Efficient Two-Stage Genetic Algorithms for Comprehensive Multi-Objective Flexible Job Shop Scheduling Problems

by

Danial Rooyani

A Thesis

presented to

The University of Guelph

In partial fulfilment of requirements

for the degree of

Master of Applied Science

in

Engineering

Guelph, Ontario, Canada

ABSTRACT

EFFICIENT TWO-STAGE GENETIC ALGORITHMS FOR COMPREHENSIVE MULTI-OBJECTIVE FLEXIBLE JOB SHOP SCHEDULING PROBLEMS

Danial Rooyani
University of Guelph, 2023

Advisor:
Professor F.M. Defersha

Among all the different scheduling problems, the Job shop Scheduling Problem (JSP) and Flexible job shop Scheduling Problem (FJSP) are the most popular and difficult optimization problems. FJSP is an expansion of JSP where an operation has a set of eligible machines, unlike only a single machine at JSP. JSP and FJSP are classified as non-polynomial-hard (NP-hard) problems that exact algorithms cannot guarantee finding the optimum solution in a finite time. So several heuristic and metaheuristic techniques have been developed to solve them, including Genetic Algorithm (GA), which is by far the most popular one. However, the quest for more efficient and effective algorithms continues.

Regular GA (RGA) for solving FJSP determines both assignments of operations to machines and their sequences simultaneously through a random process guided by the principles of natural selection and evolution. In this research, we develop a Two-Stage Genetic Algorithm (2SGA), with the first stage being different from RGA for FJSP found in the literature. The first stage has a unique solution encoding that only determines the operation sequence for assignment and then uses a greedy approach to assign the machine with the quickest completion time to each operation, considering the current machine load and process time. The first stage creates a high-quality initial population for the second stage that follows the common approach of RGA for FJSP to enable the algorithm to search the entire solution space by including solutions that

might have been excluded because of the greedy nature of the first stage.

The efficiency of 2SGA compared to RGA and some other common algorithms has been successfully tested using published FJSP benchmark problems and randomly generated examples. We also applied 2SGA to more comprehensive FJSP models with several features, including detached or attached sequence-dependent setup, machine release date, process lag time, lot streaming, outsourcing option, and subassembly requirement. Numerous comprehensive sample problems have been generated to show that 2SGA outperforms the RGA in different aspects like convergence speed and solution quality. We also noted that the superiority of the 2SGA over RGA is much greater when solving large-size and more complex problems, rendering it a viable choice for solving practical problems typically encountered in industries. We also showed that further performance improvement of the 2SGA can be achieved using parallel computation. Parallel computation is considered the most effective method to improve GA performance. However, we have demonstrated using several numerical examples that the sequential version of the 2SGA (using a single CPU) outperforms a parallel implementation of the RGA that uses many CPUs. This shows that 2SGA is potentially a game-changing concept.

The vast majority of scheduling algorithms consider only one or two objectives from a very limited list of performance metrics, while industries seek to optimize various performance measures simultaneously. Hence in this research, we expanded the 2SGA for the FJSP model to incorporate multiple (up to 10) objectives. Numerical examples are presented to illustrate the greater need for multi-objective optimization in larger problems due to their interaction and relevance in providing better solution quality. The results show the strong capability of 2SGA to jointly optimize all the objective functions and how it outperforms the RGA. However, 2SGA is not able to minimize the Earliness/Tardiness (E/T) objectives due to its greedy nature. This can be a pragmatic issue for 2SGA since job shops usually receive orders with due dates. So we have modified the 2SGA solution encoding further to include the intentional delay of the jobs and illustrated how it could address comprehensive multi-objective E/T FJSPs.

To my wife, Fatemeh*,*
who is my best friend, the love of my life,
my soul mate, and everything that I could have ever asked for.

And to my Parents*,*
who gave me all they could, so I can navigate through the journey of life.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation toward my supervisor Professor Fantahun Defersha for all his support and guidance. There were so many difficult times that I would not have been able to overcome without his advice and wisdom. I would like to thank him for being such a patient supervisor and trusting in me.

I would also like to thank my advisory committee, Professor Charlie Obimbo, Professor Ibrahim Deiab, and Professor Simon Yang, for their excellent advice and comments. I would especially like to thank Ms. Jacqueline Floyd for all her professionalism and dedication in helping graduate students.

I owe great thanks to all my managers and colleagues at Mersen Canada, Ford Canada, and AVL Manufacturing for accommodating the many days I had to take off to study. My special thanks go to Dr. Ahmed Zaghlol, who was always more than a manager to me, as he mentored me through these many years I have had the privilege of knowing him. I will also like to extend my gratitude to Mr. Vic Bakowski and Mr. Vince DiCristofaro for their support.

