6.57**

Note: The bold value indicates the best performance among the DRs.



(a) Average RDIs for 50 jobs with different α levels



(b) Average RDIs for 100 jobs with different α levels

Figure 3.9 Average RDIs of DRs for two test sets with different α levels. The best RDIs for each test set are noted by an asterisk (*).

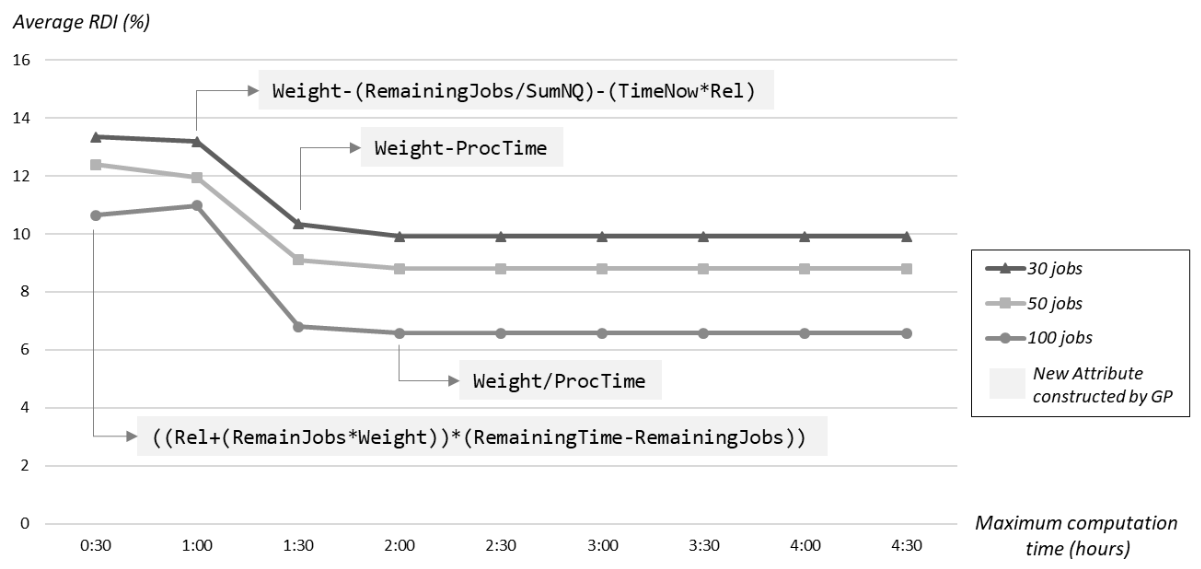


Figure 3.10 Performance improvements on average RDIs with different maximum computation times

3.5 Chapter Summary

In this chapter, the SMSP with dynamic arrivals in order to minimize the total weighted tardiness is analyzed. In order to solve the SMSP, an approach called GRAFT that consists of two phases with three sub-modules is proposed: RPSD, CARC, and FCGP. By analyzing scheduling data from given schedules, DRs were extracted as a DT that can be expressed in a set of DTs or IF-THEN rules. To extract DRs more effectively, a mathematical model for the offline SMSP is formulated. Also, an automatic process for discovering the best combination of attributes with GP is proposed to achieve high and robust performance. The results of simulations showed that the new DR generated and improved by GRAFT outperformed prevalent DRs in terms of average performance and robustness, as it produced a smaller average total weighted tardiness and RDI values.

The major contribution of this chapter is the development of a new approach for capturing both explicit and implicit knowledge from given schedules with less human intervention. Also, to improve the robustness and performance of DT-based algorithms regardless of the size of problems, the proposed GRAFT can consider more comprehensive attributes with CARC and find new attributes automatically by utilizing the evolutionary process of GP. Due to the ability to extract DRs from either previous schedules or best schedules, GRAFT can help schedulers to discover underlying logics that they might not be able to realize by themselves. Moreover, schedulers can modify the DRs generated by GRAFT, because those rules are represented in understandable formats as DTs or rule sets. This interpretability of DTs enables experts to gain a better insight into the learned model and overcome the ‘black box’ problem, which is important to many real-world applications.

CHAPTER 4. FLEXIBLE JOB SHOP SCHEDULING PROBLEM

A part of this chapter is published in *International Journal of Production Research*:

Jun, S., Lee, S., & Chun, H. (2019). Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research*, 57(10), 3290-3310

4.1 Introduction

Although the previous studies in Chapter 2.2 have proposed various approaches for dealing with FJSPs, in practice, two issues persist. The first issue is the computation time. Although exact approaches, such as MILP, can guarantee the optimal solution, they are often impractical due to their relatively long computation times for large problems. Moreover, while heuristic approaches can be applied to large problems, the computation time and quality of the solution vary with algorithm design, because their results remain limited owing to their reliance on randomized natural selection and recombination (Reynolds 1994).

Another issue is the consideration of a variety of information for FJSPs. Although a number of previous studies have investigated efficient DRs with raw and constructed attributes, the lack of consideration of various attributes at the same time and robust performance has limited the applicability of DRs in spite of their quick computation time. Therefore, the development of automated algorithms for discovering DRs with simultaneous incorporation of important attributes is crucial for various manufacturing environments. Thus, extraction of implicit knowledge from given schedules enables smart factories to automate the development of dispatching rules while reflecting various attributes.

This dissertation proposes a new approach based on inductive learning for FJSPs with release times. The approach generates a dispatching rule from scheduling knowledge of why one job is dispatched ahead of another with consideration of various attributes. The proposed approach consists of three steps: finding good solutions of FJSPs using existing solution approaches; transforming them into data for learning, and extracting dispatching rules using an ensemble machine learning method.

4.2 Problem Description

The FJSP, an extension of the JSP, allows an operation to be processed by any machine from a given set of alternative machines. The basic assumptions used in this proposal are as follows:

- The processing time of an operation on a machine, the precedence constraint of operations, weight, release time, and due date of a job are known and constant.
- All machines are available for the entire period of scheduling, and there are no machine breakdowns. Pre-emption is not allowed.

In this section, an MILP model is introduced for the FJSP based on the work by Özgüven et al. (2010). This model is extended to consider the total weighted tardiness and release times. The following notation is used for the formulation of MILP:

4.2.1 Indices and Sets

i	jobs $(i, i' \in J)$
j	operations $(j, j' \in O)$
k	machines $(k \in M)$
J	the set of jobs
O	the set of operations
O_i	the ordered set of operations of job i $(O_i \subseteq O)$, where $O_{if_{(i)}}$ is the first and $O_{il_{(i)}}$ is the last element of O_i
M	the set of machines
M_j	the set of alternative machines on which operation j can be processed $(M_j \subseteq M)$

4.2.2 Parameters

t_{ijk}	the processing time of operation O_{ij} on machine k
r_i	the release time of job i
d_i	the due date of job i
w_i	the weight of job i
L	a large number

4.2.3 Decision Variables

X_{ijk}	If machine k is selected for operation O_{ij}, $X_{ijk} = 1$; otherwise $X_{ijk} = 0$.
$Y_{iji'j'k}$	If operation O_{ij} precedes operation $O_{i'j'}$ on machine k, $Y_{iji'j'k} = 1$; otherwise $Y_{iji'j'k} = 0$.
S_{ijk}	the start time of operation O_{ij} on machine k
C_{ijk}	the completion time of operation O_{ij} on machine k
C_i	the completion time of job i
T_i	the tardiness of job i ($T_i = Max(0,\ C_i - d_i)$)

4.2.4 Mathematical Formulation

The proposed mathematical model is as follows:

$$Min \quad F = \sum w_i T_i$$

$$s.t. \quad \sum_{k \in M_j} X_{ijk} = 1 \quad \forall i \in J,\ \forall j \in O_i \tag{6}$$

$$S_{ijk} + C_{ijk} \leq (X_{ijk}) \cdot L \quad \forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j \tag{7}$$

$$C_{ijk} \geq S_{ijk} + t_{ijk} - (1 - X_{ijk}) \cdot L \quad \forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j \tag{8}$$

$$S_{ijk} \geq C_{i'j'k} - (\mathrm{Y}_{iji'j'k}) \cdot L \quad \forall i < i',\ \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \tag{9}$$

$$S_{i'j'k} \geq C_{ijk} - (1 - \mathrm{Y}_{iji'j'k}) \cdot L \quad \forall i < i',\ \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'} \tag{10}$$

$$\sum_{k \in M_j} S_{ijk} \geq \sum_{k \in M_j} C_{i,j-1,k} \quad \forall i \in J,\ \forall j \in O_i - \{O_{if_{(i)}}\} \tag{11}$$

$$C_i \geq \sum_{k \in M_j} C_{iO_{il_{(i)}}k} \quad \forall i \in J \tag{12}$$

$$\sum_{k \in M_j} S_{iO_{if_{(i)}}k} \geq r_i \quad \forall i \in J \tag{13}$$

$$T_i \geq C_i - d_i \quad \forall i \in J \tag{14}$$

and

$$X_{ijk} \in \{0,1\} \quad \forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j$$

$$S_{ijk} \geq 0 \quad \forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j$$

$$C_{ijk} \geq 0 \quad \forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j$$

$$Y_{iji'j'k} \in \{0,1\} \quad \forall i < i',\ \forall j \in O_i, \forall j' \in O_{i'}, \forall k \in M_j \cap M_{j'}$$

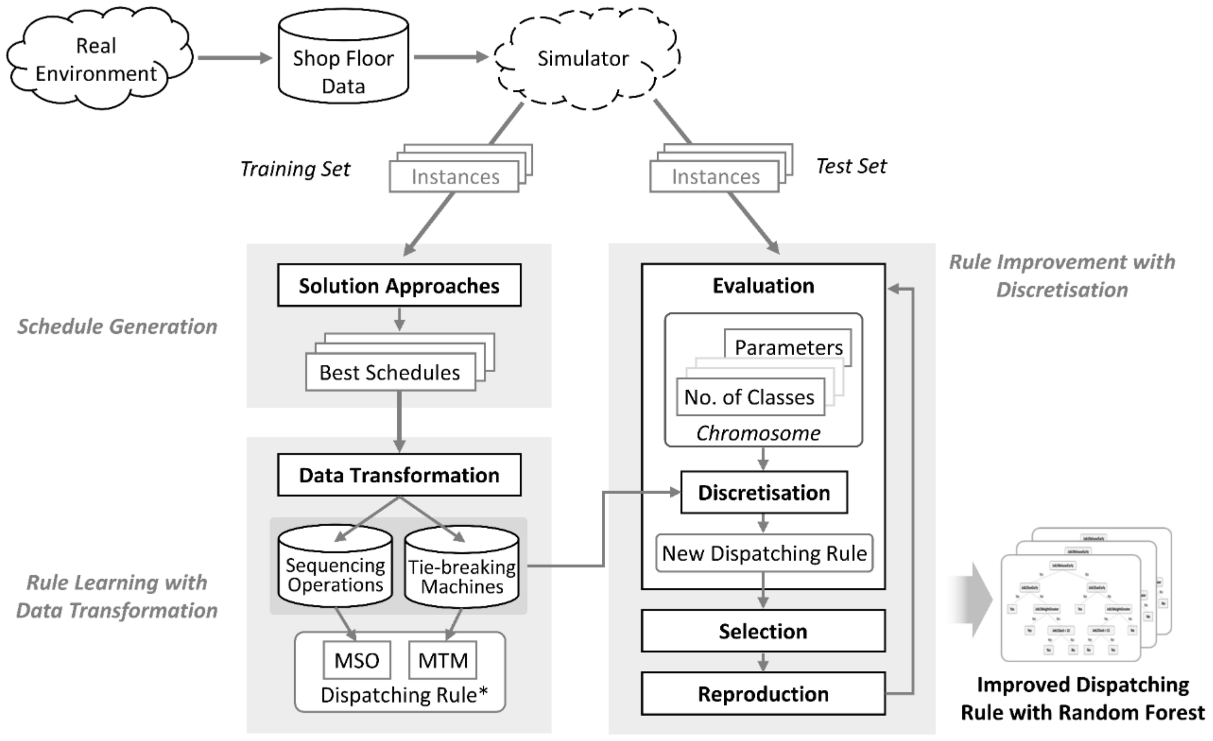
$$C_i \geq 0 \quad \forall i \in J$$

$$T_i \geq 0 \quad \forall i \in J$$

Constraint (6) guarantees that operation O_{ij} should be assigned to only one machine. Constraint (7) sets the start and completion times of operation O_{ij} on machine k to zero if it is not assigned to machine k. Otherwise, constraint (8) guarantees that the difference between the start and the completion times is at least equal to the processing time t_{ijk} on machine k. Constraints (9) and (10) ensure that operations O_{ij} and $O_{i'j'}$ cannot be processed at the same time on any machine in the set $M_j \cap M_{j'}$. Constraint (11) guarantees that the precedence constraints between the operations of a job are not violated. Constraint (12) ensures the completion times of the last operations of the jobs. Constraint (13) guarantees that jobs cannot start before their release times, and constraint (14) determines the tardiness of jobs.

4.3 Proposed Methodology

A new approach, called Random Forest for Obtaining Rules for Scheduling (RANFORS), consists of three phases: schedule generation, rule learning with data transformation, and rule improvement with discretization. The proposed approach can be summarized as follows (see Figure 4.1):



* MSO and MTM means a model for sequencing operations and tie-breaking machines, respectively.

Figure 4.1 Overall RANFORS framework

In the schedule generation phase, best solutions for given problem instances are generated as sources of learning rules for scheduling. In the second phase, once the solutions have been obtained, they are transformed into learning data by constructing new attributes. In this research, the term ‘attributes’ refers to the set of all data related to the scheduling decisions. Finally, in the rule improvement phase, a genetic algorithm improves the performance of the dispatching rule by discretizing continuous attributes and tuning the parameters of random forest models.

4.3.1 Schedule Generation

The major purpose of this phase is to find best solutions as a training set for learning good dispatching rules. In this context, three approaches widely used in the literature and industrial problems were implemented. The performances of the approaches are also compared in Chapter 4.4.

4.3.1.1 Mixed-integer Linear Programming

The MILP model proposed in Chapter 4.2.4 was coded in the IBM ILOG CPLEX, which solves the model by the branch-and-bound method, a traditional optimization technique that is widely used to solve MILP that can obtain the optimal solution.

4.3.1.2 Constraint Programming

As another commonly used approach to solve scheduling problems, CP has proven its applicability to a variety of scheduling problems, especially flexible job shops (Zhou 1996; Öztürk et al. 2013; Na and Park 2014). This CP model is based on high-level decision variables and constraints supported by IBM ILOG CP Optimizer for a concise code. A CP representation of the defined problem is as shown below.

4.3.1.2.1 Decision Variables

- $Itv^{O_{ij}}$: *interval* variable for operation O_{ij} encapsulating a start time, end time, and processing time.
- $Itv_{opt}^{O_{ij},k}$: *optional interval* variable for operation O_{ij} on machine k.

- Itv^{r_i} : *interval* variable for release time r_i

4.3.1.2.2 Objective Function

- *Minimize* $\sum_i w_i \times max(Endof\left(Itv^{O_{il_{(i)}}}\right) - d_i, 0)$

4.3.1.2.3 Constraints

- *Alternative*($Itv^{O_{ij}}, Itv_{opt}^{O_{ij},k}$) $\forall i \in J,\ \forall j \in O_i,\ \forall k \in M_j$
- *EndBeforeStart*($Itv^{O_{ij-1}}, Itv^{O_{ij}}$) $\forall i \in J,\ \forall j \in O_i - \{O_{if_{(i)}}\}$
- *EndBeforeStart*($Itv^{r_i}, Itv^{O_{if_{(i)}}}$) $\forall i \in J$

4.3.1.3 Hybrid Genetic algorithm

In this research, a hybrid genetic algorithm (HGA) with parallel variable neighborhood search (VNS) execution was implemented based on the work by Türkyılmaz and Bulkan (2015). The HGA parameters are based on the literature, and are summarized as shown in Table 4.1.

Table 4.1 Parameters of HGA with parallel VNS

Parameters	Value
Population size	500
Number of iterations	200
Number of no improvements	50
Crossover probability	0.7
Mutation probability	0.03
VNS neighborhood size	20
VNS parallel thread count	8

4.3.2 Rule Learning with Data Transformation

4.3.2.1 Data Transformation

Scheduling decisions concerning the job to prefer in job sequencing may depend not on the separate attributes but the summation, difference, or multiplication of two or more attributes (Wnek and Michalski 1994). For example, various well-known dispatching rules, such as SPT

(Shortest Processing Time) and EDD (Earliest Due Date), are based on a comparison of such attribute values as processing times and due dates when determining the next job to process. Thus, the construction of new attributes by using comparison operators and arithmetic operators can explain why one job is dispatched ahead of another.

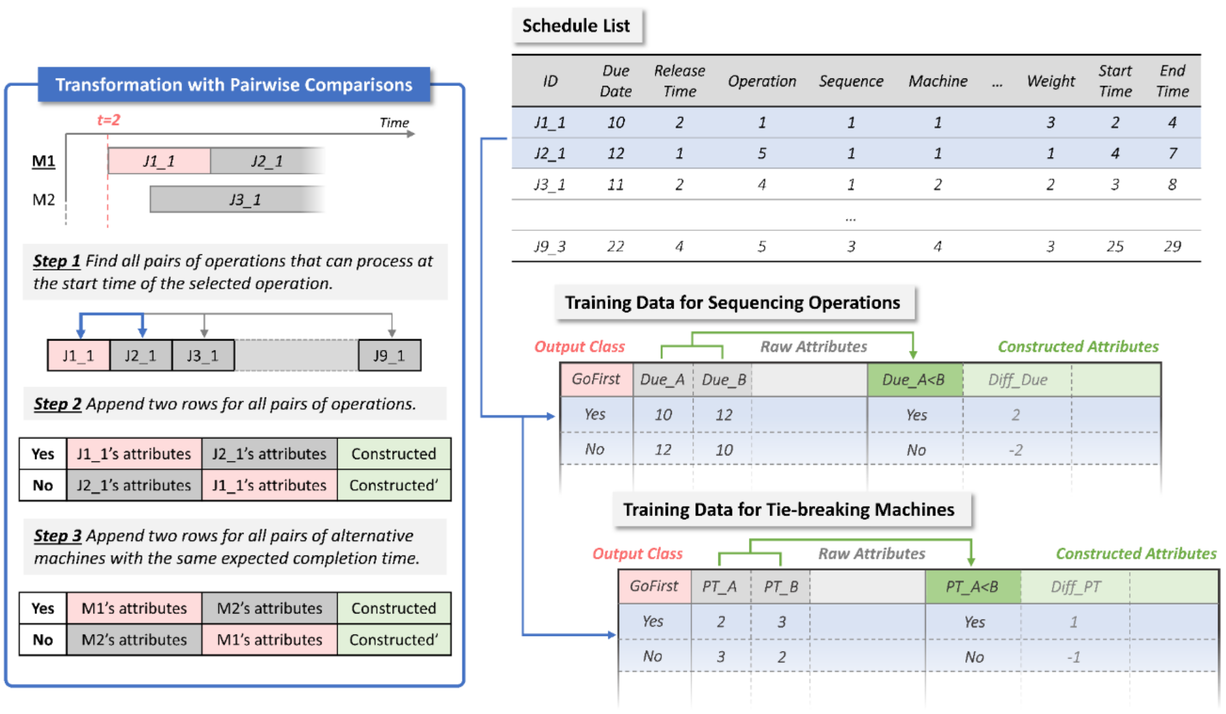


Figure 4.2 Data generation for learning from a flat data file of schedules

In the previous phase, the best solutions for each problem instance are saved as a flat data file, in which the columns represent separate data attributes and each row of the file represents a schedule of an operation. Based on a schedule list and its problem instance, a training dataset for sequencing operations and tie-breaking machines is generated by following three steps, as shown in Figure 4.2.

First, the first operation in the schedule list is selected and all operations that can be processed at the start time of that operation are listed. Then, rows for all possible pairs of operations are appended to a dataset for sequencing operations by comparing the respective operations. Finally, rows for all possible pairs of alternative machines for the selected operation are appended to the training dataset for tie-breaking machines if there are more than two machines with the same expected completion time. The expected completion time of machine k is the sum of the completion time of the last operation assigned to machine k and the processing time of the next operation in it. After completing these steps, the next operation in the schedule list is selected, and the three steps are repeated until the end of the list.

When adding rows to two datasets for sequencing operations and tie-breaking machines, in addition to raw attributes of two operations or machines, other attributes based on those raw attributes are constructed and appended as new columns. The detailed raw and constructed attributes are summarized in Table 4.2. These attributes can be categorized by source and type.