I am also eternally grateful to my mother and father for their endless support that has given me the foundation for life's journey. Finally, my most heartfelt thanks go to the love of my life, Fatemeh, for the countless sacrifices she has made through these years, her tremendous support, and encouragement. Thank you for being here for me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Symbols used in the Mathematical Model

Model Parameters and Indexes:

2SGA Model Common Parameters:

J	Total number of independent jobs to be scheduled in the system where each job has an index of $j = 1, \ \ldots, \ J$;
O_j	Number of operations of job j to be processed in a fixed sequence where each operation o (where $o = 1, \ \ldots, \ O_j$) can be assigned to one of several eligible machines;
M	Total number of machines where each machine has an index of $m = 1, \ \ldots, \ M$;
D_m	Release date of machine m;
R_m	Maximum number of production runs of machine m where each production run is indexed by r or $u = 1, 2, ..., R_m$;
$L_{o,j}$	Lag time (or waiting time) of operation o of job j;
$B_{m,j,o}$	Process time of operation o on machine m for whole batch of job j;
$S^*_{m,j,o}$	Setup time of operation o of job j if it is the first operation to be processed on machine m;
$S_{m,j,o,j',o'}$	Sequence dependent setup time of operation o of job j on machine m where operation o' of job j' is is the preceding operation on machine m;
$P_{m,j,o}$	Binary data equals to 1 if machine m is capable of processing operation o of job j, 0 otherwise;

$A_{j,o}$ Binary data equals to 1 if setup of operation o of job j is attached (non-anticipatory or inseparable), 0 if it is detached (anticipatory or separable);

Ω Large positive number;

Lot Streaming Model Parameters:

B_j The number of parts in a batch of job j;

S_j Total number of unequal sublots (transfer batches) of job j;

$T_{o,j,m}$ Unit-processing-time of operation o of job j on machine m;

Subassembly and Outsourcing Model Parameters:

$U_{j,o}$ Binary data equals to 1 if operation o of job j is outsourced, 0 if in-house;

$R_{j,o}$ Return time (lead time or turn over time) of operation o of job j that is outsourced;

$K_{j,j'}$ Binary data equals to 1 if job j' is an immediate child (subassembly) of job j in Bill of Material, 0 otherwise;

Earliness/Tardiness Scheduling Model Parameters:

E_j Binary data equal to 1 if job j has due date (or promised shipping date), 0 if it should ship ASAP;

D_j Due date (promised shipping date) for job j;

Variables:

C_{max} Makespan of the schedule;

$\hat{c}_{m,r}$ Completion time of run r of machine m;

$c_{j,o}$	Completion time of operation o of job j;
$c_{o,j,m}$	Completion time of operation o of job j on an eligible machine m;
t_j	Absolute value of earliness/tardiness or linearize the model;
$x_{m,r,j,o}$	Binary variable which takes the value 1 if the run r on machine m is assigned to operation o of job j, 0 otherwise;
$z_{m,r}$	Binary variable which takes the value 1 if run r of machine m has been assigned to any operation, 0 otherwise;

Lot Streaming Multi Objective Model Variables:

$b_{s,j}$	Size of sublot s (where $s = 1, \ \ldots, \ S_j$) of job j;
$c_{o,s,j}$	Completion time of operation o of sublot s of job j;
$\gamma_{s,j}$	Binary variable which takes the value 1 if sublot s of job j is non-zero ($b_{s,j} \geq 1$), 0 otherwise;
$d_{s,j}$	Departure time of sublot s of job j;
$\hat{d}_j$	Departure time of job j (maximum of $d_{s,j}$ for all s of job j);
$e_{s,j}$	Entry time of sublot s of job j;
$\hat{e}_j$	Entry time of job j (minimum of $e_{s,j}$ for all s of job j);
$f_{s,j}$	Flowtime of sublot s of job j;
$\hat{f}_j$	Flowtime of job j;
f_{max}	Maximum sublot flowtime;
$\hat{f}_{max}$	Maximum job flowtime;
f_{total}	Total sublot flowtime;
$\hat{f}_{total}$	Total job flowtime;
$\hat{g}_j$	Minimum sublot departure time of job j;
$\hat{h}_j$	Sublot finish separation time of job j;

$\hat{h}_{max}$	Maximum sublot finish separation time;
$\hat{h}_{total}$	Total sublot finish separation time;
$l_{m,o,s,j}$	Workload on machine m because of the setup and processing of operation o of sublot s of job j;
$\hat{l}_m$	Workload on machine m;
$\hat{l}_{min}$	Minimum machine workload;
$\hat{l}_{max}$	Maximum machine workload;
$\hat{l}_{total}$	Total machine workload;
$\hat{l}_{diff}$	Maximum machine workload difference;
$x_{r,m,o,s,j}$	Binary variable which takes the value of 1 if run r on machine m is assigned to operation o of sublot s of job j, 0 otherwise;
$y_{r,m,o,j}$	Binary variable which takes the value 1 if run r of machine m has been assigned to operation o of any sublots of job j, 0 otherwise;

LIST OF ACRONYMS

2SGA	Two-Stage Genetic Algorithm
AAM	Assignment Altering Mutation
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
ACO	Assignment Crossover Operator
AI	Artificial Intelligence
AIA	Artificial Immune Algorithm
AIS	Artificial Immune System
AJS	Assembly Job Shop
AL	Approach by Localization
ANOVA	Analysis of Variance
AOG	Average Optimality Gap
APC	Adaptive Parameter Control
ASP	Assembly Scheduling Problem
BOM	Bill of Material
CJS-LS	Classical Job Shop Lot Streaming
CP	Constraint Programming
DFAJSC	Dynamic Flexible Assembly Job Shop Control
DFJSP	Dynamic Flexible Job shop Scheduling Problem
DOE	Design of Experiments
DR	Dispatching Rule
DRL	Deep Reinforcement Learning

DS	Discrepancy Search
E/T	Earliness and Tardiness
EA	Evolutionary Algorithm
EGA	Enhanced Genetic Algorithm
ERP	Enterprise Resource Planning
ETPC	Extended Technical Precedence Constraints
FAHP	Fuzzy Analytical Hierarchy Process
FAJSP-LS	Flexible Assembly Job-shop Scheduling Problem with Lot Streaming
FIFO	First In First Out
FJS-LS	Flexible Job Shop Lot Streaming
FJSP	Flexible Job shop Scheduling Problem
GA	Genetic Algorithm
GDP	Gross Domestic Product
GOHR	Global Optimal Hit Rate
GPHH	Genetic Programming-based Hyper-Heuristic
GRASP	Greedy Randomized Adaptive Search Procedure
GWO	Gray wolf optimization
HFS-LS	Hybrid Flow Shop Lot Streaming
H-GA-TS	Hybrid Genetic Algorithm and Tabu Search
HGTS	Hybrid GA and Tabu Search
HGVNA	Hybrid Genetic and Variable Neighborhood Descent Algorithm
HHS	Hybrid Harmony Search
HMEA	Hybrid Many-objective Evolutionary Algorithm
HO	Hierarchical Optimization

HPC	High-performance Parallel Computation
HTS	Hybrid Tabu Search
ILS	Iterated Local Search
IM	Immune Algorithm
IOAM	Intelligent Operations Assignment Mutation
IWO	Invasive Weed Optimization
JCO	Job Crossover Operator
JIT	Just-In-Time
JLGSC	Job Level Gene Sequence Crossover
JLOSC	Job Level Operation Sequence Crossover
JSP	Job shop Scheduling Problem
LEGA	Learnable Genetic Architecture
LHS	Left-Hand Side (chromosome)
LPT	Longest Processing Time
MA	Memetic Algorithm
MA2	Multi-objective Memetic Algorithms
MAC	Machine Assignment Crossover
MAS	Multi-Agent Tabu Search Systems
MBM	Machine Based Mutation
MILP	Mixed Integer Linear Programming
MINLP	Mixed Integer Nonlinear Programming
MSD	Mean Square Deviation
MTO	Make to Order
NP-hard	Non-deterministic Polynomial-time hard
NS	Neighborhood Search

OGSM	Operation Gene Shift Mutation
OSM	Operations Swapping Mutation
OSSM	Operations Sequence Shift Mutation
P-2SGA	Parallel 2SGA
PBM	Position Based Mutation
PCM	Priority Constraint Matrix
PFS-LS	Pure Flow Shop Lot Streaming
PGA	Parallel GA
POX	Precedence Preserving Order-based Crossover
PPS	Precedence Preserving Shift Mutation
P-RGA	Parallel Regular GA
PSO	Particle Swarm Optimization
PVNS	Parallel Variable Neighborhood Search
RGA	Regular Genetic Algorithms
RHS	Right-Hand Side (chromosome)
RNG	Random Number Generator
ROAM	Random Operation Assignment Mutation
S-2SGA	Sequential Two-Stage Genetic Algorithm
SA	Simulated Annealing
SCO	Single-Point Crossover Operator
SFLA	Shuffled Frog-Leaping Algorithm
SGSM	Sublot Gene Swap Mutation
SGVM	Sublot Gene Value Mutation
SLGSC	Sublot Level Gene Sequence Crossover
SLOSC	Sublot Level Operation Sequence Crossover

SLS	Stochastic Local Search
SPT	Shortest Processing Time
SS	Scatter Search
SSC1	Sublot-Size Crossover-1
SSC2	Sublot-Size Crossover-2
TS	Tabu Search
VNS	Variable Neighbourhood Search
WBMR	Weight Biased Modified Release-rule
WIP	Work in Progress

Chapter 1

Introduction

1.1. Introduction

Manufacturing system transition was the main driving factor in starting the Industrial Revolution in the late $18th$ century with the invention of machines and the use of steam power. Manufacturing systems still play a vital role in our global economy and livelihood. According to World Bank (2020), in the last twenty years, the manufacturing sector represents 15.5% to 17.5% of the global Gross Domestic Product (GDP). This does not mean that it was always easy for manufacturing to maintain their high share of GDP. Nowadays, manufacturing is facing many challenges to meet increasing customers' demands for more customized and complicated products that are available upon request. Following the fundamental law of supply and demand, the continuously changing customer demands for highly customized and low-priced products forced industries to adopt more flexible and efficient manufacturing systems. The conventional mass production methods introduced during the Second Industrial Revolution, along with traditional assembly line, setup structure, and planning mechanism, has changed to *Flexible Manufacturing Systems* or FMS that uses advanced computerized multi-purpose machines. FMS system enables a manufacturing company to produce more variety of products in smaller batch sizes with a shorter lead time. It could keep up with the dynamic nature of customers' demands and global competition for quicker lead time and lower prices.

ElMaraghy (2005) claims manufacturing adopted at least 10 types of flexibility strategies as listed below, although some are inter-related:

1. Machine Flexibility: Machines with the capability of performing various operations,
2. Routing Flexibility: More than one feasible manufacturing routing for products,
3. Material Handling Flexibility: More possible paths between machines,
4. Operational Flexibility: Different processing plans for part production,
5. Process Flexibility: Changing between the production of different part types without major set-up changes, i.e. part-mix flexibility,
6. Product Flexibility: Ease (time and cost) of introducing products into an existing product mix, contributing to agility,
7. Volume Flexibility: Ability to change production volume profitably within production capacity,
8. Expansion Flexibility: Ease (effort and cost) of increasing capacity and/or capability through physical changes to the system,
9. Control Program Flexibility: Ability to run virtually uninterrupted (e.g. during the second and third shifts) due to the availability of intelligent machines and system control software,
10. Production Flexibility: Number of all part types that can be produced without adding major capital equipment.