Table 4.2 Raw and constructed attributes

Source	Attribute Type (Number of attributes)	Attribute Description
Sequencing operations	Continuous raw attributes (10)	Due date of O1, release time of O1, operation sequence of O1, end time of O1's precedent operation, weight of O1, due date of O2, release time of O2, operation sequence of O2, end time of O2's precedent operation, weight of O2
	Boolean constructed attributes (7)	Whether O1's due date is earlier than O2, whether O1's release time is earlier than O2, whether the end time of O1's precedent operation is earlier than O2, whether O1's slack is shorter than O2, whether O1's remaining time is shorter than O2, whether O1's next processing time is shorter than O2, whether O1's weight is greater than O2
	Continuous constructed attributes (7)	Slack of O1, remaining processing time of O1, next processing time of O1, slack of O2, remaining processing time of O2, next processing time of O2, difference between due dates, difference between release times, difference between end times of precedent operations, difference between slack times, difference between remaining processing times, difference between next processing times, difference between weights
Tie-breaking machines	Continuous raw attributes (10)	End time of M1, processing time of M1, end time of M2, processing time of M2, end time of precedent operation of selected operation
	Boolean constructed attributes (2)	Whether M1's end time is earlier than M2, whether M1's processing time is shorter than M2
	Continuous constructed attributes (2)	Difference between end times, difference between processing times

4.3.2.2 Learning Dispatching Rules

Based on the two datasets for sequencing operations and tie-breaking machines, two models are generated by using machine learning algorithms, which models constitute a dispatching rule: a model for sequencing operations (MSO) and a model for tie-breaking between alternative machines (MTM). A flowchart of the detailed procedure for scheduling with the MSO and MTM is shown in Figure 4.3.

First, operations that can be processed at a particular point in time are sorted by the output classes of the MSO. For example, when an output class 'GoFirst' of the MSO is 'Yes', it means that Operation 1 tends to precede Operation 2; thus, Operation 1 earns one point. If there are four waiting operations to be processed, the next operation is determined by comparing all ${}_4P_2$ possible permutations. By summing all points for these permutations, the operation with the largest number of points is selected.

After selecting the next operation to be processed, one machine from all alternative machines is selected based on the expected completion time. Similarly to the process for sequencing operations, alternative machines for the selected operation are sorted by the total earned points by according to the output classes of the MTM when there are two or more machines with the same expected completion time. Based on the assignment of all operations in jobs to alternative machines, a detailed schedule and its performance are determined.

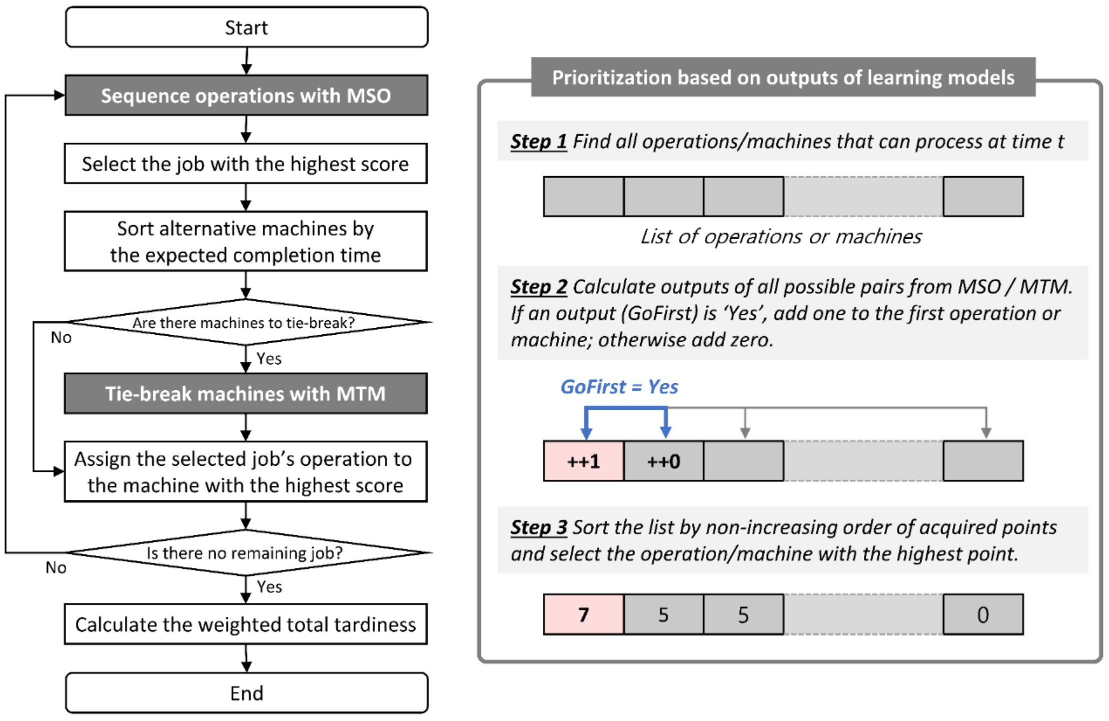

Start
Sequence operations with MSO
Select the job with the highest score
Sort alternative machines by the expected completion time
Are there machines to tie-break?
No
Yes
Tie-break machines with MTM
Assign the selected job's operation to the machine with the highest score
Is there no remaining job?
No
Yes
Calculate the weighted total tardiness
End
Prioritization based on outputs of learning models
Step 1 Find all operations/machines that can process at time t
List of operations or machines
Step 2 Calculate outputs of all possible pairs from MSO / MTM. If an output (GoFirst) is 'Yes', add one to the first operation or machine; otherwise add zero.
GoFirst = Yes
++1
++0
Step 3 Sort the list by non-increasing order of acquired points and select the operation/machine with the highest point.
7
5
5
0

Figure 4.3 Flowchart of dispatching rule based on an MSO and MTM

Through these procedures, a dispatching rule based on the MSO and MTM generates the detailed schedule as well as the total weighted tardiness. For generating the MSO and MTM, three approaches (C4.5, random forest without discretization, and RANFORS) were implemented and compared their performances (see Chapter 4.4.2).

Figure 4.4 illustrates a decision tree of an MSO generated by the C4.5 algorithm; its maximum height is 5. The decision tree can also be expressed by a set of IF-THEN rules. In order to deal with continuous attributes such as due dates, a binary split with the maximum gain ratio for each attribute is performed. For example, in the case of the difference between due dates, a binary split at 25.5 results in the best gain ratio for given data. When the output class 'GoFirst' of the MSO is 'Yes', it means that Operation 1 tends to precede Operation 2, according to the best solutions from the schedule generation phase.

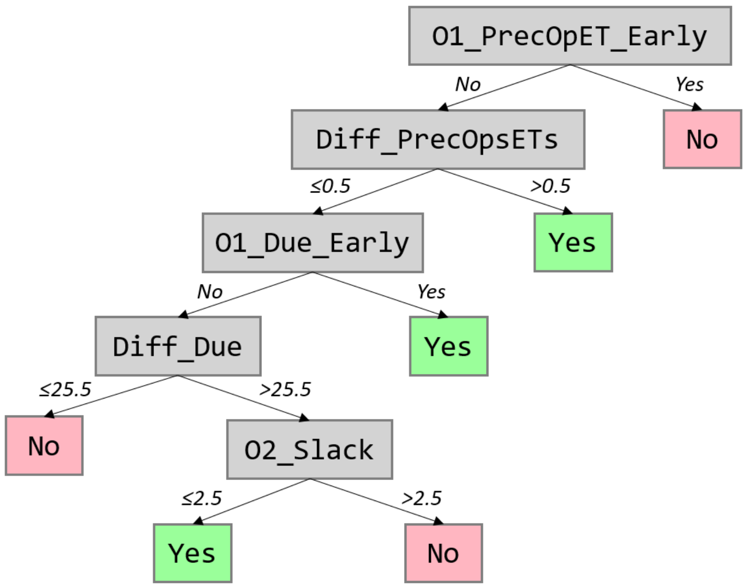


Figure 4.4 Illustration of decision tree of MSO generated by C4.5

4.3.3 Rule Improvement with Discretization

In spite of the power of interpretability, decision-tree-based algorithms have some drawbacks such as over-fitting and binary splits. For example, as shown in Figure 4.4, decision trees use binary splits, which are less suitable for continuous attributes. Multi-interval discretization methods are known to produce more accurate decision trees than binary discretization (Perner and Trautzsch 1998). Thus, to address these drawbacks, a new approach utilizes an evolutionary process to find the best multi-point splits and to tune parameters.

RANFORS applies the random forests among ensemble methods, which combine multiple ML models to create more powerful models. A random forest consists of decision trees where each tree is different from the others. Based on the group of decision trees, an output is delivered by majority voting where each decision tree contributes a single vote and a random forest chooses the class having the most votes among the decision trees (Hall, Bowyer et al. 2000). For example, if more than 50 percent of decision trees produce 'Yes' for the output class 'GoFirst', the final output of the random forest is determined as 'Yes'.

To improve the average total weighted tardiness, a genetic algorithm is applied for searching of the best set of discretization strategies and parameters for a random forest. Each chromosome represents a specific number of classes for continuous attributes and parameters in an MSO and

MTM, as shown in Figure 4.5. These chromosomes act as individuals in a population, and their improvement occurs after the repeated application of reproduction, evaluation, and selection.

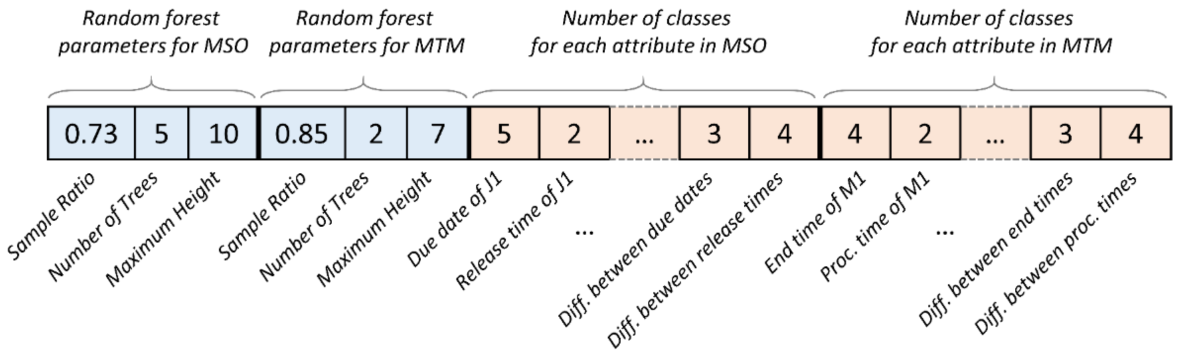


Figure 4.5 Chromosome representation

4.3.3.1 Discretization for Learning

In the discretization process, the proposed approach uses the equal frequency method, which partitions the domain of the continuous attribute so that the sample frequency in each interval is approximately the same (Chmielewski and Grzymala-Busse 1996). Based on integer values in a chromosome, all continuous attributes in the training data are discretized. If the number of classes is 2, it uses the binary discretization of C4.5; otherwise, it discretizes continuous values in an attribute into a given number of classes.

As shown in Figure 4.6, if the value of an attribute (the difference between due dates) is located between boundaries B1 and B2, it is discretized as class C2. The interval boundaries such as B1 and B2 are determined so that each interval contains approximately the same number of instances. By discretizing all values in the continuous attribute into three classes (for example, C1 for low, C2 for medium, and C3 for high), decision trees in a random forest can have multi-point splits instead of a binary split. A new dispatching rule for the chromosome is then derived from an MSO and MTM, which are updated according to the discretized attributes and the parameters of the chromosome.

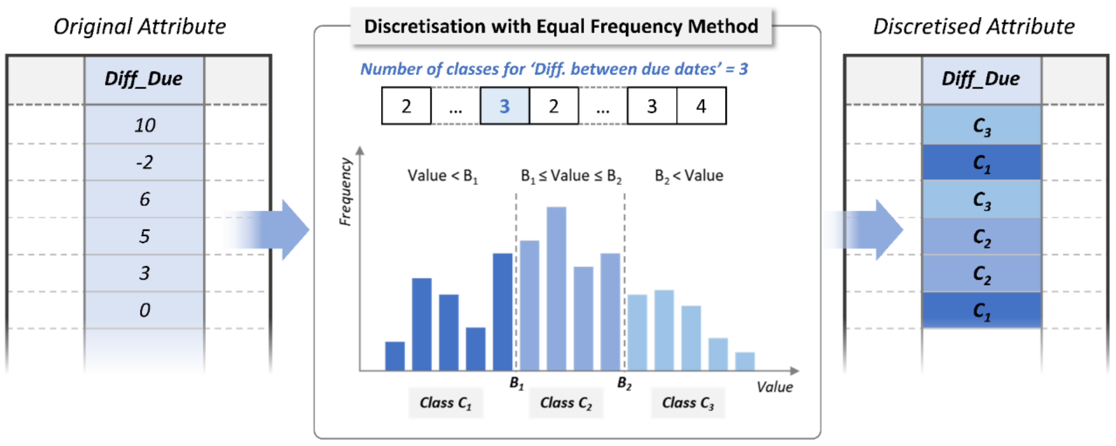


Figure 4.6 Discretization with equal frequency method

4.3.3.2 Evaluation

To evaluate the average performance of new dispatching rules, a set of problem instances for testing is generated with the same parameters of the training set at the beginning of every evaluation. Hence, problem instances in a testing set share parameters with the training set but they are different from those of the training set used in Chapter 4.3.2. The continuous attributes that were discretized for multi-point splits in a chromosome are categorized into the corresponding classes when sequencing operations and tie-breaking machines for scheduling with the updated MSO and MTM. After generating schedules with the dispatching rules for the testing set, the average total weighted tardiness is calculated.

4.3.3.3 Selection

To preserve the desirable characteristics of chromosomes for the next generation, the selection of surviving chromosomes is determined by the tournament selection method, and a fitness function represents the average total weighted tardiness.

4.3.3.4 Reproduction

To avoid local minima by preventing chromosomes from becoming too similar, mutation and crossover operators in the reproduction process are applied as shown in Figure 4.7. A mutation operator selects a position randomly in a chromosome and changes its value to a random number between predefined lower and upper bounds. In the case of crossover, a two-point crossover operator that changes only the selected values between the two points is applied.

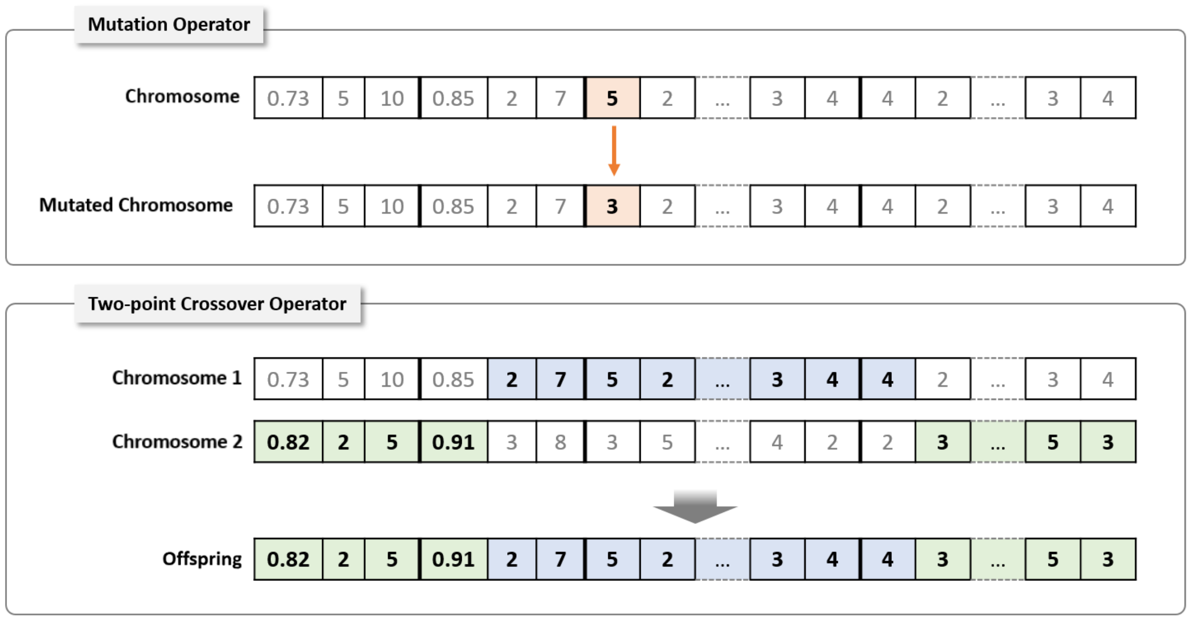


Figure 4.7 Mutation and crossover operators for reproduction

4.4 Experimental Results

In simulation experiments, two types of FJSP as training sets are considered: FJSP_10 (with 10 jobs, five machines, and five operations) and FJSP_30 (with 30 jobs, 15 machines, and 10 operations). Problem instances were randomly generated by using parameters from the method proposed by Brandimarte (1993). The due date of each job was specified by a date tightness parameter as in Tay and Ho (2008). The equation for generating due dates based on release times is

$$d_i = r_i + c \times \sum_{j=1}^{n_i} \bar{p}_{ij} \tag{15}$$

where $\bar{p}_{ij}$ is the average processing time of operation j of job i among the set of alternative machines M_j, c is the tightness factor of the due date, n_i is the number of operations of job i, and the release time of job i (r_i) is generated by $U\left[0, \sum_{j=1}^{n_i} \bar{p}_{ij}/n_i\right]$.

All of the algorithms were coded in C#, and the experiments were run on an Intel Xeon E5-1620 3.6 GHz processor with 16 GB of RAM. The detailed experimental parameters used to generate the problem instances are listed in Table 4.3.

Table 4.3 Parameters for two training sets of problem instances: FJSP_10 and FJSP_30

Parameter	Value	
	FJSP_10	FJSP_30
Number of jobs	10	30
Minimum and maximum numbers of operations per job	2–4	2–15
Number of machines	5	15
Maximum number of equivalent machines per operation	4	15
Minimum and maximum processing times per operation	3–10	
Tightness factor of due date	1	1.2
Number of problem instances	30	

4.4.1 Comparison between Solution Approaches in Schedule Generation

To obtain the best solutions for learning, simulation experiments on 30 problem instances for each training set were performed and the average performance of the solution approaches described in Chapter 4.3 were compared. To solve the CP and MILP, IBM ILOG CP and CPLEX optimizer version 12.7.1 with default settings were used. The run time was limited to 3,600 seconds for the MILP and CP in order to compare their performances. The solutions obtained by the three approaches are compared in Tables 4.4 and 4.5.

In summary, the comparison of the solution approaches for the two types of FJSPs indicated that MILP could find only a few optimal solutions for the small problem (FJSP_10), and could not find the optimal solution for the large problem (FJSP_30) under the same CPU time limit. The results also show that CP outperformed MILP and HGA in terms of the average total weighted tardiness.

Although CP and MILP produce better performances than does HGA, they encounter some obstacles for complex problems in the real world. First, MILP merely finds feasible solutions for a large problem within a reasonable time. In the case of CP, continuous decision variables should be approximated to integer variables by scaling up, because CP supports only integer decision variables (Bożek and Werner 2017). Thus, heuristic approaches such as HGA are more applicable for solving complicated problems efficiently. As shown in Figure 12, two sets of problem instances were used as a training set, and the best solutions for FJSP_10 and FJSP_30 were transformed into the learning data for the next phase.

Table 4.4 Experimental Results of the total weighted tardiness for two training sets

Instance Number	FJSP_10			FJSP_30		
	MILP	CP	HGA	MILP	CP	HGA
1	106	**95**	165	238	**134**	442
2	121	**120**	128	324	**154**	705
3	98	**91**	117	186	**132**	395
4	42	**36**	46	158	**19**	239
5	8	**7**	15	587	**339**	1048
6	42	**41**	59	469	**261**	609
7	92	**82**	92	32	**18**	210
8	71	**66**	94	522	**317**	883
9	**80**	**80**	125	312	**171**	497
10	73	**72**	95	170	**49**	345
11	58	**58**	70	500	**114**	438
12	30	**30**	56	321	**99**	477
13	**67***	**67**	99	319	**77**	415
14	68	**57**	80	195	**42**	492
15	**39**	**39**	46	256	**100**	424
16	34	**33**	52	330	**242**	558
17	30	**28**	35	316	**65**	396
18	**74**	**74**	108	435	**71**	316
19	**150**	**150**	185	515	**381**	840
20	**30**	31	41	229	**0**	244
21	64	**61**	82	353	**235**	756
22	95	**88**	140	383	**107**	551
23	80	**72**	122	498	**221**	854
24	**8**	**8**	14	945	**446**	979
25	**3***	**3**	7	168	**51**	259
26	**45***	**45**	66	368	**225**	701
27	94	**93**	110	102	**6**	441
28	148	**129**	194	321	**186**	449
29	44	**37**	77	580	**390**	909
30	**31**	**31**	56	217	**167**	414

Note: The optimal solutions are noted by an asterisk (*) and best feasible solutions are highlighted in bold.

Table 4.5 Result summary for solution approaches with training sets

	FJSP_10			FJSP_30		
	MILP	CP	HGA	MILP	CP	HGA
Average $\sum w_i T_i$	64.16	60.8	85.86	344.96	160.63	542.87
Number of optimal solutions	3	0	0	0	0	0
Average computation time (s)	3,600	3,600	204.9	3,600	3,600	1314.03

4.4.2 Comparison between Learning Algorithms

In this section, the applicability and average performance of RANFORS were tested by using the learning data transformed from the best solutions for each training set. To compare the performances of the different inductive learning techniques, two decision tree algorithms, C4.5 and Random Forest without a rule improvement phase, are also compared. Several pilot tests were run to choose the best parameters for C4.5 and Random Forest. To improve the generated dispatching rule through RANFORS, the computation times for the rule improvement phase were 6.15 and 104.31 hours for FJSP_10 and FJSP_30, respectively. The detailed parameters of C4.5, random forest, and RANFORS are shown in Table 4.6.

Table 4.6 Parameters of learning algorithms and RANFORS

Algorithm	Parameter	Value
C4.5	Maximum height of trees	10
Random Forest	Maximum height of trees	10
	Number of trees	3
	Sample ratio	0.8
RANFORS	Minimum and maximum height of trees	[5, 10]
	Minimum and maximum number of trees	[1, 5]
	Minimum and maximum sample ratio	[0.5, 1.0]
	Minimum and maximum number of classes	[2, 5]
	Population	500
	Number of generations	100
	Mutation rate	0.5
	Crossover rate	0.2
	Tournament size	5
	Number of problem instances for evaluation	60

The results for each algorithm are summarized in Table 4.7. They show that the accuracy of the MSO in RANFORS was higher than in the other algorithms, whereas there was no significant difference among the computation times for FJSP_10. On the contrary, in terms of FJSP_30, there were no significant differences in accuracy, and even the computation times of the algorithms based on random forest slightly increased. In the case of the average total weighted tardiness, the result shows that RANFORS can find better dispatching rules with discretization, because the average total weighted tardiness of the training sets is smaller than the others.

One possible explanation for this result is that the number of rows in FJSP_30 was considerably larger, approximately 16 times larger in fact, compared with FJSP_10. An example of generated decision trees in a random forest for sequencing operations that were improved by RANFORS for FJSP_30 is shown in Figure 4.8. The decision tree contains discretized attributes highlighted in yellow, which have multiple branches determined by the rule improvement phase of RANFORS.

Table 4.7 Summary of results of learning algorithms and RANFORS

		FJSP_10		FJSP_30	
		MSO	MTM	MSO	MTM
Computation time for training (s)	C4.5	0.6	**0.1**	**8.3**	0.4
	Random Forest	0.6	0.3	16.6	1.3
	RANFORS	**0.4**	**0.1**	13.2	**0.2**
Accuracy (%)	C4.5	91.65	87.21	92.63	88.26
	Random Forest	93.21	**89.4**	92.56	**88.52**
	RANFORS	**99.09**	87.53	**92.58**	88.43
Average $\sum w_i T_i$	C4.5	263.83		2071.36	
	Random Forest	208.3		1855.7	
	RANFORS	**200.56**		**1654.03**	

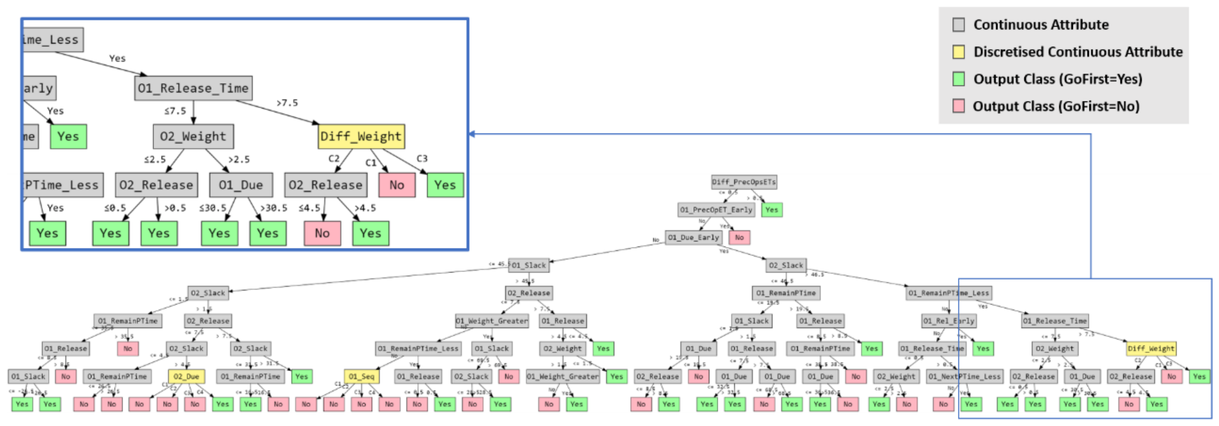


Figure 4.8 Generated decision tree in MSO of RANFORS for FJSP_30

4.4.3 Comparison among Algorithms for Similar Types of Problem Instances

In order to check the average performance for similar types of problem instances with the training data, three variations of problem instances for each size with different due-date-tightness levels were generated: 0.8 (tight), 1.0 (moderate), 1.2 (loose). Also, to validate the competitive performance of a dispatching rule leant by RANFORS, eight well-known dispatching rules that consider due dates or release dates for sequencing operations were implemented: EDD, MOD, MDD, Slack, ATC, COVERT, AT, and PT+WINQ+SL (Jeong and Kim 1998, Rajabinasab and Mansour 2011). The machine allocation for these dispatching rules was based on the earliest completion time with random tie-breaking.

In addition to the average total weighted tardinesses, the relative deviation index (RDI) was applied to compare the relative performances of the dispatching rules with HGA. The RDI of the kth experiment was calculated as shown in Equation (5) (Akhshabi, Tavakkoli-Moghaddam, and Rahnamay-Roodposhti 2014). The average total weighted tardiness, computation times, and RDIs of the dispatching rules and HGA over 30 problem instances for each due-date-tightness level are compared in Tables 4.9-10 and Figure 4.9.

Among the dispatching rules, RANFORS outperformed the others in terms of the average total weighted tardiness for FJSP_10 and FJSP_30, which were similar to the training sets. In terms of relative performance, the results showed that, as the size of a problem increased, the overall relative performance of RANFORS became worse, but RANFORS still outperformed the other dispatching rules in that it yielded the smallest average RDI value for every case.

4.4.4 Comparison among Algorithms for Different Types of Problem Instances

In order to check the robustness of performance under more various conditions, two sets of larger problem instances were generated: FJSP_50 and FJSP_100. In addition, two types of experiments for FJSP_50 and FJSP_100 were designed: with different due-date-tightness levels and different flexibilities (Nie et al. 2013). For the larger problems, a dispatching rule learned from the learning data of FJSP_30 is used. The parameters used in the different types of problem instances are summarized in Table 4.8.

Table 4.8 Parameters for four test sets with different types of problem instances

Parameter	Value	
	FJSP_50	FJSP_100
Number of jobs	50	100
Minimum and maximum numbers of operations per job	2–15	
Number of machines	15	
Maximum number of equivalent machines per operation (Flexibility; f)	8 (f=50%), 11 (f=70%)	
Minimum and maximum processing times per operation	2–15	
Tightness factor of due date (c)	0.8, 1.0, 1.2	
Number of problem instances	30	

4.4.4.1 Larger Number of Jobs with Different Due-date-tightness Levels

In the case of FJSP_50 with different due-date-tightness levels, the dispatching rule of RANFORS still produced the lowest average total weighted tardiness, as shown in Table 4.11. By contrast, for the largest problems with 100 jobs, it could not always obtain the best performance in cases where the problem types deviate from the training sets (FJSP_100 with c=1.0 and c=1.2) as shown in Table 4.12 and Figure 4.10. Thus, the results showed that RANFORS can offer good performance for the larger number of jobs with different due-date-tightness levels compared with the others until the types of problem instances are too divergent from the training set.

The results also indicated that, although HGA always finds the best schedules compared to the dispatching rules, the gap of the average computation time between HGA and the dispatching rules continued to broaden when the size of the problem increased. Thus, the dispatching rules, with their quicker computation speed, are more applicable in terms of scalability for extremely large problems in the real world. In addition, the development of efficient dispatching rules significantly improves the performance of heuristics when using them as an initial population of heuristics.

4.4.4.2 Larger Number of Jobs with Different Flexibilities

In the case of FJSP_50 and FJSP_100 with different flexibilities, the dispatching rule of RANFORS produced the lowest average total weighted tardiness, as shown in Tables 4.13 and 4.14. In terms of relative performance, the results showed that RANFORS still outperformed the other dispatching rules, in that it yielded the smallest average RDI value for every case, as shown in Figure 4.11.

In summary, the simulation results demonstrated that the proposed RANFORS has an advantage over the other dispatching rules in terms of finding robust and good solutions for larger problem sizes, especially for problems that are similar to the training sets. In addition, RANFORS appears to find more generalized dispatching rules with discretization, because the average total weighted tardiness and RDI under different scenarios are smaller than the other algorithms without discretization.

Table 4.9 Average performances of dispatching rules over three test sets of FJSP_10 with different due-date-tightness levels

	FJSP_10								
	$c = 0.8$			$c = 1.0$			$c = 1.2$		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	168.63	205.17	0	87.57	200.93	0	56.83	211.06	0
C4.5	332.17	<1	33.27	271.8	<1	31.61	220.27	<1	29.80
Random Forest	301.73	<1	26.73	213.27	<1	21.36	160.27	<1	19.17
RANFORS	**290.4**	<1	**24.64**	**207.1**	<1	**20.29**	**138.73**	<1	**14.57**
EDD	447.83	<1	53.29	382.27	<1	48.66	367.07	<1	53.25
MOD	396.97	<1	43.39	307.1	<1	36.49	235.0	<1	30.85
Slack	408.63	<1	46.27	324.2	<1	41.07	246.9	<1	35.05
ATC	651.03	<1	89.71	603.23	<1	88.64	597.87	<1	92.74
COVERT	635.77	<1	85.94	556.23	<1	76.57	507.53	<1	75.48
MDD	465.17	<1	58.19	417.07	<1	54.07	368.83	<1	53.38
AT	459.9	<1	57.38	424.03	<1	53.99	384.93	<1	58.55
PT+WINQ+SL	423.37	<1	49.52	421.17	<1	55.04	368.83	<1	53.39

Note: Bold values indicate the best performances among the dispatching rules.

Table 4.10 Average performances of dispatching rules over three test sets of FJSP_30 with different due-date-tightness levels

	FJSP_10								
	$c = 0.8$			$c = 1.0$			$c = 1.2$		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	1245.17	1351.47	0	900.8	1413.87	0	569.23	1344.37	0
C4.5	2806.93	<1	43.04	2710.6	<1	54.64	2400.83	<1	48.54
Random Forest	2435.7	<1	33.03	2276	<1	40.49	2092.07	<1	40.86
RANFORS	**2254.93**	<1	**28.28**	**2069.7**	<1	**34.07**	**1807.8**	<1	**33.34**
EDD	2913.7	<1	45.38	2627.6	<1	51.20	2180.9	<1	44.21
MOD	2911.4	<1	45.26	2462.13	<1	46.40	1921.63	<1	35.96
Slack	2942.73	<1	47.64	2369.17	<1	43.78	1848.3	<1	34.81
ATC	4090.57	<1	77.78	3745.27	<1	83.58	3738.2	<1	83.56
COVERT	4269.17	<1	78.68	3441.9	<1	75.53	3195.4	<1	69.63
MDD	3377.17	<1	57.02	2925.53	<1	60.03	2471.67	<1	51.77
AT	4327.57	<1	83.76	3929.37	<1	86.75	3505.57	<1	79.01
PT+WINQ+SL	2909.8	<1	45.40	2862.9	<1	56.67	2165.67	<1	43.23

Note: Bold values indicate the best performances among the dispatching rules.

Table 4.11 Average performances of dispatching rules over three test sets of FJSP_50 with different due-date-tightness levels

	FJSP_50								
	$c = 0.8$			$c = 1.0$			$c = 1.2$		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	4506.63	2465.43	0	3920.13	2386.47	0	3127.63	2443.97	0
C4.5	9385.67	<1	53.32	8559.07	<1	44.21	8169.73	<1	51.94
Random Forest	8608.9	<1	45.02	7786.7	<1	36.64	7149.0	<1	40.70
RANFORS	**7391.47**	<1	**31.02**	**6994.33**	<1	**29.12**	**6192.57**	<1	**31.30**
EDD	8423.57	<1	42.47	8067.9	<1	38.48	7754.5	<1	47.80
MOD	9204.87	<1	51.99	8512.6	<1	43.02	7303.67	<1	43.33
Slack	8301.2	<1	42.48	7926.4	<1	35.22	6634.33	<1	36.83
ATC	11646.33	<1	74.85	12370.9	<1	75.08	11528.67	<1	83.60
COVERT	11303.97	<1	71.53	10608.9	<1	62.00	10097.5	<1	71.71
MDD	9726.53	<1	56.48	9733.57	<1	54.75	8767.07	<1	57.69
AT	12665.03	<1	86.56	13638.7	<1	87.92	11711.8	<1	85.44
PT+WINQ+SL	9449.57	<1	54.34	9199.9	<1	48.31	7783.2	<1	46.30

Note: Bold values indicate the best performances among the dispatching rules.

Table 4.12 Average performances of dispatching rules over three test sets of FJSP_100 with different due-date-tightness levels

	FJSP_100								
	$c = 0.8$			$c = 1.0$			$c = 1.2$		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	21771.8	3613	0	20609.37	3613.67	0	18479.73	3611.37	0
C4.5	37979.7	<1	45.62	42372.27	<1	57.95	37843.23	<1	57.08
Random Forest	34927.43	<1	37.28	35890.73	<1	40.41	33700.5	<1	44.38
RANFORS	**32821.13**	<1	**30.38**	33883.63	<1	35.24	30763.57	<1	34.40
EDD	35225.3	<1	37.22	**33528.9**	<1	**34.61**	31029.9	<1	36.37
MOD	37938.43	<1	45.60	37063.47	<1	44.27	32772.47	<1	41.97
Slack	38561.2	<1	47.59	33856.37	<1	35.18	**27546.83**	<1	**25.39**
ATC	46153.7	<1	69.49	43716.5	<1	62.42	43760.8	<1	67.31
COVERT	43200.77	<1	60.52	40515	<1	53.55	39974.6	<1	56.97
MDD	41187.63	<1	55.85	41488.93	<1	56.11	37477.57	<1	55.67
AT	55712.37	<1	91.71	55974.4	<1	90.34	52696.17	<1	92.61
PT+WINQ+SL	45046.2	<1	62.86	42185.57	<1	55.70	35319.73	<1	46.83

Note: Bold values indicate the best performances among the dispatching rules

Table 4.13 Average performances of dispatching rules over three test sets of FJSP_50 with different flexibilities

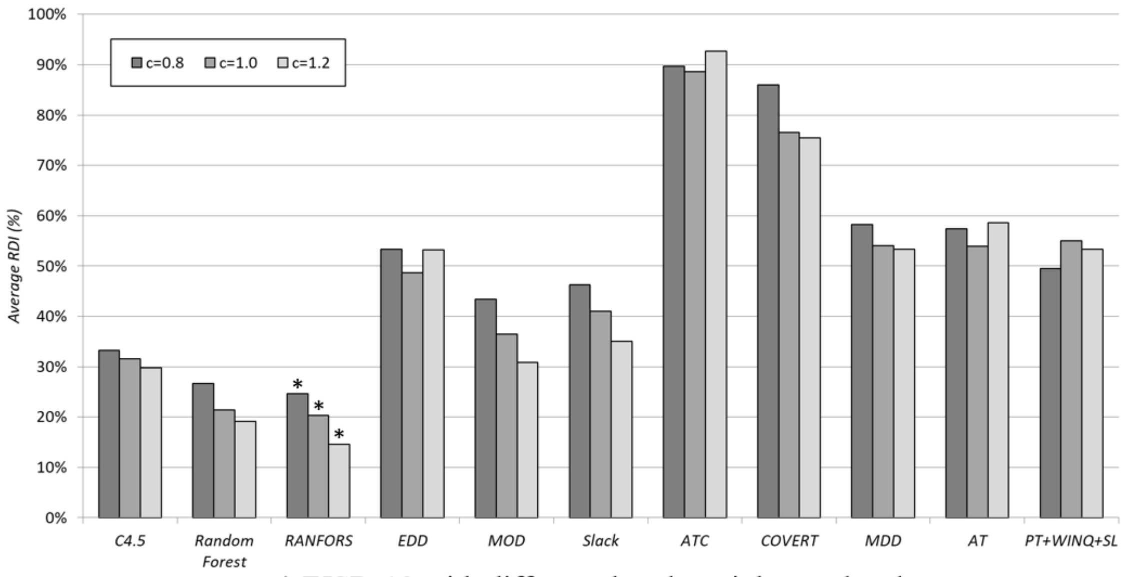
	FJSP_50					
	f = 50% (c = 0.8)			f = 70% (c = 0.8)		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	3871.83	2757.33	0	3868.47	2839.03	0
C4.5	8173.53	<1	48.66	7384.8	<1	38.96
Random Forest	7020.97	<1	35.68	5923.1	<1	22.11
RANFORS	**6349.8**	<1	**27.70**	**5413.1**	<1	**16.58**
EDD	8103.4	<1	48.85	7732.93	<1	42.50
MOD	8284.43	<1	50.68	8554.13	<1	51.07
Slack	7031.8	<1	35.65	6247.53	<1	25.98
ATC	11412.83	<1	83.68	12189.9	<1	90.99
COVERT	10433.53	<1	72.69	11082.53	<1	78.48
MDD	9364.3	<1	63.45	9659.37	<1	63.57
AT	11071.63	<1	80.38	10035.67	<1	66.74
PT+WINQ+SL	8708.5	<1	53.99	8464.27	<1	49.59

Note: Bold values indicate the best performances among the dispatching rules.

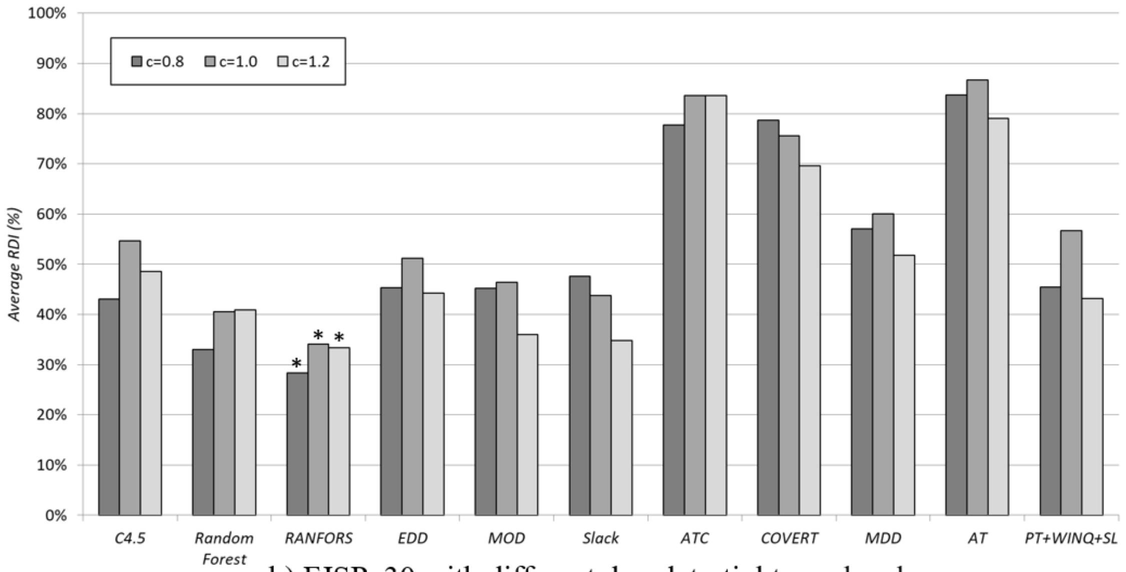
Table 4.14 Average performances of dispatching rules over three test sets of FJSP_100 with different flexibilities

	FJSP_100					
	f = 50% (c = 0.8)			f = 70% (c = 0.8)		
	$\sum w_i T_i$	Computation Time (s)	RDI (%)	$\sum w_i T_i$	Computation Time (s)	RDI (%)
HGA	20096.13	3614.2	0	19416.13	3615.8	0
C4.5	35573.97	<1	45.57	31914.57	<1	40.53
Random Forest	30151.4	<1	28.31	27433.83	<1	25.94
RANFORS	**28496.57**	<1	**23.79**	**24447.17**	<1	**16.33**
EDD	31458.57	<1	33.59	31127.4	<1	37.22
MOD	35579.53	<1	45.56	35053.37	<1	50.13
Slack	30186.87	<1	29.18	26053.2	<1	21.23
ATC	45783.2	<1	74.90	46277.27	<1	85.12
COVERT	44755.13	<1	69.47	42977.43	<1	74.87
MDD	39610.77	<1	57.92	37093.63	<1	55.34
AT	50548	<1	83.96	44697.67	<1	76.55
PT+WINQ+SL	41310.67	<1	59.15	39884.8	<1	61.74

Note: Bold values indicate the best performances among the dispatching rules.