Classic Job Shop Manufacturing is based on the concept of having "Machine Flexibility" that enables a manufacturing system to produce a variety of custom products in smaller batches. In this research, we focus on *Flexible Job Shop Manufacturing System* which incorporated "machine flexibility" with "routing flexibility". Routing Flexibility provides the possibility of alternative machines (which are already flexible and can handle different operations) for processing a particular product. However, this level of flexibility introduces a new level of complexity in resource sharing and production planning. Specifically scheduling becomes a more challenging problem since it is allocating several tasks to limited resources to optimize certain performance criteria. Depending

on the scheduling problem size, such as the number of tasks and resources, this task can turn into the most time-consuming and challenging activity within the enterprise. We should also consider that scheduling determines the performance of the whole manufacturing system. So in other words, to take full advantage of the flexibility of the FMS and to produce more variety of products in smaller batch sizes and shorter lead times, the company needs to develop an efficient, effective, and accurate scheduling system. This makes scheduling an extremely important part of any manufacturing system to the extent that production scheduling has become a necessity for the survival of manufacturing firms in today's competitive marketplace. The performance of the FMS with an inefficient scheduling system may be significantly low and even can take little advantage of its flexibility features. A poor scheduling system will also cause inefficient resource allocation, either over-loaded or idle capacity. This can result in a long production lead time, high cost of production, unreliable due date commitments, customer dissatisfaction, and a loss of market share, among other problems.

1.2. Job Shop Scheduling

1.2.1. Scheduling

Scheduling can be defined as the critical decision-making process of allocating available and limited resources to perform a set of tasks over a period of time in order to produce the desired outputs. In manufacturing systems, scheduling can be described as allocating production tasks (i.e., jobs, parts, and operations) to manufacturing resources (e.g., machines and operators) to be processed optimally. Every production scheduling problem aims to optimize one or a combination of performance indicators, such as minimizing the maximum time required for completing all operations (makespan), maximum tardiness (lateness), maximum earliness, maximum machine load, and so on. Hence, scheduling significantly impacts many manufacturing performance criteria, such as machine utilization, manufacturing lead times, inventory costs, meeting due dates, and customer satisfaction. In other words, production scheduling is a critical decision-making process

that plays an essential role in improving manufacturing productivity in today's competitive manufacturing world. Therefore, efficient, effective, and accurate scheduling is vital to achieving the best results. Indeed, it ultimately can determine the operational performance of a manufacturing system.

To any optimization problem, a feasible solution is an assignment of values to the variables in a way that satisfies all the constraint equations. The objective function value for this given variable assignment is called the objective function value. The optimal solution is a feasible set of variable values (solution) that results in the best objective function value. There are several well-defined procedure solution methods or algorithms for finding a good or optimum feasible solution for an optimization problem. The algorithm is called exact if it guarantees the optimal solution, no matter how lengthy or complicated the calculation is. Otherwise, it is called approximate or heuristic, which can guarantee only obtaining reasonable, feasible solutions.

Researchers have developed numerous techniques to solve scheduling problems. These techniques can be categorized into "Exact Algorithms", "Dispatching Rules", "Metaheuristics, and Artificial Intelligence". There are several Exact algorithms or mathematical techniques like linear programming, dynamic programming, or branch and bound algorithms that guarantee the global optimal solution. Unfortunately, they are all computationally intensive to the point that only makes them practical for small-size FJSPs. This is due to the flexibility of FJSP, which creates a large number of alternative solutions that consequently make the search space of these problems substantial. To avoid solving optimization problems, there are also scheduling methodologies called Dispatching Rules that are used to prioritize and release the jobs to the shop floor or a waiting machine. They follow simple decision factors such as "Shortest Processing Time (SPT)", "First In First Out(FIFO)", "Longest Processing Time (LPT)", "Minimal Slack Time" or "Earliest Due Date" or a complex combination of these factors. Unlike exact algorithms, dispatching rules are pretty straightforward and need minimum computational effort, but they cannot guarantee to achieve the optimal solution or

even a near-optimum solution. Metaheuristic algorithms (i.e., Genetic Algorithm, Simulated Annealing, Tabu search, or Ant Colony) and Artificial Intelligence Algorithms such as Neural Networks or Fuzzy Logic have been proven to be the best way to address scheduling problems. Although these algorithms cannot guarantee the optimal solution, researchers have shown that these algorithms achieve near-to-optimum (sometimes the optimal) solutions in a very reasonable computational time. They can easily be scaled and solve significant and even real scheduling problems.

It was only in the mid-1950s when Johnson (1954) introduced the first systematic approach to scheduling problems for two machines (or for three machines with some restrictions). The only reason for considering only a two-machine scenario was that production scheduling problems involve many variables and constraints and is considered one of the very difficult combinatorial optimization problems. In the simplest case, combinatorial problems mean that some or all of the decision variables take only discrete values. Most of the scheduling problems even fall into the class of Non-deterministic Polynomial-time hard (NP-hard) combinatorial problems. The name of NP-hard or non-deterministic polynomial-time hard roots in the fact that exact algorithms cannot guarantee to find their optimum solutions in finite time. To this end, it is not surprising to know that scheduling is an active field in operations research, and a countless number of articles can be found in the area dealing with wide variety of scheduling problems.

1.2.2. JSP and FJSP

Among all the different scheduling problems, the Job shop Scheduling Problem (JSP) is one of the most popular and difficult optimization problems. A classic JSP consists of scheduling a given number of jobs on a set of dissimilar machines. Each job needs to be processed through a predefined sequence of operations (routing) and designated machines. Since each operation only has a single designated machine, JSP solutions only consist of sequencing and starting time of operations on each machine which is called the sequencing problem. JSP addresses environments with a low-volume and high-variety of products or services and has a wide variety of applications, from information services to

manufacturing systems. In order to explain it in more detail, we can consider a classical JSP that consists of n jobs with O_j number of operations that need to be processed on m unrelated machines according to a predefined and fixed routing. In other words, routing or the order of operations to complete the product has been specified, and there is one and only one machine that can process each operation. So JSP only needs to solve the "sequencing problem" to determine each operation's sequence and starting times on its eligible machine. Since most of the JSPs are NP-hard problems, it has been a very active field of scheduling research and has been studied for over six decades.

JSP addresses sequencing decisions of manufacturing settings where products have a fixed routing (a set of specific machines to process the sequence of operations) that differs from one product to the other. By removing the limitation of fixed routing, Brucker and Schlie (1990) introduced the Flexible Job shop Scheduling Problem (FJSP) in 1990 as an extension of classic JSP in which each operation has an eligible machine set with the same or different process times. A classic FJSP can be described as follows: there are m different machines $(M_1, M_2, \ldots M_m)$ and also n independent jobs $(J_1, J_2, \ldots, J_n)$ where each job contains O_j number of operations. Each operation could be processed on different machines with different processing times. Processing time of operation o of job j on machine m can be indicated as $T_{(o,j,m)}$ and is given in advance along with routing of jobs (the sequence of processes).

While this "routing flexibility" can result in a better schedule by reducing machine bottlenecks, it also adds the routing problem (assigning each operation to one of its alternative machines) to the original JSP's sequencing problem (prioritizing operations on each machine). Obviously, FJSPs like JSPs are classified as NP-hard problems (Ishikawa et al. (2015)). Hierarchical and Integrated (concurrent) are two different approaches for solving "routing" and "sequencing" problems of the FJSP (Brandimarte (1993)). The integrated approach is based on the idea of solving both sub-problems simultaneously, while in the hierarchical approach, FJSP is decomposed into subproblems in order to reduce its complexity. These subproblems will be solved separately with several ways of communication among different levels. The decomposition can be based on different

approaches like the separation of constraints, but in the simplest form is separating the machine routing and operation sequencing problems (Brandimarte (1993)). In general hierarchical approach creates quick results, but its solution may not be optimal due to the influence of the different levels on each other (Xue et al. (2011)).

Having two interactive problems, routing and sequencing problems, makes FJSP even more intractable than JSP (more NP-hard too), meaning computation time increases exponentially when problem size increases. To illustrate the complexity of scheduling problems, Framinan et al. (2014) calculated the computation time of finding the solution with minimum total tardiness of a simple model with $1, 2, ..., 30$ jobs on only a single machine. There would be $1!, 2!, ..., 30!$ possible solutions as a function of the number of jobs. In general, a problem with n jobs and m machines will have $(n!)^m$ possible sequences. They considered two computers capable of doing one billion basic operations per second (the equivalent of a CPU running at 1.0GHz) and another one that is 5 million times faster and running at 5PHz (way beyond current computers' capability that rarely operates at frequencies higher than 6 GHz). For simplicity, they also assumed one full schedule could be calculated in just one basic operation (normally, n individual tardiness has to be added to calculate the total tardiness). Both computers solve problems with up to 12 jobs in less than half a second, but increasing to 15 jobs (a 25% increase) results in a CPU time of 21.79 min for the slow computer ($2,730$ times slower). Bigger size problems quickly turn infeasible for the slow computer, 99 hrs. for 17 jobs, 4 years for 19 jobs, and solving a problem with 30 job needs over 600 thousands time longer than "Age of the Universe", estimated to be 14 billion years. Even a 5 million times faster computer can practically solve problems of up to only 21 jobs that take less than 3 hrs. However, solving a problem with 30 jobs takes about 1.7 billion years. Needless to say, this means realistically sized problems are entirely out of the question, like a job shop with 50 jobs and 10 machines that will have $(50!)^{10} = 6.77 * 10^{644}$ possible sequences that is an extremely huge number (the number of atoms in the observable universe is estimated to be between 10^{78} and 10^{80}).

As it was said, each scheduling problem aims to optimize one or a set of objectives.

There are many objective functions for FJSP in the literature, including minimizing maximum completion time (makespan), minimizing machine total processing time/flow time, minimizing maximum machine load, minimizing maximum earliness/ tardiness, minimizing total Work in Progress (WIP), and maximizing equipment utilization rate. Minimizing maximum completion time (makespan) is minimizing the completion time of the last operation of the last job and is a primary measure of many scheduling systems. Makespan has been reported as the most popular performance measure for FJSP scheduling problems. Since at FJSPs, machines' loads are different according to the scheduling scheme, minimizing the maximum machine load and total workload of machines are other common objectives. A good scheduling algorithm should reduce the maximum load of machines by balancing operation assignments between machines and improving their utilization. In chapter 4, we will explore a multi-objective FJSP and show how different objectives can adversely affect each other. Also, we will demonstrate how interaction and tradeoffs between objectives increase with increasing the problem size.