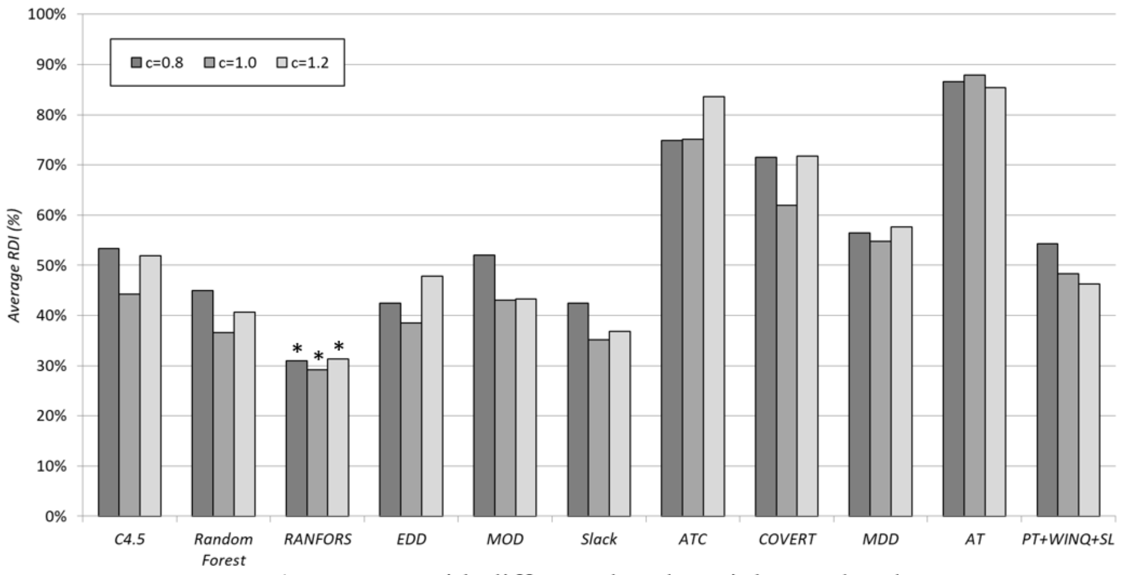


a) FJSP_10 with different due-date-tightness levels

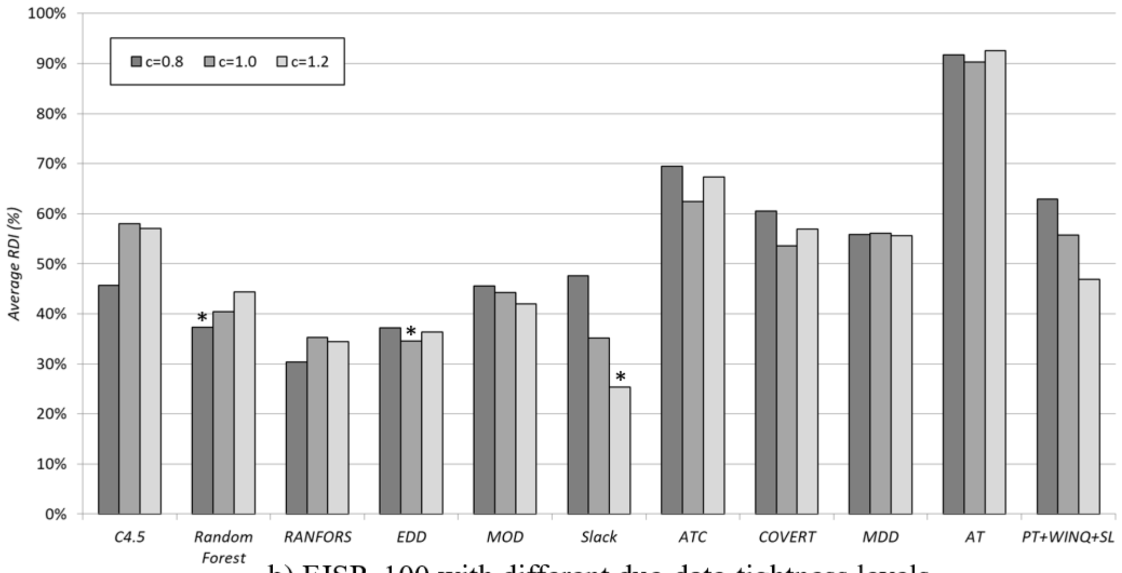


b) FJSP_30 with different due-date-tightness levels

Figure 4.9 Average RDIs of dispatching rules for three test sets of FJSP_10 and FJSP_30 with different due-date-tightness levels. The best RDIs for each test set are noted by an asterisk (*).

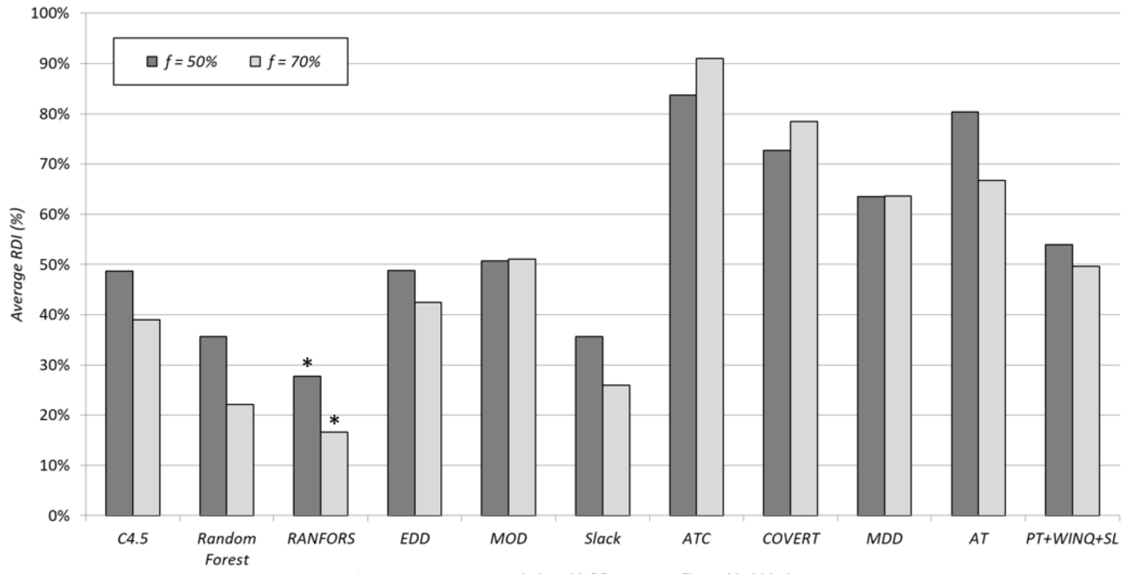


a) FJSP_50 with different due-date-tightness levels

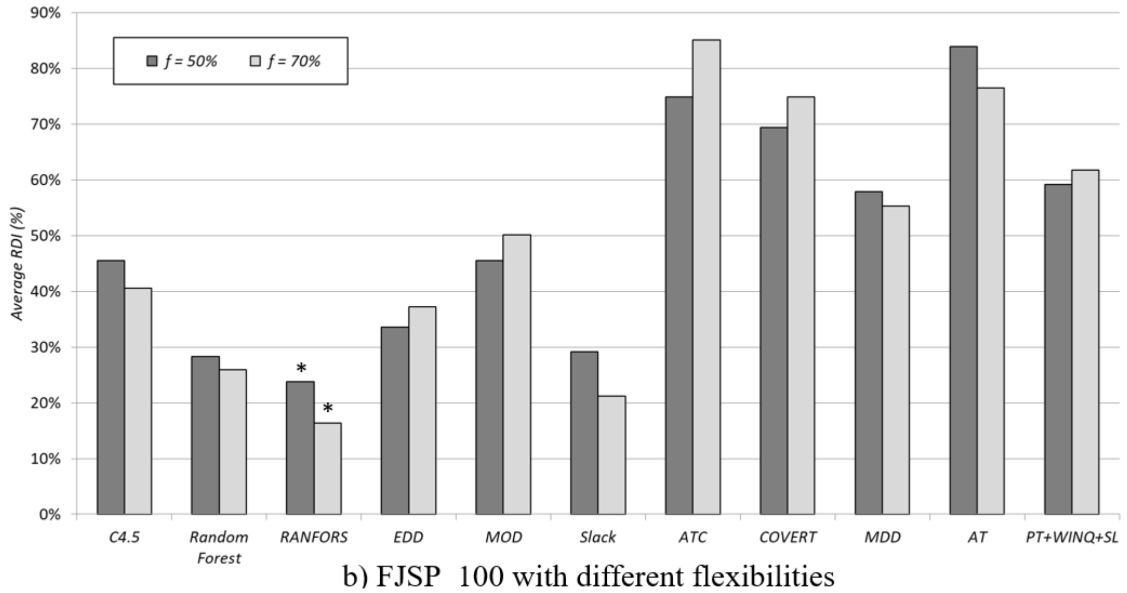


b) FJSP_100 with different due-date-tightness levels

Figure 4.10 Average RDIs of dispatching rules for three test sets of FJSP_50 and FJSP_100 with different due-date-tightness levels. The best RDIs for each test set are noted by an asterisk (*).



a) FJSP_50 with different flexibilities



b) FJSP_100 with different flexibilities

Figure 4.11 Average RDIs of dispatching rules for three test sets of FJSP_50 and FJSP_100 with different flexibilities. The best RDIs for each test set are noted by an asterisk (*).

4.5 Chapter Summary

In this chapter, the FJSP with release times to minimize the total weighted tardiness is analyzed. This study proposes an approach called RANFORS that consists of three phases: schedule generation, rule learning with data transformation, and rule improvement with discretization. By analyzing scheduling data from the best solutions, dispatching rules were extracted as two random forest models that can be expressed in a set of decision trees or IF-THEN rules. To extract dispatching rules more effectively, a GA is designed to find parameters and discretize the continuous attributes while improving the performance and the generalizability of the learned results. The results of simulations showed that the new dispatching rule generated by RANFORS outperformed prevalent dispatching rules in terms of average performance and robustness, as it produced a smaller average total weighted tardiness and RDI values.

The major contribution of this chapter is the development of a new approach for capturing both explicit and implicit knowledge from scheduling data. Furthermore, RANFORS can extract dispatching rules from both past production data and the best solutions through simulations. The results of this study indicate that RANFORS can be used to develop dispatching rules with less human intervention through the discovery of underlying logics that might not be realized by the schedulers themselves. Moreover, schedulers can modify the dispatching rules generated by RANFORS, because those rules are represented in such understandable formats as decision trees or rule sets.

CHAPTER 5. LOGISTICS WITH AUTONOMOUS ROBOTS

5.1 Introduction

As shown in Chapter 2.4, considerable work relating to PF, VR, and CR has been done. However, consideration of all three sub-problems and their interconnections has been relatively neglected. Even though some approaches with task allocation, such as MAPD, have been researched, assumptions including single-load and simple task allocations may not be appropriate to deal with more complicated problems involving multi-load AMRs in practice. Also, in the case of VR, lack of consideration of conflicts between paths can cause unexpected delays due to deadlocks and collisions and thus can affect system performance significantly, especially when using many AMRs in a limited space. Finally, due to the inherent complexities of sub-problems, the centralized approach is not suitable for solving all three sub-problems within a reasonable time when scaling up.

Thus, a comprehensive framework for scheduling and routing of AMRs is absolute necessary in order to tackle computational complexity and inherent interdependencies. The goal of this chapter is to address the optimization problems for minimizing total tardiness with consideration of the shortest paths and conflicts. The major contributions of this chapter can be summarized as follows:

- In order to solve the sub-problems more efficiently, we propose a comprehensive approach for PF, VR, and CR that solves them by connecting the outputs for each.
- The more generalized objective function, total tardiness, is considered for minimization of delays of transportation requests and thus optimization of material flow.
- A PF framework for discretizing a map into grids and finding all required paths is suggested to consider obstacles as well as recharging during operations.
- Regarding VR, a contextual-bandit-based algorithm is developed to optimize the selection process of local search operators for minimum human intervention.
- In the case of CR, an agent-based model with a set of behaviors for resolving conflicts is designed.

5.2 Problem Description

All required information for logistics problems with AMRs consists of three datasets: map, AMRs' status, and transportation requests. The map data includes work centers with loading/unloading stations and obstacles. Also, at the beginning of formulating a new operational plan, the current status for each AMR, such as includes location and remaining battery level, is updated. Finally, a set of transportation requests with two paired nodes (pickup and delivery) with due dates is given.

Based on these datasets, optimization problems can be disintegrated into the three sub-problems as shown in Figure 5.1. The objective function is to minimize the total tardiness, which is defined as the sum of all of tardiness values of transportation requests. The tardiness for each request is defined as *max*(completion time of delivery – due date, 0). The following sections elucidate the assumptions and characteristics for each sub-problem in this dissertation.

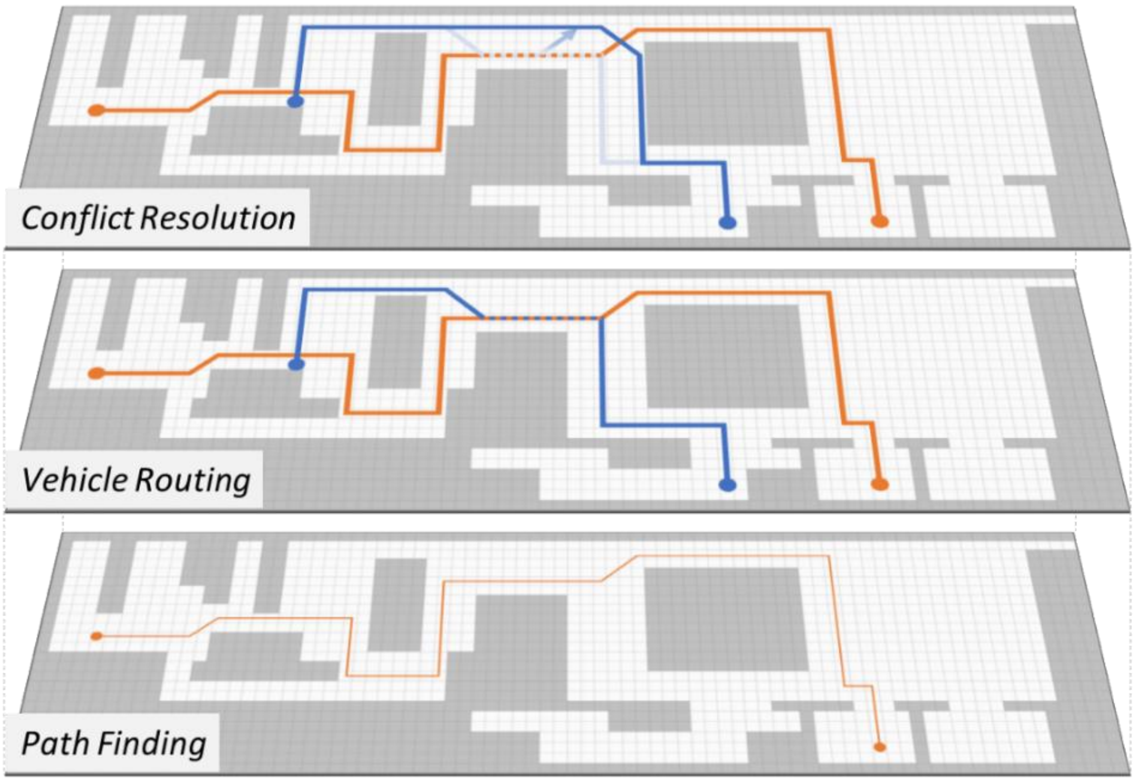
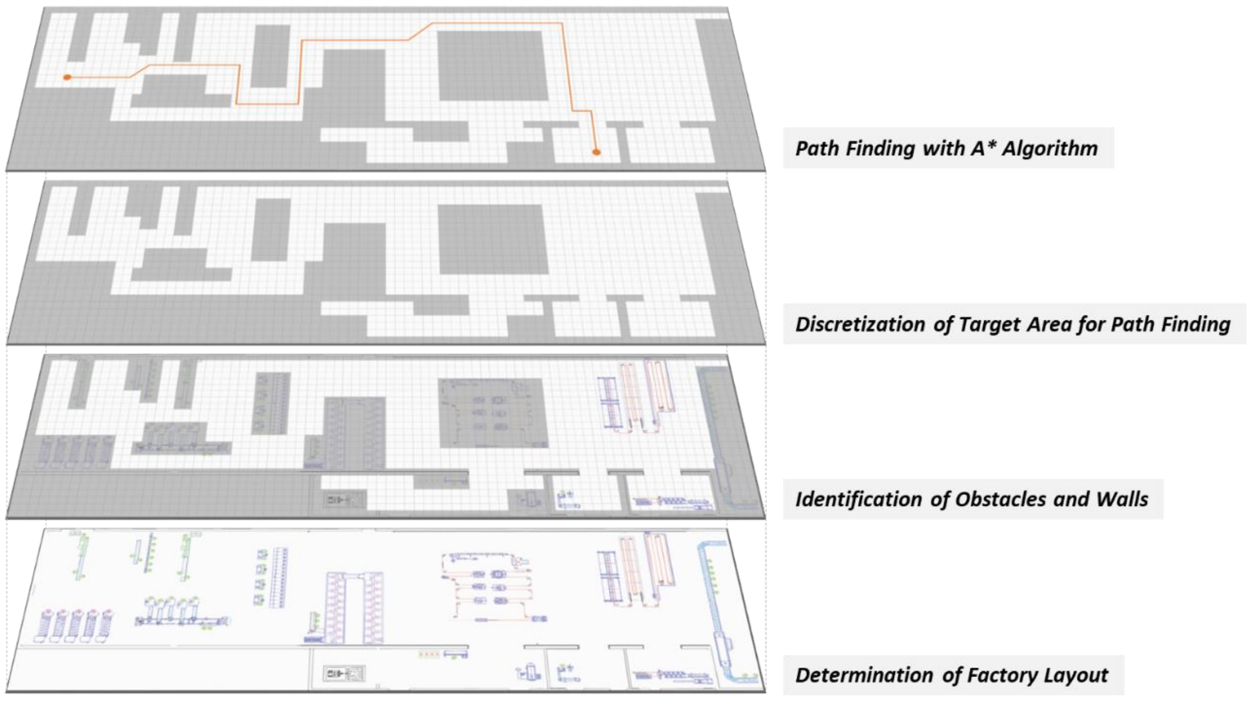


Figure 5.1 Overview of three sub-problems for scheduling and routing of AMRs

5.2.1 Path Finding (PF) with Map Discretization

The goal of PF is to identify the shortest paths considering obstacles between all possible locations including initial locations of all vehicles, loading/unloading (UL) stations, and recharging stations. In order to minimize computation time and guarantee the appropriate safety distances between AMRs, the grid-based approach that has been widely used in the previous literature is applied (Cirillo et al. 2014). A PF problem can be represented as a graph G, which contains a set of $\{n_i\}$ nodes and a set of arcs $\{e_{ij}\}$. For example, if an arc e_{pq} is an element of $\{e_{ij}\}$, it represents a directed line segment with the cost c_{pq} from node n_p to n_q. An optimal path from n_i to n_j is a path having the smallest cost over the set of all paths from n_i to n_j.

As shown in Figure 5.2, the map data is discretized with a given size of grids based on a layout of the shop floor. After discretizing the layout, the shortest paths between locations are pre-calculated and stored for the other sub-problems (VR and CR). However, when the paths should be updated because of changes (e.g. new obstacles, additional transportation requests, breakdowns), new paths with consideration of those changes are newly calculated by solving PF again.



Path Finding with A* Algorithm
Discretization of Target Area for Path Finding
Identification of Obstacles and Walls
Determination of Factory Layout

Figure 5.2 Illustration of grid-based procedures for path finding

5.2.2 Vehicle Routing (VR) for AMRs

In the case of VR for AMRs, the following assumptions are made in this dissertation:

- Each AMR has a maximum payload, initial and maximum travel times, and speed.
- Each transportation request having a due date and weight consists of two nodes: a node for picking up from one UL station and a node for delivery to another UL station.
- While an AMR is travelling, the battery level decreases linearly with the distance; therefore, the AMR might need to visit the closest charging station between the pickup and delivery locations. The battery can be charged to any level, and the charging time depends on the recharging rate and the amount to be charged.
- A required detour for charging is also considered; Figure 5.3 shows a comparison of paths with and without recharging. The brown areas represent obstacles such as pillars and other machines. In order to consider charging between operations, the required detours for charging are precalculated. The green paths in Figure 3 represent the detours between two nodes.

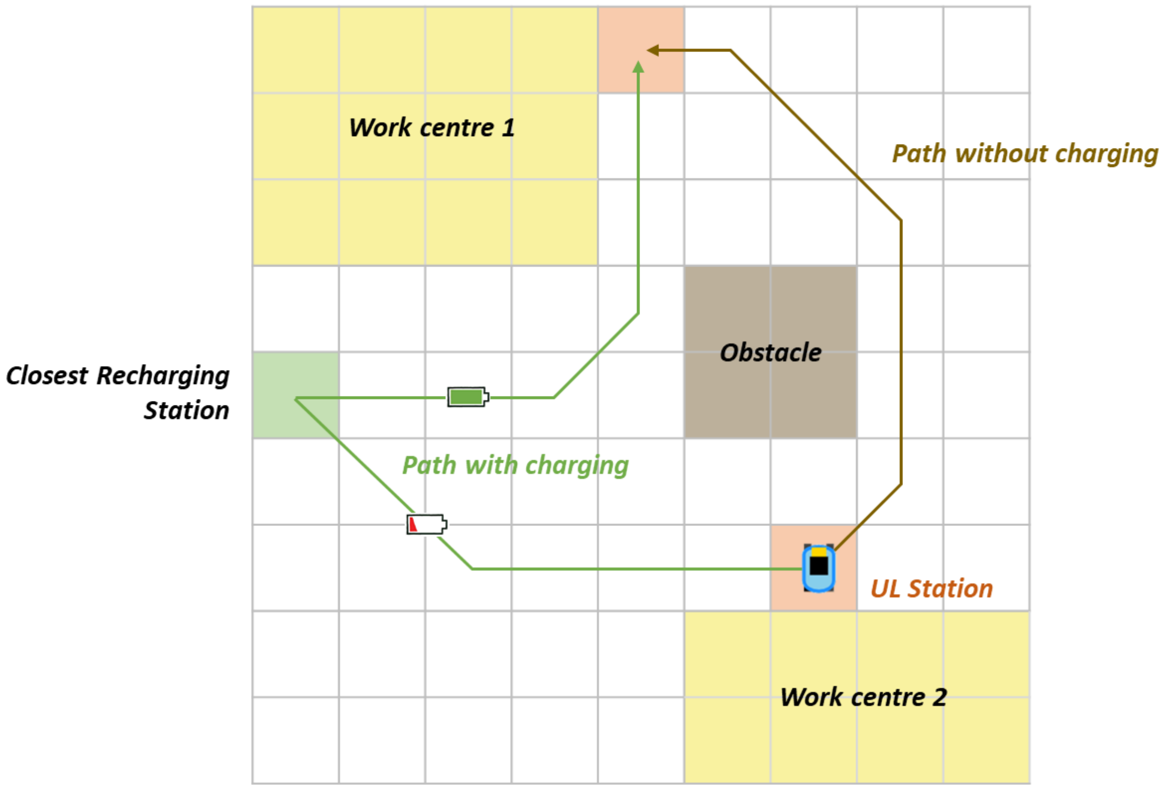
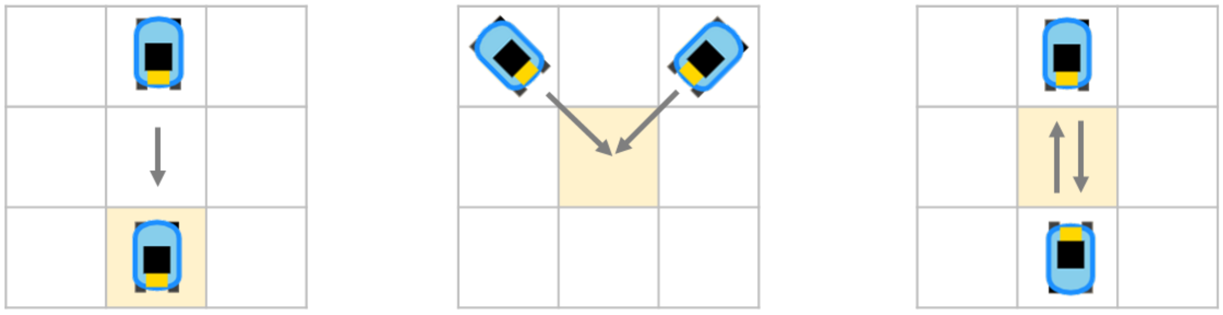


Figure 5.3 Illustration of paths with and without recharging in VR

5.2.3 Conflict Resolution (CR)



(a) Collision (rear) (b) Collision (side-on) (c) Deadlock

Figure 5.4 Types of conflicts between AMRs

As shown in Figure 5.4, conflicts between two AMRs in a grid space can be classified into two categories: collision and deadlock. Figure (a) and (b) represent collisions that occur when more than two AMRs are heading to the same node in different directions or when one AMR is going toward a node occupied by another AMR (Xue et al. 2017).

In order to resolve those conflicts, Sun et al. (2014) proposed a collision-avoidance-behavior named *WaitForGoThrough*. Under this behavior, when the AMR of higher priority is detected, the AMR of lower priority must wait until the grid is freed. However, none of the AMRs can advance and all AMRs will be permanently blocked unless corrective action is taken. This situation is characterized as a deadlock (c) and, in current practice, deadlocks can be resolved by re-routing (Reveliotis 2000).

5.3 Proposed Approach

In this section, a framework for scheduling of AMRs with consideration of PF, VR, and CR is proposed in streamlined procedures. In order to address all of the sub-problems, adaptive local search based on contextual bandits is incorporated into A* algorithm and ABM as shown in Figure 5.5. In the following sections, the pre-processing of map and path finding with A* are first described. Based on the shortest paths, in order to assign AMRs to transportation requests efficiently, an efficient local search technique for improving the initial solution are explained. Finally, a decentralized approach based on ABM is proposed to identify expected delays caused by conflict resolution.

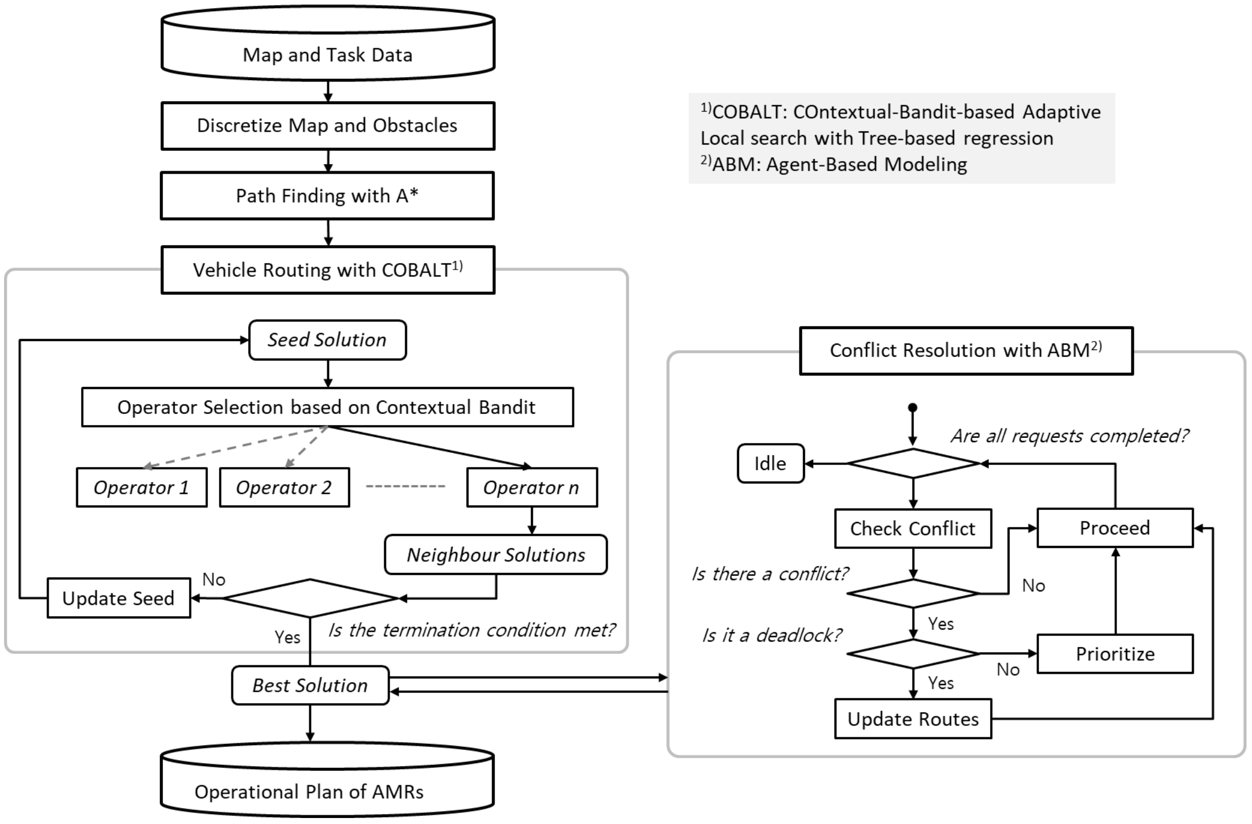

Map and Task Data
Discretize Map and Obstacles
Path Finding with A*
Vehicle Routing with COBALT1)
Seed Solution
Operator Selection based on Contextual Bandit
Operator 1
Operator 2
Operator n
Neighbour Solutions
Update Seed
No
Yes
Is the termination condition met?
Best Solution
Operational Plan of AMRs
1)COBALT: COntextual-Bandit-based Adaptive Local search with Tree-based regression
2)ABM: Agent-Based Modeling
Conflict Resolution with ABM2)
Are all requests completed?
Idle
Check Conflict
Proceed
Is there a conflict?
No
Yes
Is it a deadlock?
Prioritize
No
Yes
Update Routes

Figure 5.5 Overall framework of proposed approach

5.3.1 PF with A* Algorithm

A* algorithm, proposed by Hart (1968), is one of the best-known search algorithms for robotics, utilizing a heuristic function to accelerate the computation time. A* algorithm aims to find a path incurring the smallest cost over the set of all paths from a start node s to a goal node. In order to determine the next node n to extend at each iteration, A* selects the node that minimizes the evaluation function $f(n)$, which is calculated as follows:

$$f(n) = g(n) + h(n) \tag{16}$$

where $g(n)$ is the cost of the path from s to n with minimum cost (thus far found by A*), and $h(n)$ is any estimate of the cost of an optimal path from n to a preferred goal node of n. For the heuristic function $h(n)$, various types of distances such as Manhattan, Euclidean or Chebyshev are widely used. In this dissertation, Euclidean distance for $h(n)$ and found the shortest paths between all possible locations is used.

In addition to the standard A* algorithm, the Jump Point Search (JPS) method proposed by Harabor and Grastien (2011) was applied. By reducing the number of examined cells, JPS can accelerate the computation speed of A* algorithm. According to the previous literature (Duchoň et al. 2014), the experimental results of several scenarios for a mobile robot showed that JPS with A* algorithm was the best algorithm among all path finding algorithms in terms of computation speed.

5.3.2 VR with Contextual-bandit-based Adaptive Local Search

In this module, a dispatching rule for assigning AMRs to requests based on due dates is presented to generate a seed solution. In addition, a new local search algorithm called COBALT (COntextual-Bandit-based Adaptive Local search with Tree-based regression) is proposed whereby a seed solution is iteratively improved by choosing local search operators based on contextual bandits.

5.3.2.1 Generation of Seed Solution

First, a seed solution is generated by the dispatching rule based on due dates (Earliest Due Time; EDT) (Ho and Chien 2006). The EDT selects the transportation request having the earliest due date (*NextRequest*); then, the nearest vehicle to the pickup node of *NextRequest* is chosen. When recharging is required, AMRs need to visit the closest charging station, and they are charged to the full battery capacity for each recharging stop. The detailed procedures of the EDT are presented in Algorithm 3.

Algorithm 3: Earliest Due Time

```
1:  Obtain a list of requests, a list of vehicles
2:  do {
3:      Initialise NextRequest and NextVehicle
4:      NextRequest ← a request with the earliest due date in a list of unserved requests
5:      NextVehicle ← the nearest vehicle to the pickup node of NextRequest
6:      Serve pickup and delivery nodes with NextVehicle
7:      Update the total tardiness and NextVehicle's current location and completion time
8:  } while (unserved requests exist)
```

5.3.2.2 Contextual-bandit-based adaptive local search

Various local search techniques can be recognized for a decision problem in which a search operator must be chosen among candidate operators (Nareyek 2003). Local search operators for optimization problems can be classified into two categories: exploitation and exploration. Exploitation operators aim at converging quickly to a local optimum based on the seed solution, while exploration operators focus on increasing the diversity of the seed solution at the expense of the current local improvement. For balancing between exploitation and exploration, reinforcement-learning-based approaches have drawn considerable attention. Of the various reinforcement learning techniques, multi-armed bandit (MAB) is an emerging area of research and application, due specifically to its versatility. Goëffon et al. (2016) proposed a new model for simulating non-stationary operators in search algorithms that should alternate between exploitation and exploitation stages in their search processes.

In general, the performance of local search algorithms largely depends on the characteristics of the problem's search space and the designs of the search operators. Especially, learning from previous choices and improvements is necessary in order to develop an adaptive local search algorithm, because the context, which includes the seed solution and the problem instance, may have a huge impact on these two objectives. While much previous work relating to local search algorithms with MAB has been done, consideration of the context along with non-stationary rewards has been far less studied. Thus, to select the best operator with consideration of contexts, a new local search algorithm called COntextual-Bandit-based Adaptive Local search with Tree-based regression (COBALT) is developed.

5.3.2.2.1 Design of Local Search Operators

In the process of COBALT, a set of neighborhoods is generated repeatedly by choosing and applying a chosen local search operator on the current seed solution. To improve the initial solution produced by EDT, four operators based on the previous literature that are widely used in various local-search techniques for the PDP were implemented (Li and Lim 2003; Cherkesly et al. 2015; Li et al. 2016). These local search operators are designed to improve the expected tardiness without consideration of conflicts between routes. After finding the best neighborhood solution, the current seed solution is updated according to the best one, and the entire process is repeated up to the maximum computation time.

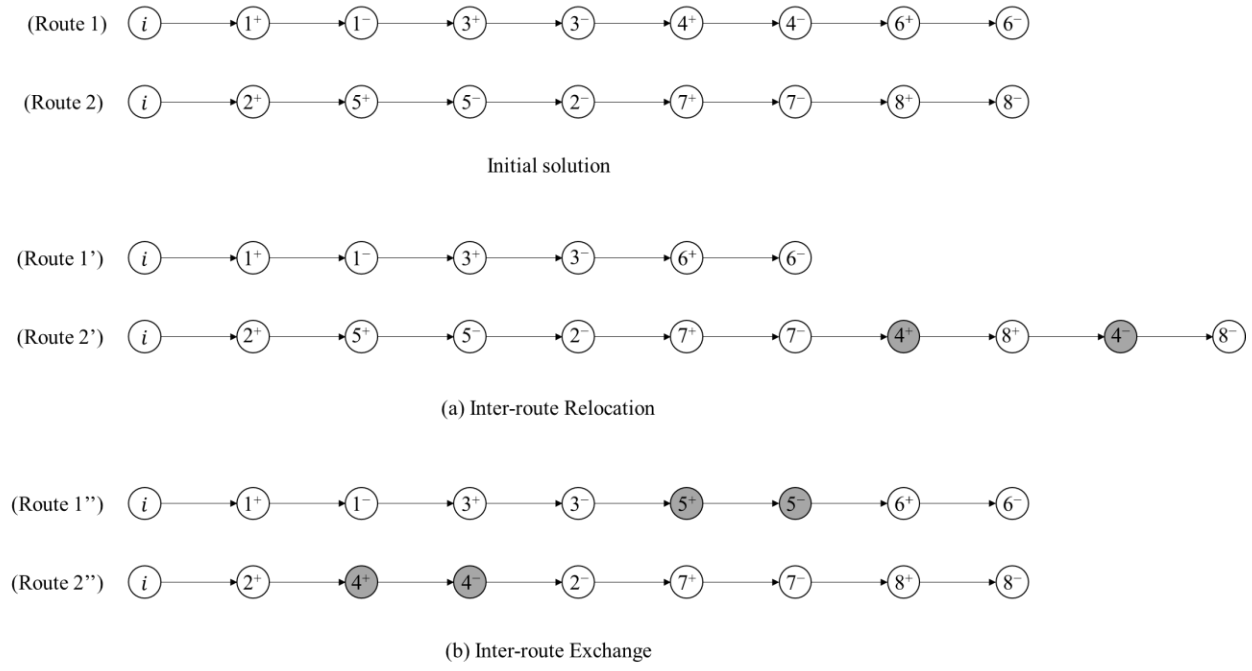

(Route 1) i 1+ 1− 3+ 3− 4+ 4− 6+ 6−
(Route 2) i 2+ 5+ 5− 2− 7+ 7− 8+ 8−
Initial solution
(Route 1') i 1+ 1− 3+ 3− 6+ 6−
(Route 2') i 2+ 5+ 5− 2− 7+ 7− 4+ 8+ 4− 8−
(a) Inter-route Relocation
(Route 1'') i 1+ 1− 3+ 3− 5+ 5− 6+ 6−
(Route 2'') i 2+ 4+ 4− 2− 7+ 7− 8+ 8−
(b) Inter-route Exchange

Figure 5.6 Inter-route relocation and exchange

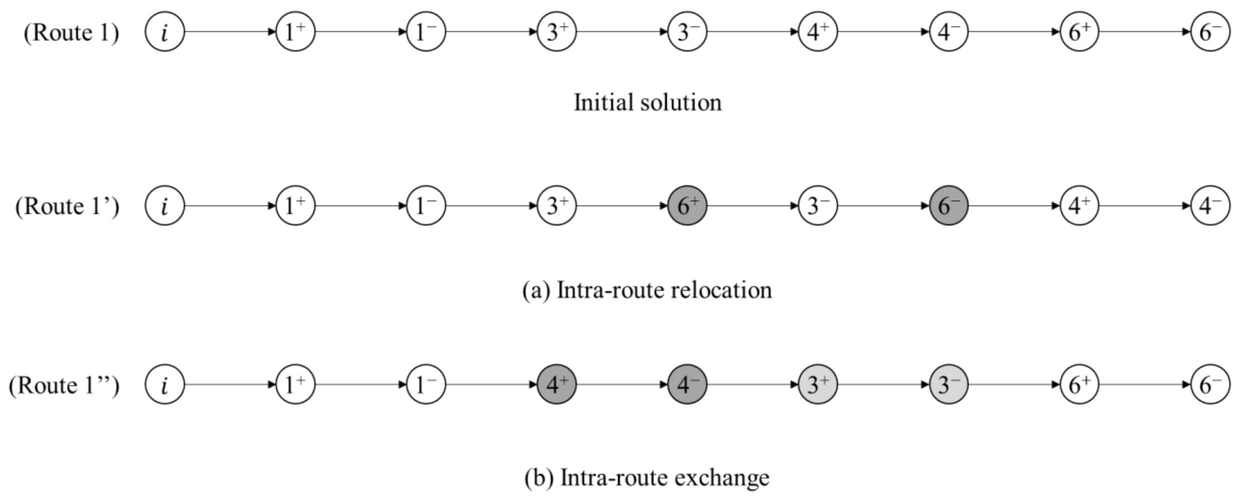


Figure 5.7 Intra-route relocation and exchange

Inter-route Relocation

In inter-route relocation, two AMRs and a pair of nodes from one AMR are randomly selected and the inter-route relocation returns the best solution among neighborhoods that are generated by inserting the removed nodes from an AMR to all possible insertion points to another AMR. Figure 5.6 (a) explains an inter-route relocation whereby the pickup and delivery nodes of request 4 are inserted into another AMR.