1.3. Genetic Algorithm for FJSP

Genetic algorithm (GA) belongs to a larger class of Evolutionary Algorithms (EA) that use the evolution mechanism of the biological community and have a high degree of parallelism, randomness, adaptiveness, and other advantages of a combinatorial optimization search method. GA was initially proposed by Taylor (1994) and is based on Darwin's evolution theory and Mendel's genetic theory. GA combines Darwin's theory of the "natural selection: survival of the fittest" principle with a structured and randomized offspring creation.

GA starts from an initial population of chromosomes (solutions) and follows a structured but randomized algorithm to populate a new generation of chromosomes (solutions) that are more fit (i.e., better makespan in FJSP) than their parents. This evolution continues until the stop condition is met (usually the number of iterations)

and the fittest chromosome is selected as the best solution. Essential parts of any standard GA are chromosome encoding (the way each possible solution is represented), reproduction operators (selection, crossover, and mutation operators), fitness function (suitability of each solution in terms of objective function value), and GA parameters (like population size, stop condition, mutation, and crossover probability). Based on Gen and Lin (2014), five basic components of GA are:

- Solution representation or chromosome encoding,
- Initial population generation mechanism,
- Fitness function to determine the suitability of each solution,
- Genetic operators that generate offspring (selection, crossover, and mutation)
- Genetic parameter values (like population size, probabilities of applying genetic operators, etc.)

In its simplest form, GA starts by randomly creating an initial population of chromosomes (possible feasible problem solutions) based on the encoding process that converts the possible solution to a chromosome structure. Then GA selection operators randomly select some chromosomes as parents and put them in a mating pool. The selection operators give higher chances to more fit chromosomes according to the survival of the fittest principle. These chromosomes from the mating pool randomly will pair off and use crossover operators to create offspring or the next generation of the chromosomes, which are usually more fit. The offspring may also go through mutation operations to increase the chance of exploring new areas of solution space. The population of chromosomes evolves consecutively until the stopping condition is satisfied. Then the chromosome with the best fitness value is selected that will create the best solution according to the decoding procedure. GA is naturally parallel because it evaluates and improves a population of solutions. Here we will introduce the basics of the application of the Genetic Algorithm to address JSP and FJSP. Section 3.3 will provide a more detailed description while introducing the Two-Stage GA for FJSP.

GA Solution Encoding and Genetic Operators

Chromosome encoding is an essential part of the GA process. In fact, the success of the GA is greatly affected by a proper encoding solution. Researchers have used several encoding techniques to solve JSP and FJSP with GA. In section 3.3.1, we discussed the common solution encodings that have been widely used in the literature to solve FJSPs. Among all these variants, the gene structure of (j, o, m) is a prevalent form that we adopted in our novel Two-Stage GA (described in chapter 3) as the core algorithm of this research.

Additionally, we modified this solution encoding structure in chapters 4 and 5 to enable the GA process to address the FJSP problems with lot streaming and earliness/tardiness scheduling (by adding the intentional delay of the jobs) respectively. This chromosome encoding is an operation-based gene structure that takes a triple (j, o, m) where j is the job number, o is the progressive number of the operation within job j, and m is the machine assigned to that operation. This solution representation combines the machine assignment decision with the sequence decision in a single gene encoding, covering the whole solution space. This effective representation reduces memory usage and facilitates more efficient GA operators. The length of the chromosome is equal to the total number of operations to be scheduled. This representation has another important feature that allows it to model alternative routing of the problem by simply changing the index m. Our Two-Stage Genetic Algorithm uses this structure in its second stage, while the solution encoding of the first stage does not explicitly encode operations assignment and sequencing. The first stage uses a greedy approach to select the machine with the best completion time, which is described in more detail in chapter 3.

GA uses several methodologies called genetic operators that are inspired by natural evolution to evolve and produce the next generation. Genetic operators are usually categorized as selection, crossover, and mutation operators and are used to evolve the population by creating more fit chromosomes (solutions). These GA operators, along with GA parameters (i.e., population size, crossover and mutation probability rate, and termination criteria), significantly affect the GA performance. Selection operators choose

chromosomes and put them in a reproduction (mating) pool. There are three popular selection operators: (1) proportional, (2) linear ranking, and (3) tournament selections. In proportionate fitness selection or roulette wheel selection, each chromosome is assigned a selection probability equal to its fitness value divided by the total fitness of all chromosomes. Similar to the gaming roulette wheel, a chromosome with a higher fitness value will be assigned a bigger sector of the wheel that is associated with a higher probability of selection. Then a random number determines which chromosome to be selected to go into the mating pool. Since chromosomes or solutions are ranked according to their fitness, solutions with better fitness values have higher chances of being selected for mating and have higher chances for reproduction. Although this is the basis of the GA and evaluation theory of "natural selection: survival of the fittest" and helps the algorithm to explore promising areas of the solution space more than others, it also can cause the algorithm to converge prematurely to a local optimum and the algorithm to be trapped in a local minimum specifically. In a linear ranking selection, the individuals in the population are assigned ranks based on a sorted sequence of their objective function values and then selected based on a probability function, sometimes similar to roulette wheel selection. At tournament selection, k (tournament size) chromosomes are randomly selected to enter the tournament. Then the chromosome with the highest fitness value is selected as the winner and will be added to the reproduction pool. This process continues until the number of individuals in the reproduction pool equals the population size. Roulette wheel selection is less popular than it was in the past due to the chance of premature convergence if some "super individuals" exist. However, in the tournament selection that we selected for this research, there is no arithmetical computation based on the fitness value. Hence, even "weaker" chromosomes have some chance to be selected to lessen the chance of GA permutation. In Section 4.3.5, we presented the mathematical formulation of these selection operators and then, in section 4.4.3, performed comparative empirical studies which indicate the better performance of tournament selection for the proposed problem.