Inter-route Exchange

The inter-route exchange operator removes two pickup and delivery nodes from each AMR and exchanges their positions between AMRs. Figure 5.6 (b) illustrates an inter-route exchange whereby the pickup and delivery nodes of requests 4 and 5 are exchanged with each other.

Intra-route Relocation

The intra-route relocation first chooses an AMR and a transportation request randomly and inserts nodes for the transportation request into random positions. Figure 5.7 (a) depicts an intra-route relocation whereby a delivery node and a pickup node of request 6 are relocated. This operator is repeated 10 times to make neighborhoods and returns the best solution among them.

<u>*Intra-route Exchange*</u>

The locations of the two pickup and delivery nodes in a randomly chosen AMR are exchanged for the same vehicle. Figure 5.7 (b) shows an intra-route exchange whereby nodes of request 3 and nodes of request 4 are exchanged. The intra-route exchange operator selects a number of two transportation requests (set as 10) randomly, exchanges them, and returns the best solution among those changes.

5.3.2.2.2 Selection of Operators based on Contextual Bandit

In COBALT, the seed solution is continuously improved in every trial *t* by observing a current context, choosing a local search operator, and replacing the seed solution with a better solution of a new neighborhood of the chosen operator. Using an M-dimensional context vector $x_t \in \mathbb{R}^M$ as input, the algorithm chooses one of K possible local search operators at trial *t*. x_t contains various features of the current solution and factors that may affect the improvement of operators. In this dissertation, cumulative improvement, previous improvement, previous operator, current total tardiness, total distance, average distance, and number of requests were used. $a_t \in \{1, \dots, K\}$ denotes the chosen operator at trial *t* by the algorithm. Let F_0 and F_t and denote the total tardinesses of the seed solution at the beginning and trial *t*, respectively. After checking the total tardiness of the best neighborhood solution, the reward r_{t,a_t} associated with the chosen operator a_t and context x_t can be observed as shown below.

$$r_{t,a_t} = 1 - \frac{F_t}{F_0} - \sum_{t=1}^{t-1} r_{t,a_t} \tag{17}$$

The reward r_{t,a_t} represents the improvement by choosing a_t at trial *t* excluding the previous improvements. The entire selection process is repeated until a termination condition is met. In this dissertation, the maximum computation time (600 seconds) is used as the termination condition considering the required times for the other sub-problems (PF and CR).

To address the contextual bandit problem for choosing an operator, McNellis et al. (2017) proposed a practical method with bootstrapping and decision trees, which are non-parametric and interpretable. The detailed procedures are presented in Algorithm 4.

Algorithm 4: COBALT-C()

```
1:  do {
2:    Observe context vector $x_t$
3:    for $a$ = 1, …, $K$ do
4:      Sample bootstrapped dataset $\widetilde{D}_{t,a}$ from $D_{t,a}$
5:      Fit decision tree $\tilde{\theta}_{t,a}$ to $\widetilde{D}_{t,a}$ with CART algorithm
6:    End
7:    Choose action $a_t = arg \max_a \hat{p}(\tilde{\theta}_{t,a}, x_t)$
8:    Generate a set of neighborhood solutions based on $a_t$ and observe $r_{t,a_t}$
9:    Update $D_{t,a}$ with $(x_t, r_{t,a_t})$ and $t \leftarrow t + 1$
10: } while (the termination condition is not met)
```

In Algorithm 4, $D_{t,a}$ is the set of observations for action at trial t, and $\widetilde{D}_{t,a}$ denotes the dataset obtained by bootstrapping $D_{t,a}$. $\tilde{\theta}_{t,a}$ is a decision tree fit on a given dataset, and $\hat{p}(\tilde{\theta}_{t,a}, x_t)$ means the estimated probability of reward from using decision tree $\tilde{\theta}_{t,a}$ in context x_t. In addition to COBALT-C, an ensemble model of boosted regression trees, namely Multiple Additive Regression Trees (MART), is also considered to compare the performances with and without bootstrapping (COBALT-M). MART is widely used in practice due to its high prediction accuracy for various applications (Vinayak and Gilad-Bachrach, 2015). The main steps of local search with MART are summarized in Algorithm 5.

Algorithm 5: COBALT-M()

```
1:  do {
2:    Observe context vector $x_t$
3:    for $a$ = 1, …, $K$ do
4:      Fit decision tree $\tilde{\theta}_{t,a}$ to $D_{t,a}$ with MART algorithm
5:    end
6:    Choose action $a_t = arg \max_a \hat{p}(\tilde{\theta}_{t,a}, x_t)$
8:      Generate a set of neighborhood solutions based on $a_t$ and observe $r_{t,a_t}$
9:    Update $D_{t,a}$ with $(x_t, r_{t,a_t})$ and $t \leftarrow t + 1$
10: } while (the termination condition is not met)
```

5.3.3 CR with agent-based modeling

In order to check the actual tardiness considering conflicts and delays, a decentralized approach for resolving conflicts between routes with agent-based modeling (ABM) was implemented. The basic idea of agent-based simulation is the implementation of autonomous functional units that communicate using dedicated protocols and cooperate to solve complex tasks. In this dissertation, an agent refers to an autonomous robot with embedded behaviors for conflict resolution that can be expressed by a flow chart as shown in Figure 5.8. The protocols and states are based on the work of Zhao et al. (2015) and Draganjac et al. (2016).

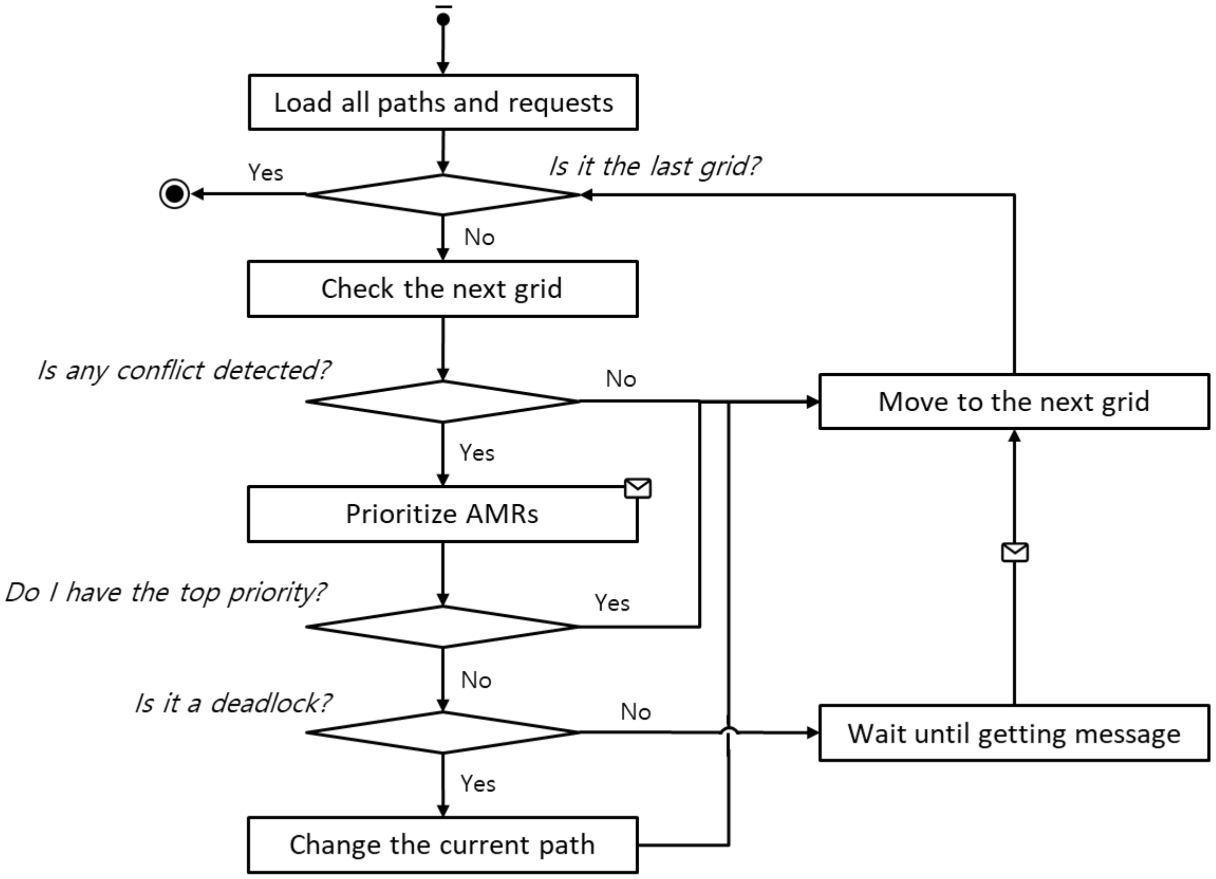


Figure 5.8 Summary of protocols and states between AMRs for CR

5.3.3.1 *Collision Resolution*

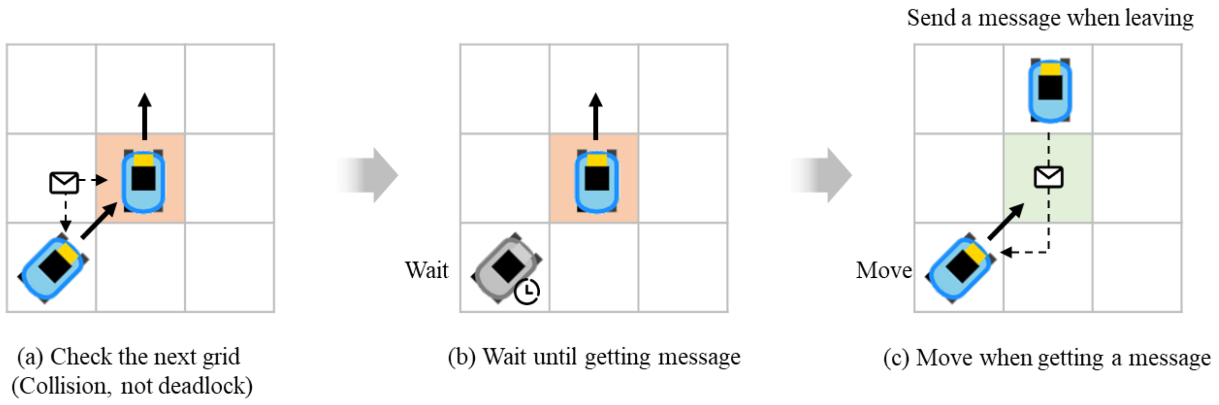

Send a message when leaving
Wait
Move
(a) Check the next grid
(Collision, not deadlock)
(b) Wait until getting message
(c) Move when getting a message

Figure 5.9 Collision avoidance with prioritizing and waiting

When a collision between AMRs is anticipated, it can be easily resolved by prioritizing AMRs and applying a simple wait-and-move strategy for the AMRs other than the one having the highest priority. Figure 5.8 represents the wait-and-move strategy with prioritization by exchange of messages.

5.3.3.2 *Deadlock Resolution*

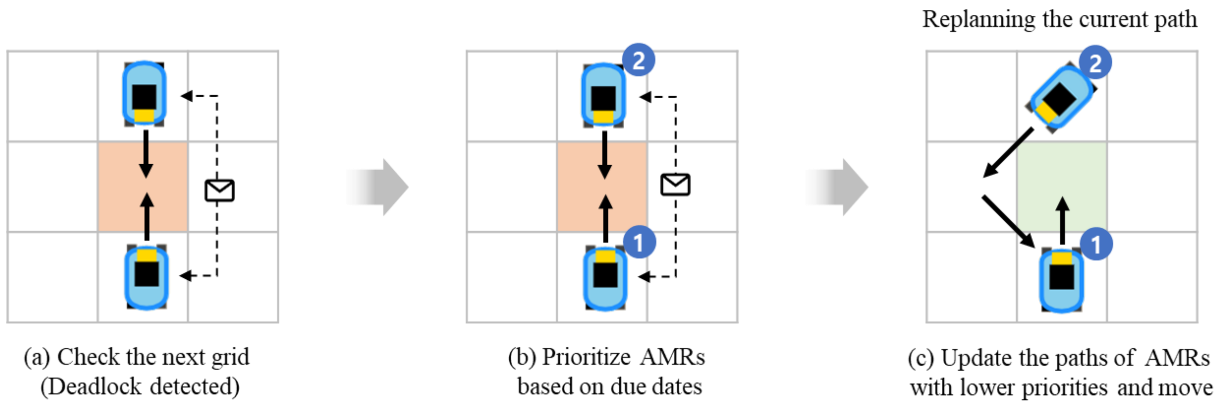

Replanning the current path
2
1
(a) Check the next grid
(Deadlock detected)
(b) Prioritize AMRs
based on due dates
(c) Update the paths of AMRs
with lower priorities and move

Figure 5.10 Deadlock resolution by updating route

Unlike collisions, in the case of deadlocks, the wait-and-move strategy cannot offer any resolution, because a deadlock occurs when an AMR is waiting for a grid occupied by another AMR to be cleared. Thus, in order to resolve the deadlock, AMRs are prioritized according to the earliest due date, and the route of an AMR having a lower priority is updated as shown in Figure 5.10.

5.4 Experimental Results

5.4.1 Experimental Design

The approach proposed in Chapter 5.3 was tested via simulation experiments. To evaluate and compare the performances of the proposed approach, 6 scenarios of 30 problem instances with different numbers of transportation requests (50, 60, 70, 80, 90, and 100) were designed. Each problem instance contained a set of transportation requests along with AMRs on a map with work centers, UL stations, and obstacles. Also, to validate the competitive performances among metaheuristics, two evolutionary algorithms that have been widely used were implemented: simple genetic algorithm (SGA) and neighborhood search (NS). Several pilot tests were run to choose the best parameters for the SGA. The initial population for the SGA was generated by adding a solution using the EDT, as well as randomly generated solutions for the population diversity. In the case of NS, all four operators in Chapter 5.3.2.2.1 were used to generate a set of neighborhoods.

Table 5.1 Parameters for generation of problem instances

Parameter	Value
Map width (meters)	500
Map height (meters)	300
Grid size	1×1 m^2
Number of AMRs	10
Number of work centers	20
Range of width and depth for work centers (meters)	[10, 20]
Number of obstacles	30
Range of width and depth for obstacles (meters)	[3, 10]
Neighborhood type of A*	Moore
Heuristic function of A*	Euclidean
Range of due dates (hours)	[0.01, 0.5]
Range of weights for transportation requests (kg)	[1, 5]
Service time (hours)	0.005
Vehicle speed (km/h)	4
Recharge time from empty to full (hours)	3
Maximum payload (kg)	100
Maximum travel time (hours)	9

Table 5.2 Parameters for VR algorithms

Algorithm	Parameter	Value
COBALT-MART	Number of trees	100
	Number of leaves	20
	Minimum number for leaf	10
	Learning rate	0.2
COBALT-CART	Maximum tree height	10
SGA	Population size	500
	Crossover rate	0.25
	Mutation rate	0.55
	Number of survivors	100
	Tournament size	5

The maximum computation time of all of the algorithms in VR was determined to be 10 minutes for prompt reflection of shop floor changes. All of the algorithms were coded in C# (for A* and COBALT) and AnyLogic (for ABM) and executed on an Intel Xeon E5-1620 3.6 GHz processor with 16 GB RAM. Tables 5.1 and 5.2 present the data relevant to the experimental design parameters used in this dissertation.

5.4.2 Results and Discussion

The minimum, maximum, and average total tardinesses of all of the algorithms for the 30 problem instances for each scenario are compared in Tables 5.3 and 5.4. In addition to the total tardiness, the relative deviation index (RDI) was used to compare the relative performances of the algorithms. The RDI of the k^{th} experiment was calculated using Equation (5). Additionally, the average RDIs for the respective test sets are plotted in Figure 5.14.

In the case of PF, as shown in Figure 5.11, A* with JPS could find the shortest paths between all possible locations with consideration of obstacles. The average computation time largely depended on the size of the problem instances: from 552.98 (with 50 transportation requests) to 1944.79 (with 100 transportation requests) seconds. Although the A* with JPS took a significant amount of computation time initially, the computation time for the upcoming transportation requests could be greatly reduced by recalculating only the shortest paths between newly added locations.

After finding the shortest paths from PF, the total tardiness is improved by solving VR, and CR finalizes the delays caused by waiting and detouring. In order to illustrate the routes of AMRs by solving VR, Figure 5.12 represents the solutions of the EDT for the problem instances of Figure 5.11. Based on a given solution from VR, as shown in Figure 5.13, ABM in CR could resolve conflicts between routes (highlighted in red) in a decentralized way and estimate the actual total tardiness considering delays. Also, the results in Table 5.3 show that the number of conflicts increased as the performance of solutions was improved. One possible explanation for this result is that, due to the improved efficiency of schedules, the shortest routes became overlapped more frequently within a short period and thus more conflicts happened, similarly to the case of traffic jams during peak times. In checking the actual total tardiness with ABM in CR, the average computation time was 160.74 seconds for all sizes of problem instances.

Also, in order to identify the delayed times caused by conflicts and their impacts on performance, a sensitivity analysis was conducted by using EDT. The number of conflicts (collisions and deadlocks) and tardiness are summarized in Table 5.5 and Figure 5.15. The results indicated that the number of conflicts increased while the average tardiness reduced significantly when the number of AMRs increased.

Among the VR algorithms, NS outperformed the others in terms of average total weighted tardiness and RDI for smaller problems (with 50, 60, and 70 transportation requests). However, the results showed that, as the size of a problem increased, the overall performance of NS worsened while that of COBALT-C improved relative to the smaller problems. In the case of COBALT-M, the results indicated that it can offer robust performance regardless of the size of problem instances. In terms of the average performance for all sizes of problem instances, the results showed that the COBALT algorithms outperformed SGA and NS, in that they yielded the smallest average total tardiness and RDI values. In summary, the simulation results demonstrated that the proposed COBALT-M has an advantage over NS and SGA in terms of finding robust solutions, and that COBALT-C appears to find good solutions, especially for larger problems.

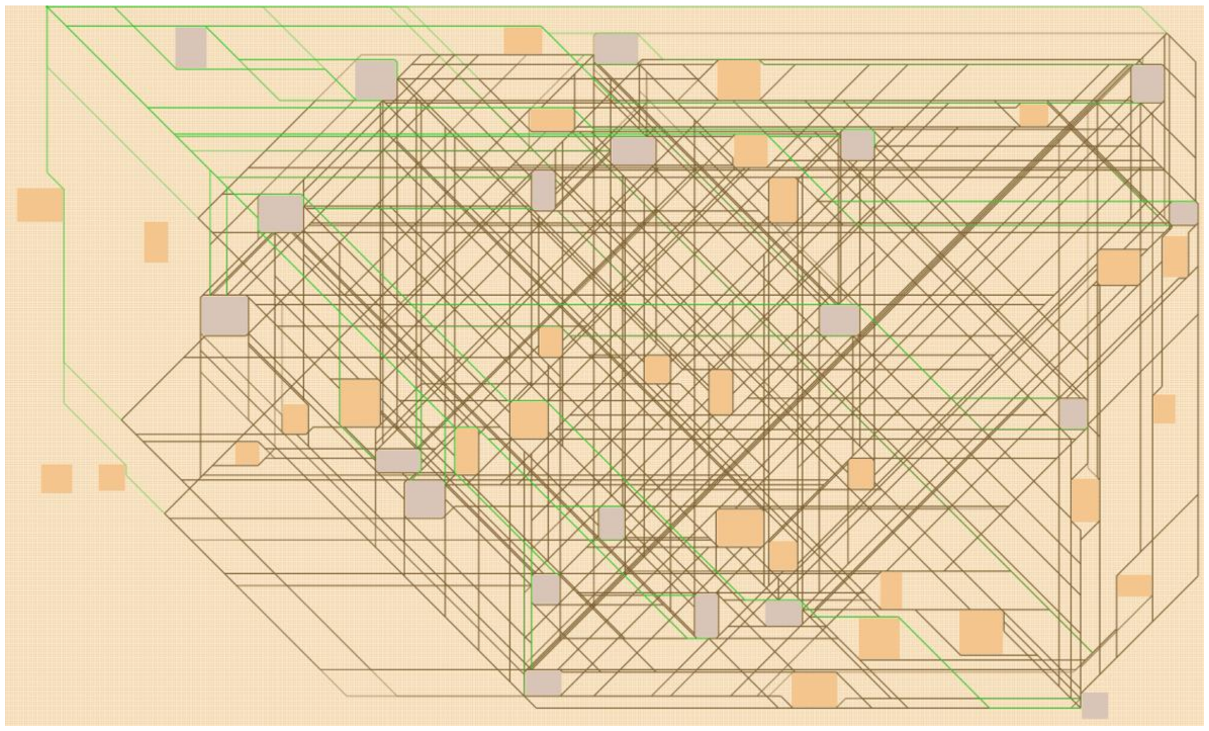


Figure 5.11 Illustration of shortest paths produced by A* with JPS. The purple and brown rectangles indicate work centers and obstacles, respectively. The green lines represent detouring paths for recharging, and the brown lines represent paths for serving of requests.