After forming the reproduction pool, GA applies crossover and/or mutation operators to create the next generation. The crossover chromosomes generate two children from two randomly selected parents and are paired from the reproduction pool. Crossover operators continue to apply until they create the same number of children equal to the number of parents. Then mutation operators will apply to the children according to mutation probability. Unlike crossover operators, which create two children from two parents, a mutation operator only applies and reforms one chromosome. Mutation operators introduce extra variability and help GA procedure explore a new solution space area. A simplified GA procedure has been illustrated in Figure 1.1.

Step 1.	Create an initial population of random chromosomes (solutions)
Step 2.	While termination condition has not met (like no. of generations) otherwise go to 8
Step 3.	Evaluate the fitness of each chromosome of the population
Step 4.	Produce new generation
Step 5.	Select two parents, run crossover operators, generate two offspring
Step 6.	Select one chromosome, run mutation, generate one offspring
Step 7.	Replace old generation with new offspring and go to Step 2
Step 8.	Report the chromosome with the best fitness (function value)

Figure 1.1: Simplified GA procedure

To improve GA performance, researchers have developed several strategies that can be very sophisticated. Among them, improving GA operators and enhancing GA search mechanisms are two techniques that, while highly effective in improving GA performance, are not too complicated. Selecting/creating suitable genetic operators is not only very essential for the success of any GA in gradually improving the fitness values. It also needs to ensure that the offspring do not become infeasible. To explain, we need to note that GA operators are changing the chromosome, so they can easily create infeasible

offspring, for example, due to violating the precedence constraints among operations of the same job. To fix these issues, there are two general approaches: either a repair or correction mechanism has to be defined and applied on infeasible offspring, or define the operators in such a way that they do not create infeasible solutions at all. Obviously, having such well-defined GA operators is more favorable since the repair procedure adds to the computational time. Since some FJSP studies divided the chromosome into two parts where one part defines the operation sequencing, and the other defines the machine assignment, they have to have different GA operators to apply to each part too. This way, they can avoid producing infeasible chromosomes easier. However, this chromosome encoding is more complicated than the gene of three triples (job, operation, machine) representation, in addition to the need for different sets of GA operators, which in turn increases the computational time. In this research, we have carefully selected (or created) several GA operators that respect all the constraints. Hence, no infeasible chromosomes are created, and no repairing mechanism is needed. The GA operators of each developed FJSP model are described in detail in the corresponding chapters of 3,4, and 5.

1.4. Research Structure and Motivation

The auther has worked with several local and international industries based on the job shop manufacturing concept producing many products in small batches with frequent changes in product mix and demand. Although several of these companies implemented Enterprise Resource Planning (ERP) systems, still their production scheduling heavily relied on manual planning, using spreadsheets and several meetings. Therefore, the author is motivated to contribute to scheduling knowledge to help the Job Shop Manufacturing companies. This section describes the research objectives, methodology, structure, and novelty.

1.4.1. Aim and Objective

Real-world scheduling problems are incredibly different from conventional research problems and are often very complicated. Çaliş and Bulkan (2015) pointed out that only 8% of published JSP with AI techniques papers between 1997 and 2012 have addressed industrial cases, and the rest just focused on algorithm development. One of the main contributing factors to this gap is the number of impractical assumptions that scheduling studies are considering in order to simplify the problem.

In this thesis, we developed a highly efficient GA approach that is able to solve more complex FJSP with less limitation and reaches better solutions in a shorter computational time, and then address the flexible job-shop scheduling problems with several realistic features. The objectives of this research are listed below:

- Developing a high-performance Two-Stage GA model for FJSP,
- Implementing the two-stage approach on more comprehensive FJSPs that include attached or detached sequence-dependent setup, machine release dates, and operation time lag,
- Extending the single objective FJSP to a multi-objective model and addressing it by the Two-Stage GA,
- Solving lot streaming (splitting the job into sublots) by the Two-Stage GA,
- Including product assembly structure: subassembly and final assembly jobs,
- Permitting outsourcing of some operations,
- Adding intentional delay to address earliness and tardiness objectives for the jobs with due dates,
- Concluding the superiority of the Two-Stage GA in addressing all these comprehensive FJSPs.

1.4.2. Methodologies and Structure

In this research, we first developed a highly effective Two-Stage GA approach to solve FJSP and illustrated how it outperforms regular GA for FJSP found in the literature.

Then we implemented the Two-Stage GA to solve FJSPs with several features to represent the FJSP in the real world better. We also utilized the Two-Stage to address FJSPs with multi-objective functions, including earliness and tardiness functions. This section briefly describes these developed models and the research structure.