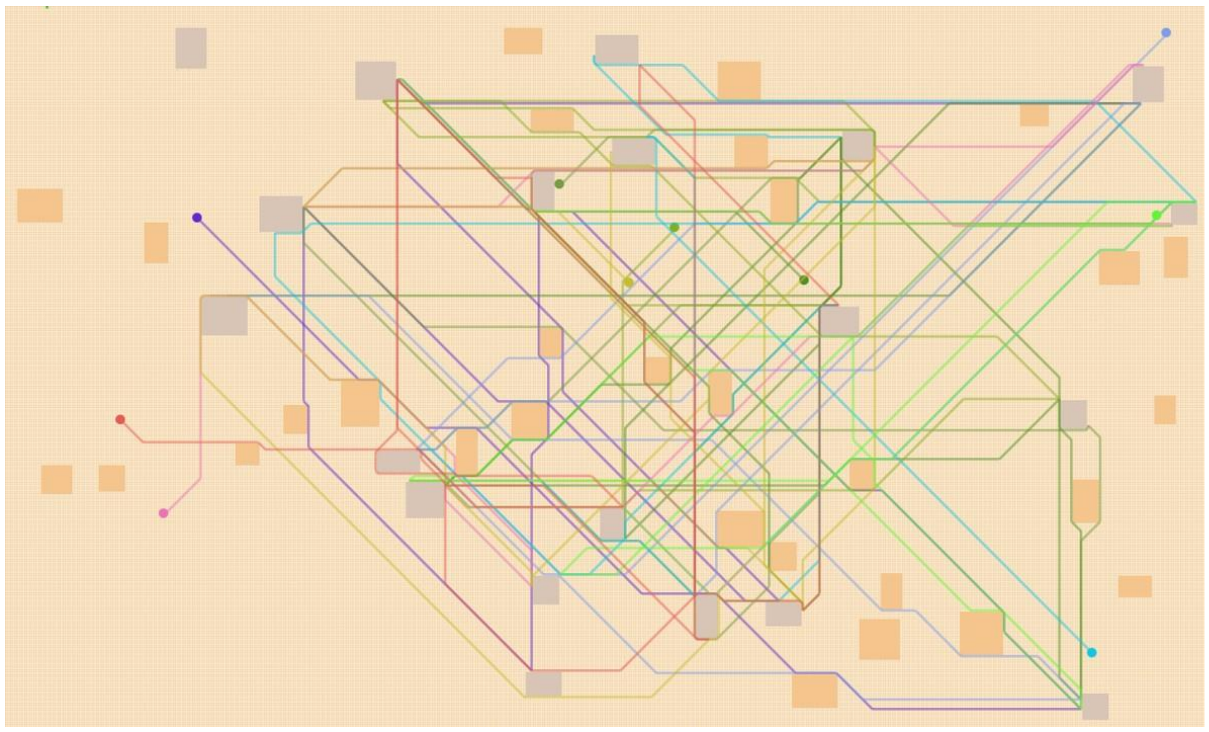
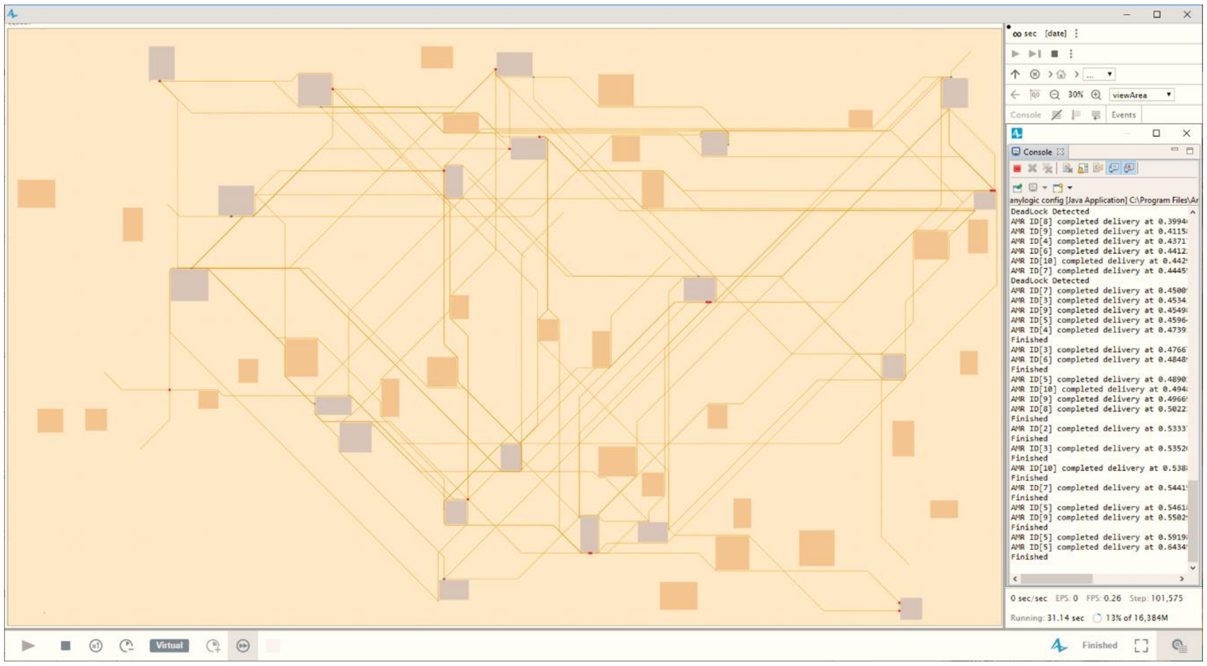


Figure 5.12 Illustration of routes for each AMR produced by EDT. Each AMR's route is differentiated by color, and its initial location is denoted as a circle.



```
anylogic config [Java Application] C:\Program Files\Ar
DeadLock Detected
AMR ID[8] completed delivery at 0.3994
AMR ID[9] completed delivery at 0.4115
AMR ID[4] completed delivery at 0.4371
AMR ID[6] completed delivery at 0.4412
AMR ID[10] completed delivery at 0.442
AMR ID[7] completed delivery at 0.4445
DeadLock Detected
AMR ID[7] completed delivery at 0.4500
AMR ID[3] completed delivery at 0.4534
AMR ID[9] completed delivery at 0.4549
AMR ID[5] completed delivery at 0.4596
AMR ID[4] completed delivery at 0.4739
Finished
AMR ID[3] completed delivery at 0.4766
AMR ID[6] completed delivery at 0.4848
Finished
AMR ID[5] completed delivery at 0.4890
AMR ID[10] completed delivery at 0.494
AMR ID[9] completed delivery at 0.4966
AMR ID[8] completed delivery at 0.5022
Finished
AMR ID[2] completed delivery at 0.5333
Finished
AMR ID[3] completed delivery at 0.5352
Finished
AMR ID[10] completed delivery at 0.538
Finished
AMR ID[7] completed delivery at 0.5441
Finished
AMR ID[5] completed delivery at 0.5461
AMR ID[9] completed delivery at 0.5502
Finished
AMR ID[5] completed delivery at 0.5919
AMR ID[5] completed delivery at 0.6434
Finished
```

0 sec/sec EPS: 0 FPS: 0.26 Step: 101,575

Running: 31.14 sec 13% of 16,384M

Figure 5.13 Screenshot of ABM simulation for conflict resolution. The grids wherein conflicts happened are highlighted in red.

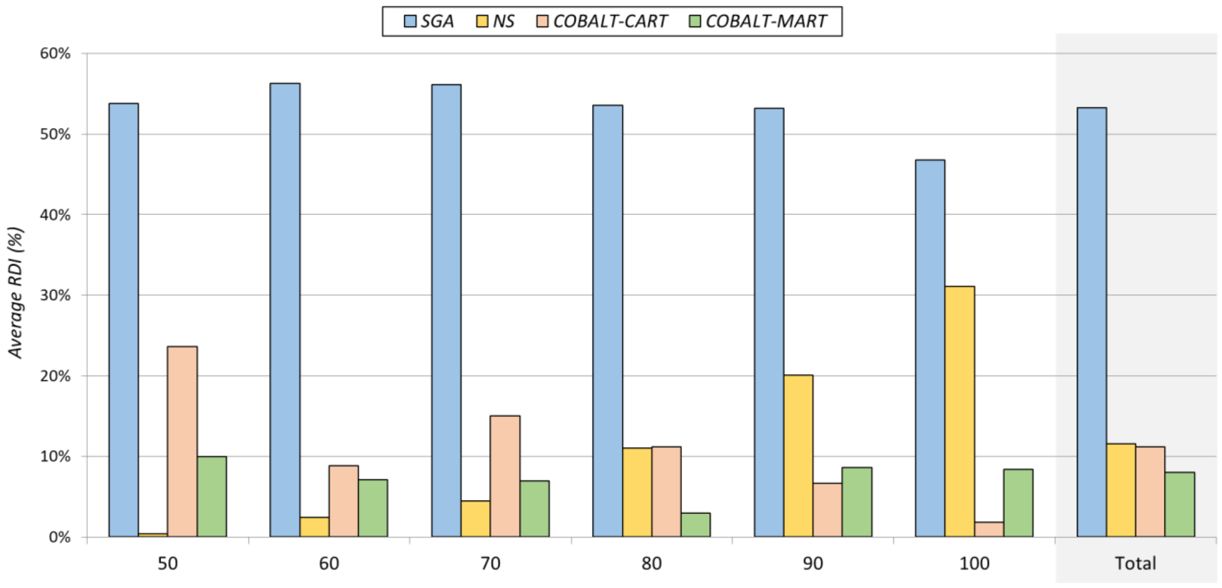


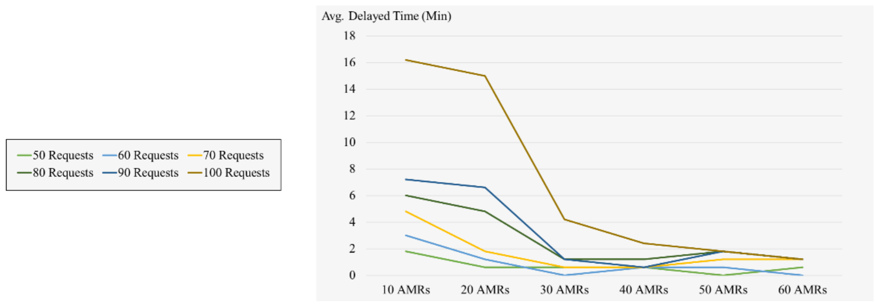
Figure 5.14 Average RDIs of algorithms for six test sets of problem instances with different numbers of transportation requests

Table 5.3 Average performances of VR algorithms for 30 problem instances with different numbers of transportation requests

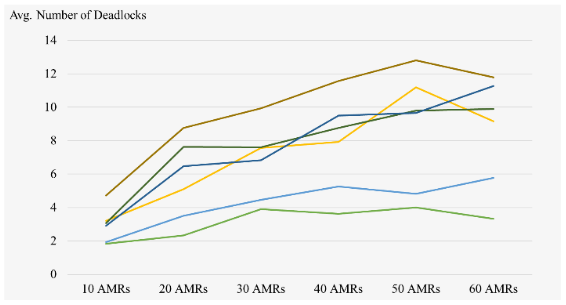
Number of requests	Algorithms	$\sum T_i$ without CR	$\sum T_i$ with CR	Number of collisions	Number of deadlocks	RDI (%)
50	EDT	4.44	4.48	8.93	2.6	100
	SGA	2.76	2.83	8.17	6.43	53.76
	NS	0.95	0.99	7	10.67	0.38
	COBALT-C	1.67	1.75	7.97	8.2	23.62
	COBALT-M	1.25	1.33	8	9.87	9.99
60	EDT	9.03	9.08	7.73	2.77	100
	SGA	5.97	6.12	8.67	8	56.29
	NS	2.34	2.52	9.53	15.33	2.45
	COBALT-C	2.81	2.98	10.63	14.07	8.88
	COBALT-M	2.69	2.86	10.9	13.63	7.13
70	EDT	13.73	13.82	11.2	2.97	100
	SGA	9.43	9.66	12.83	10.83	56.15
	NS	4.45	4.76	12.6	18.03	4.51
	COBALT-C	5.43	5.68	12.3	15.9	15.04
	COBALT-M	4.67	4.99	12.53	16.33	7.01
80	EDT	20.27	20.4	10.13	4.13	100
	SGA	14.46	14.76	13.23	12.43	53.55
	NS	9.23	9.63	14.87	17.7	11.05
	COBALT-C	9.17	9.54	18.67	16.23	11.2
	COBALT-M	8.67	9.1	15.47	17.03	6.84
90	EDT	27.09	27.3	15.17	4.67	100
	SGA	20.14	20.69	15.23	14.8	53.17
	NS	15.32	15.93	19.47	16.67	20.07
	COBALT-C	13.38	14.01	17.93	17.27	6.66
	COBALT-M	13.77	14.44	19.53	17.37	8.62
100	EDT	36.46	36.69	15	4.37	100
	SGA	27.97	28.64	20.83	15.03	46.76
	NS	25.77	26.35	19.73	12.73	31.11
	COBALT-C	21.25	21.94	18.77	14.7	1.86
	COBALT-M	22.32	22.95	20.13	14.57	8.38
Total	EDT	18.5	18.63	11.36	3.59	100
	SGA	13.46	13.78	13.16	11.25	53.28
	NS	9.68	10.03	13.87	15.19	11.6
	COBALT-C	8.95	9.32	14.38	14.4	11.21
	COBALT-M	8.9	9.28	14.43	14.8	8

Table 5.4 Result summary of all algorithms for 30 problem instances with different numbers of transportation requests

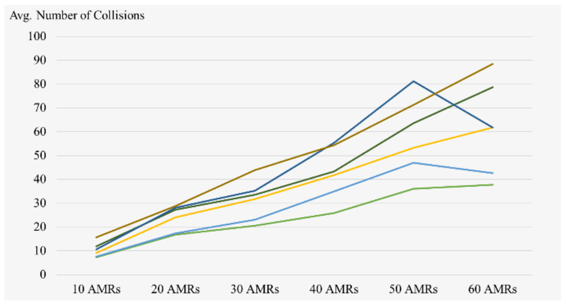
		50	60	70	80	90	100
EDT	Max.	7.12	12.17	16.75	23.63	33.08	42.32
	Min.	2.82	5.75	11.45	15.92	20.61	29.26
	Avg.	4.48	9.08	13.82	20.4	27.3	36.69
	S.D.	1.2	1.67	1.34	1.72	2.92	3.01
SGA	Max.	4.51	8.05	12.16	17.89	26.59	34.29
	Min.	1.86	3.96	7.59	10.52	15.53	22.64
	Avg.	2.83	6.12	9.66	14.76	20.69	28.64
	S.D.	0.72	1.11	1.05	1.51	2.66	2.94
NS	Max.	1.46	4.03	6.43	11.57	20.15	31.7
	Min.	0.35	1.0	3.1	7.14	12.34	20.38
	Avg.	0.99	2.52	4.76	9.63	15.93	26.35
	S.D.	0.32	0.72	0.8	1.02	1.86	2.66
COBALT-C	Max.	2.71	6.96	9.46	12.55	18.62	28.16
	Min.	0.97	1.1	2.9	5.72	10.83	16.94
	Avg.	1.75	2.98	5.68	9.54	14.01	21.94
	S.D.	1.75	2.98	1.71	1.85	1.89	2.57
COBALT-M	Max.	2.79	5.02	6.81	11.27	18.12	27.92
	Min.	0.69	1.34	3.29	6.79	9.94	16.51
	Avg.	1.33	2.86	4.99	9.1	14.44	22.95
	S.D.	0.47	0.82	0.82	1.23	2.02	2.45



(a) Average delayed time (Minutes)



(b) Average number of deadlocks



(c) Average number of collisions

Figure 5.15 Average delayed times caused by conflicts and average numbers of deadlocks and conllisions with different numbers of AMRs and transportation requests

Table 5.5 Sensitivity analysis on different numbers of transportations requests and AMRs

Number of requests	Number of AMRs	$\sum T_i$ without CR	$\sum T_i$ with CR	Number of collisions	Number of deadlocks
50	10	2.89	2.86	1.83	7.2
	20	0.46	0.45	2.33	16.77
	30	0.47	0.46	3.9	20.57
	40	0.44	0.43	3.63	25.83
	50	0.44	0.44	4	36.1
	60	0.42	0.41	3.33	37.67
60	10	7.42	7.37	1.93	7.6
	20	0.57	0.55	3.5	17.27
	30	0.43	0.43	4.47	23.1
	40	0.45	0.44	5.26	35
	50	0.44	0.43	4.83	46.9
	60	0.44	0.44	5.77	42.6
70	10	12.85	12.77	3.2	9.03
	20	1.85	1.82	5.1	24.07
	30	1.22	1.21	7.57	31.73
	40	1.16	1.15	7.93	41.83
	50	1.17	1.15	11.2	53.3
	60	1.2	1.18	9.17	61.83
80	10	21.16	21.06	3.06	11.93
	20	3.74	3.66	7.63	27.27
	30	1.57	1.55	7.6	33.47
	40	1.32	1.3	8.77	43.27
	50	1.31	1.28	9.8	63.53
	60	1.31	1.29	9.9	78.67
90	10	26.23	26.11	2.9	10.57
	20	3.82	3.71	6.47	28.03
	30	1.37	1.35	6.83	35.2
	40	1.31	1.3	9.5	55.27
	50	1.33	1.3	9.67	81.13
	60	1.29	1.27	11.27	61.8
100	10	36.54	36.27	4.73	15.63
	20	8.42	8.17	8.77	28.77
	30	1.76	1.69	9.93	43.9
	40	1.22	1.18	11.57	54.4
	50	1.16	1.13	12.8	71.27
	60	1.13	1.11	11.8	88.43

5.5 Chapter Summary

In this chapter, the three sub-problems (PF, VR, and CR) for scheduling and routing of AMRs are identified to minimize the total tardiness of all transportation requests. In order to solve the defined sub-problems practically, this study proposes a comprehensive framework that is based on three different solution approaches: A* with JPS, COBALT, and ABM. Especially, in the proposed COBALT, for the exploration and exploitation of near-optimal solutions, a tree-based regression algorithm was utilized to select the best operator with consideration of contexts. The ABM is also designed for resolving collisions and deadlocks and checking delays caused by them in a decentralized way. With simulation experiments under various numbers of requests, the proposed framework is evaluated compared to other heuristics (SGA and NS) with regard to the average total tardiness and RDI.

The major contributions of this study are the development of (1) a comprehensive framework for solving PF, VR, and CR, (2) a contextual-bandit-based optimization algorithm for VR, and (3) a decentralized and agent-based model with a set of behaviors for CR. Beyond traditional AGVs, the introduction of AMRs for material handling can enable factories and warehouses to function as more intelligent and flexible systems. Given AMRs' great potentials, their adoption should be accelerated.

CHAPTER 6. CONCLUSION AND FUTURE RESEARCH

This dissertation analyzed optimization problems for scheduling and routing with ML applications in manufacturing systems. Due to the inherent complexity of those problems, there is a significant need for discovering efficient solution approaches automatically with minimization of computation time and human intervention. In this vein, the proposed ML approaches based on DTs or reinforcement learning can help systems make better decisions while considering various environments.

In this dissertation, two scheduling problems (the SMSP and FJSP) that have been widely used in practice are studied. The proposed GRAFT and RANFORS can extract DRs as a set of IF-THEN rules from given schedules and produce high and robust performance by using an automatic process for discovering the best combination of parameters and attributes. These approaches can cover a variety of scheduling problem in manufacturing systems from a single machine to parallel machines with complicated sequences of operations.

In addition to the scheduling problems, this study identified sub-problems for scheduling and routing of AMRs and their interdependencies. In order to tackle computational complexity and inherent interdependencies, a comprehensive framework with three different solution approaches (A* with JPS, COBALT, and ABM) for each sub-problem. Especially in COBALT, in order to balance the exploration and exploitation, this study addresses a contextual bandit, which is an extension of multi-armed bandits that is a classic example of reinforcement learning.

Finally, this dissertation identifies various directions for future work in the following.

6.1 Single-Machine Scheduling Problem

In the SMSP, the following directions are interesting and worthy of further investigation. First, in order to deal with more complex scheduling problems having multiple machines, applications of GRAFT to flow shops and job shops are interesting and worthy of investigation, especially for verification of the proposed algorithm’s performance under various environments. Additionally, various objective functions such as minimization of earliness and tardiness can be studied. Finally, discretization of continuous attributes used in Chapter 4 can be supplemented to the proposed GRAFT for consideration of its potential effects.

6.2 Flexible Job Shop Scheduling Problem

Future work for Chapter 4 will proceed in the following directions. First, the RANFORS re-initiation process could be used for adaptation to system-behavior-based changes such as resource configuration variation and dynamic performance fluctuation. For example, if the difference between the training set and the recent set deviates beyond a certain threshold, the learned dispatching rule might be replaced with one newly generated to deal with the recent problems. Additionally, the proposed algorithm could be applied to other types of scheduling problems such as hybrid flow shop scheduling problems. Furthermore, dynamic environments, such as stochastic processing times or machine breakdown, also are interesting and worthy of future investigation.

6.3 Logistics with Autonomous Robots

The proposed framework covers the entire control problem for AMRs, and thus assumed the required information to be known in advance. However, unexpected changes such as newly added transportation requests and unexpected delays may be occurred according to uncertainties on the shop floor. Thus, in order to deal with those uncertainties, the proposed framework can be validated with a new set of problem instances from the real world. In addition, due to the NP-hard nature of VR, the development of automated approaches to the understanding of underlying decisions from best solutions with scalable inputs is worthy of further research. Due to similarities between VR and scheduling problems, the ML approaches used for scheduling problems such as GRAFT and RANFORS can be applied to PDP effectively (Kouki et al. 2007).

REFERENCES

Abar, S., Theodoropoulos, G. K., Lemarinier, P., & O'Hare, G. M. (2017). Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review*, *24*, 13-33.

Akhshabi, M., Tavakkoli-Moghaddam, R., & Rahnamay-Roodposhti, F. (2014). A hybrid particle swarm optimization algorithm for a no-wait flow shop scheduling problem with the total flow time. *The International Journal of Advanced Manufacturing Technology*, 70(5-8), 1181-1188.

Akturk, M. S., & Ozdemir, D. (2000). An exact approach to minimizing total weighted tardiness with release dates. *IIE transactions*, *32*(11), 1091-1101.

Anderson, E. J., & Potts, C. N. (2002, January). On-line scheduling of a single machine to minimize total weighted completion time. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 548-557). Society for Industrial and Applied Mathematics.

Arnau, Q., Juan, A. A., & Serra, I. (2018). On the use of learnheuristics in vehicle routing optimization problems with dynamic inputs. *Algorithms*, 11(12), 208.

Asama, H., Ozaki, K., Itakura, H., Matsumoto, A., Ishida, Y., & Endo, I. (1991, November). Collision avoidance among multiple mobile robots based on rules and communication. In *Proceedings IROS'91: IEEE/RSJ International Workshop on Intelligent Robots and Systems' 91* (pp. 1215-1220). IEEE.

Baptiste, P., Le Pape, C., & Nuijten, W. (2012). *Constraint-based scheduling: applying constraint programming to scheduling problems* (Vol. 39). Springer Science & Business Media.

Bergmann, S., Feldkamp, N., & Strassburger, S. (2015, December). Approximation of dispatching rules for manufacturing simulation using data mining methods. In *Proceedings of the 2015 Winter Simulation Conference* (pp. 2329-2340). IEEE Press.

Botea, A., Müller, M., & Schaeffer, J. (2004). Near optimal hierarchical path-finding. *Journal of game development*, *1*(1), 7-28.

Bożek, A., & Werner, F. (2017). Flexible job shop scheduling with lot streaming and sublot size optimisation. *International Journal of Production Research*, 1-21.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations research*, 41(3), 157-183.

Cakar, T., & Koker, R. (2015). Solving single machine total weighted tardiness problem with unequal release date using neurohybrid particle swarm optimization approach. *Computational Intelligence and Neuroscience*, *2015*, 70.

Çaliş, B., & Bulkan, S. (2015). A research survey: review of AI solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5), 961-973.

Chand, S., Traub, R., & Uzsoy, R. (1997). Rolling horizon procedures for the single machine deterministic total completion time scheduling problem with release dates. *Annals of Operations Research*, *70*, 115-125.

Chen, W., Song, J., Shi, L., Pi, L., & Sun, P. (2013). Data mining-based dispatching system for solving the local pickup and delivery problem. *Annals of Operations Research*, 203(1), 351-370.

Chen, Y. F., Everett, M., Liu, M., & How, J. P. (2017, September). Socially aware motion planning with deep reinforcement learning. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1343-1350). IEEE.

Cherkesly, M., Desaulniers, G., & Laporte, G. (2015). A population-based metaheuristic for the pickup and delivery problem with time windows and LIFO loading. *Computers & Operations Research*, *62*, 23-35.

Chmielewski, M. R., & Grzymala-Busse, J. W. (1996). Global discretization of continuous attributes as preprocessing for machine learning. *International journal of approximate reasoning*, 15(4), 319-331.

Chou, F. D., Chang, T. Y., & Lee, C. E. (2005). A heuristic algorithm to minimize total weighted tardiness on a single machine with release times. *International Transactions in Operational Research*, *12*(2), 215-233.

Chu, C. (1992). A branch-and-bound algorithm to minimize total tardiness with different release dates. *Naval Research Logistics (NRL)*, *39*(2), 265-283.

Cirillo, M., Uras, T., & Koenig, S. (2014, September). A lattice-based approach to multi-robot motion planning for non-holonomic vehicles. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 232-239). IEEE.

Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., & Kumar, T. K. (2018, July). The FastMap algorithm for shortest path computations. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (pp. 1427-1433). AAAI Press.

Congram, R. K., Potts, C. N., & van de Velde, S. L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, *14*(1), 52-67.

Curtois, T., Landa-Silva, D., Qu, Y., & Laesanklang, W. (2018). Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. *EURO Journal on Transportation and Logistics*, *7*(2), 151-192.

Dabhi, V. K., & Vij, S. K. (2011, November). Empirical modeling using symbolic regression via postfix genetic programming. In *2011 International Conference on Image Information Processing* (pp. 1-6). IEEE.

Demir, Y., & İşleyen, S. K. (2013). Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3), 977-988.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271.

Draganjac, I., Miklić, D., Kovačić, Z., Vasiljević, G., & Bogdan, S. (2016). Decentralized control of multi-AGV systems in autonomous warehousing applications. *IEEE Transactions on Automation Science and Engineering*, *13*(4), 1433-1447.

Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, *96*, 59-69.

Fattahi, P., Mehrabad, M. S., & Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of intelligent manufacturing,* 18(3), 331.

Furtado, M. G. S., Munari, P., & Morabito, R. (2017). Pickup and delivery problem with time windows: a new compact two-index formulation. *Operations Research Letters*, *45*(4), 334-341.

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2), 117-129.

Godoy, J. E., Karamouzas, I., Guy, S. J., & Gini, M. (2015, May). Adaptive learning for multi-agent navigation. In Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (pp. 1577-1585). International Foundation for Autonomous Agents and Multiagent Systems.

Goëffon, A., Lardeux, F., & Saubion, F. (2016). Simulating non-stationary operators in search algorithms. *Applied Soft Computing*, *38*, 257-268.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., & Dubey, P. (2009, August). Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 177-187). ACM.

Ham, A. M., & Cakici, E. (2016). Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. *Computers & Industrial Engineering*, 102, 160-165.

Harabor, D. D., & Grastien, A. (2011, August). Online graph pruning for pathfinding on grid maps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100-107.

Ho, Y. C., & Chien, S. H. (2006). A simulation study on the performance of task-determination rules and delivery-dispatching rules for multiple-load AGVs. *International journal of production research*, *44*(20), 4193-4222.

Ho, Y. C., & Liu, H. C. (2009). The performance of load-selection rules and pickup-dispatching rules for multiple-load AGVs. *Journal of Manufacturing Systems*, *28*(1), 1-10.

Holthaus, O., & Rajendran, C. (2005). A fast ant-colony algorithm for single-machine scheduling to minimize the sum of weighted tardiness of jobs. *Journal of the Operational Research Society*, *56*(8), 947-953.

Jakobović, D., & Budin, L. (2006, April). Dynamic scheduling with genetic programming. In *European Conference on Genetic Programming* (pp. 73-84). Springer, Berlin, Heidelberg.

Jayamohan, M. S., & Rajendran, C. (2004). Development and analysis of cost-based dispatching rules for job shop scheduling. *European journal of operational research*, *157*(2), 307-321.

Jeong, K. C., & Kim, Y. D. (1998). A real-time scheduling mechanism for a flexible manufacturing system: using simulation and dispatching rules. *International Journal of Production Research*, 36(9), 2609-2626.

Jouglet, A., Savourey, D., Carlier, J., & Baptiste, P. (2008). Dominance-based heuristics for one-machine total cost scheduling problems. *European Journal of Operational Research*, *184*(3), 879-899.

Jun, S., Lee, S., & Chun, H. (2019). Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research*, *57(10)*, 3290-3310.

Kim, K. J., & Han, I. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2), 125-132.

Koenig, S., & Likhachev, M. (2002). D* Lite. In *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, pages 476-483, 2002.

Kouki, Z., Chaar, B. F., Hammadi, S., & Ksouri, M. (2007, December). Analogies between flexible job shop scheduling and vehicle routing problems. In *Industrial Engineering and Engineering Management, 2007 IEEE International Conference on* (pp. 880-884). IEEE.

Kumar, S., & Rao, C. S. P. (2009). Application of ant colony, genetic algorithm and data mining-based techniques for scheduling. *Robotics and Computer-Integrated Manufacturing*, 25(6), 901-908.

Lawler, E. L. (1977). A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In *Annals of discrete Mathematics* (Vol. 1, pp. 331-342). Elsevier.

Li, H., & Lim, A. (2003). A metaheuristic for the pickup and delivery problem with time windows. *International Journal on Artificial Intelligence Tools*, *12*(02), 173-186.

Li, X., & Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6), 515-527.

Li, Y., Chen, H., & Prins, C. (2016). Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, *252*(1), 27-38.

Liu, M., Ma, H., Li, J., & Koenig, S. (2019, May). Task and Path Planning for Multi-Agent Pickup and Delivery. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 1152-1160). International Foundation for Autonomous Agents and Multiagent Systems.

Liu, M., Xu, Y., Chu, C., & Zheng, F. (2009). Online scheduling to minimize modified total tardiness with an availability constraint. *Theoretical Computer Science*, *410*(47-49), 5039-5046.

Long, P., Fanl, T., Liao, X., Liu, W., Zhang, H., & Pan, J. (2018, May). Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 6252-6259). IEEE.

Ma, H., Harabor, D., Stuckey, P. J., Li, J., & Koenig, S. (2019, July). Searching with consistent prioritization for multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 33, pp. 7643-7650).

Mahmoudi, M., & Zhou, X. (2016). Finding optimal solutions for vehicle routing problem with pickup and delivery services with time windows: A dynamic programming approach based on state–space–time network representations. *Transportation Research Part B: Methodological*, *89*, 19-42.

Material Handling Institute. (2018). *The 2018 MHI Annual Industry Report - Overcoming Barriers to NextGen Supply Chain Innovation.* Retrieved from https://www.mhi.org/publications/report.

McNellis, R., Elmachtoub, A. N., Oh, S., & Petrik, M. (2017). A Practical Method for Solving Contextual Bandit Problems Using Decision Trees. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI* (pp. 11-15).

Mitrović-Minić, S., Krishnamurti, R., & Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, *38*(8), 669-685.

Moore, E. F. (1959). The shortest path through a maze. In *Proc. Int. Symp. Switching Theory, 1959* (pp. 285-292).

Na, H., & Park, J. (2014). Multi-level job scheduling in a flexible job shop environment. *International Journal of Production Research*, 52(13), 3877-3887.

Nareyek, A. (2003). Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer decision-making* (pp. 523-544). Springer, Boston, MA.

Nazari, M., Oroojlooy, A., Snyder, L., & Takác, M. (2018). Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems* (pp. 9839-9849).

Nguyen, S. (2016). A learning and optimizing system for order acceptance and scheduling. *The International Journal of Advanced Manufacturing Technology*, *86*(5-8), 2021-2036.

Nguyen, S., Mei, Y., & Zhang, M. (2017). Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, *3*(1), 41-66.

Nie, L., Gao, L., Li, P., & Li, X. (2013). A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing*, *24*(4), 763-774.

Olafsson, S., & Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, *128*(1), 118-126.

Özgüven, C., Özbakır, L., & Yavuz, Y. (2010). Mathematical models for job-shop scheduling problems with routing and process plan flexibility. *Applied Mathematical Modelling*, 34(6), 1539-1548.

Öztürk, C., Tunalı, S., Hnich, B., & Örnek, M. A. (2013). Balancing and scheduling of flexible mixed model assembly lines. *Constraints*, 18(3), 434-469.

Perner, P., & Trautzsch, S. (1998, August). Multi-interval discretization methods for decision tree learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)* (pp. 475-482). Springer, Berlin, Heidelberg.

Phan, D. H., & Suzuki, J. (2016). Evolutionary multiobjective optimization for the pickup and delivery problem with time windows and demands. *Mobile Networks and Applications*, *21*(1), 175-190.

Pinedo, M. (2012). *Scheduling: Theory, Algorithms, and Systems*. New York: Springer.

Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

Rahili, S., Riviere, B., Olivier, S., & Chung, S. J. (2018, November). Optimal routing for autonomous taxis using distributed reinforcement learning. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 556-563). IEEE.

Rajabinasab, A., & Mansour, S. (2011). Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach. *The International Journal of Advanced Manufacturing Technology*, *54*(9-12), 1091-1107.

Reveliotis, S. A. (2000). Conflict resolution in AGV systems. *IIE Transactions*, *32*(7), 647-659.

Reynolds, R. G. (1994, February). An introduction to cultural algorithms. In *Proceedings of the third annual conference on evolutionary programming* (Vol. 131139). Singapore.

Reynolds, R. G. (1994, February). An introduction to cultural algorithms. In *Proceedings of the third annual conference on evolutionary programming* (Vol. 131139). Singapore.

Schouwenaars, T., De Moor, B., Feron, E., & How, J. (2001, September). Mixed integer programming for multi-vehicle path planning. In *2001 European control conference (ECC)* (pp. 2603-2608). IEEE.

Shahzad, A., & Mebarki, N. (2016). Learning dispatching rules for scheduling: a synergistic view comprising decision trees, tabu search and simulation. *Computers*, *5*(1), 3.

Sharon, G., Stern, R., Felner, A., & Sturtevant, N. R. (2015). Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219, 40-66.

Shi, J., Gao, Y., Wang, W., Yu, N., & Ioannou, P. A. (2019). Operating Electric Vehicle Fleet for Ride-Hailing Services With Reinforcement Learning. *IEEE Transactions on Intelligent Transportation Systems*.

Sigurdson, D., Bulitko, V., Yeoh, W., Hernández, C., & Koenig, S. (2018, August). Multi-Agent Pathfinding with Real-Time Heuristic Search. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)* (pp. 1-8). IEEE.

Smith, M. G., & Bull, L. (2005). Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming and Evolvable Machines*, *6*(3), 265-281.

Srivastava, S. C., Choudhary, A. K., Kumar, S., & Tiwari, M. K. (2008). Development of an intelligent agent-based AGV controller for a flexible manufacturing system. *The International Journal of Advanced Manufacturing Technology*, *36*(7-8), 780.

Sun, D., Kleiner, A., & Nebel, B. (2014, May). Behavior-based multi-robot collision avoidance. In 2014 IEEE international conference on robotics and automation (ICRA) (pp. 1668-1673). IEEE.

Tay, J. C., & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3), 453-473.

Türkyılmaz, A., & Bulkan, S. (2015). A hybrid algorithm for total tardiness minimisation in flexible job shop: genetic algorithm with parallel VNS execution. *International Journal of Production Research*, 53(6), 1832-1848.

Vinayak, R. K., & Gilad-Bachrach, R. (2015, February). DART: Dropouts meet Multiple Additive Regression Trees. In *Artificial Intelligence and Statistics* (pp. 489-497).

Wagner, G., & Choset, H. (2011, September). M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ international conference on intelligent robots and systems* (pp. 3260-3267). IEEE.

Wang, B., Xi, Y., & Gu, H. (2005). Terminal penalty rolling scheduling based on an initial schedule for single-machine scheduling problem. *Computers & operations research*, *32*(11), 3059-3072.

Wang, T., Lima, R. M., Giraldi, L., & Knio, O. M. (2019). Trajectory planning for autonomous underwater vehicles in the presence of obstacles and a nonlinear flow field using mixed integer nonlinear programming. *Computers & Operations Research*, *101*, 55-75.

Wnek, J., & Michalski, R. S. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning,* 14(2), 139-168.

Xue, L., Luo, Z., & Lim, A. (2016). Exact approaches for the pickup and delivery problem with loading cost. *Omega*, *59*, 131-145.

Yazdani, M., Amiri, M., & Zandieh, M. (2010). Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37(1), 678-687.

Yu, J., & LaValle, S. M. (2016). Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, *32*(5), 1163-1177.

Yuan, R., Dong, T., & Li, J. (2016). Research on the collision-free path planning of multi-AGVs system based on improved A* algorithm. *American Journal of Operations Research*, *6*(06), 442.

Zahmani, M. H., & Atmani, B. (2018). Extraction of dispatching rules for single machine total weighted tardiness using a modified genetic algorithm and data mining. *International Journal of Manufacturing Research*, *13*(1), 1-25.

Zhao, Y., Man, K. L., Liang, H. N., Wang, W., Yue, Y., & Jeong, T. (2015, November). Design of intelligent algorithms for multi-mobile robot systems. In *2015 International SoC Design Conference (ISOCC)* (pp. 177-178). IEEE.

Zheng, K., Tang, D., Gu, W., & Dai, M. (2013). Distributed control of multi-AGV system based on regional control model. *Production Engineering*, 7(4), 433-441.

Zhou, J. (1996). A constraint program for solving the job-shop problem. In *Principles and Practice of Constraint Programming—CP96* (pp. 510-524). Springer Berlin/Heidelberg.

Zhou, Y., Yang, J. J., & Huang, Z. (2019). Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. *International Journal of Production Research*, 1-20.

PUBLICATIONS

1. Jun, S., Lee, S., & Yih, Y. (2019). Pickup and delivery problem with recharging for material handling systems utilising autonomous mobile robots. *Manuscript submitted for publication*.
2. Jun, S., Choi, C. & Lee, S. (2019). Design of efficient control systems for autonomous mobile robots with reinforcement learning. *Manuscript submitted for publication*.
3. Jun, S.& Lee, S. (2020). Learning dispatching rules for single machine scheduling with dynamic arrivals based on decision trees and feature construction. *International Journal of Production Research* doi: 10.1080/00207543.2020.1741716
4. Jun, S., Lee, S., & Chun, H. (2019). Learning dispatching rules using random forest in flexible job shop scheduling problems. *International Journal of Production Research*, 57(10), 3290-3310.
5. Jun, S., Chang, T. W., & Yoon, H. J. (2018). Placing Visual Sensors Using Heuristic Algorithms for Bridge Surveillance. *Applied Sciences*, 8(1), 70.
6. Jun, S., Chang, T. W., Jeong, H., & Lee, S. (2017). Camera Placement in Smart Cities for Maximizing Weighted Coverage with Budget Limit. *IEEE Sensors Journal*, 17 (23), 7694-7703
7. Lee, J., Jun, S., Chang, T. W., & Park, J. (2017). A Smartness Assessment Framework for Smart Factories Using Analytic Network Process. *Sustainability*, 9(5), 794.
8. Jun, S., & Park, J. (2015). A hybrid genetic algorithm for the hybrid flow shop scheduling problem with nighttime work and simultaneous work constraints: A case study from the transformer industry. *Expert Systems with Applications*, 42(15-16), 6196–6204.
9. Jun, S., Lee, D., & Park, J. (2013). Determining business models in bottom-of-the-pyramid markets. *Industrial Management & Data Systems*, 113(7), 8-8.