In chapter 3, we present the novel Two-Stage GA approach and evaluate its performance by presenting several numerical examples as it was initially published in Rooyani and Defersha (2019) to solve the classic FJSP. Then we present an implementation of the Two-Stage GA to solve a more comprehensive FJSP model with attached or detached sequence-dependent setup, machine release date, and operation lag time that is published in Defersha and Rooyani (2020). Chapter 4 includes the extended model of chapter 3 considering the multi-objective function (10 objectives) and lot-streaming that is published under Rooyani and Defersha (2022). In chapter 5, we enable the Two-Stage GA to address earliness and tardiness scheduling by adding the intentional delay mechanism to the chromosome encoding. We also included subassembly requirements and outsourcing options to the model of chapter 3. In this research, we used C++ on Microsoft Visual Studio Platform to code the two-staged GA and all the extended models and sample problem generators. To solve mathematical models and linear programming, we have utilized Lingo.

Two-Stage Genetic Algorithm

Genetic Algorithm (GA) is the most popular technique for solving FJSP due to its many advantages and capabilities. To improve the performance of GA methodology for solving FJSPs, we developed a Two-Stage Genetic Algorithm with a unique and greedy mechanism for machine assignment in the first stage. In the first stage, the solution encoding only dictates the operation sequence for machine assignment. Then for each operation, the machine that can complete the operation the soonest is selected by considering the processing time and operations that are already assigned to this machine.

The second stage, starting from the solution population of the first stage, follows the typical approach of GA for FJSP which the GA determines both operation sequencing and machine assignment. This enables the algorithm to search the entire solution space by including solutions that might have been excluded because of the greedy nature of the first stage. We demonstrated via many numerical studies that we indeed transformed this simple idea into an algorithm that can potentially challenge future research in algorithm development for FJSP, particularly when solving large-size problems. This approach has been initially reported and published at Rooyani and Defersha (2019). The next step was implementing the Two-Stage GA on the FJSP model of Defersha and Chen (2010b), which has already incorporated attached or detached sequence-dependent setup, machine release dates, and operation time lag requirements into the regular FJSP. We also compared Two-Stage GA performance with GA parallel computation on 48 CPUs. Although parallel computation is known as a promising performance improvement for GA, it was interesting to find out that the Two-Stage GA running on a single CPU outperforms the parallel version of the regular GA on 48 CPUs both in terms of convergence speed and final solution quality. The result was another proof of the superiority of the Two-Stage GA. This work is reported at Defersha and Rooyani (2020) and has already attracted the attention of many researchers.

Multi-Objective GA with Lot Streaming for FJSP

Like other optimization problems, every scheduling problem aims to optimize one or more objectives, including makespan, mean and total tardiness, total completion time, and so on. Among all scheduling objective functions, minimization of the maximum completion time (makespan) has been reported as the most common objective function, followed by minimization of the total tardiness. Most FJSP studies are single objectives. Even studies with multi-objective functions have only considered one or two additional performance metrics from a very limited list, such as machine workload and production cost. However, real-world manufacturing systems cannot rely on these simplified scheduling objectives. To lessen this gap, we extended the Two-Stage GA for FJSP to

incorporate multiple (10) objectives. In addition to the initial model features (attached or detached sequence dependent setup time, process lag time, and machine release date), we extended the application of the algorithm to solve a lot streaming problem in FJSP. As described in chapter 4 the objective function of the new model includes minimization of (1) makespan, (2) maximum sublot flowtime, (3) total sublot flowtime, (4) maximum job flowtime, (5) total job flowtime, (6) maximum sublot finish-time separation, (7) total sublot finish-time separation, (8) maximum machine load, (9) total machine load, and (10) maximum machine load difference.

Chapter 4 also includes many numerical studies to illustrate several features. All ten objective function terms are illustrated using a small prototype problem to contrast the importance of multi-objective optimization in small versus large-size problems. The provided numerical examples also indicated the greater need for multi-objective optimization in larger problems due to the interaction of the various objectives and their importance in providing a better quality and more practical solution. The capability of the Two-Stage Genetic Algorithm to jointly optimize all the objective function terms is also evaluated. The quality of the initial population and the convergence behavior of the Two-Stage Genetic Algorithm is contrasted against the regular genetic algorithm with respect to each of the objective function terms. We have illustrated that the Two-Stage GA can generate initial solutions that are highly improved in all of the objectives and outperforms the regular GA in convergence speed and final solution quality in solving the multi-objective FJSP lot streaming. This work has been published under Rooyani and Defersha (2022).

Earliness/Tardiness Scheduling in the Presence of Assembly Requirements and Outsourcing

As mentioned in the previous section, minimization of total tardiness is one of the most common objective functions for FJSPs after makespan. This can be due to the fact that receiving orders with due dates or promised shipping dates is prevalent in the job shop industries. Not meeting these due dates may result in some sort of penalties or

at least causes customer dissatisfaction that can result in the loss of future business opportunities. Similarly, completing the job before its due date is not desirable either. The company may not be allowed to ship the completed job order ahead of the planned date and had to keep it as finished good inventory. This will cost the company, and there is a damaging risk due to material handling and shelf life.

As a result, there is a whole group of Earliness and Tardiness (E/T) Scheduling that tries to maximize "On-Time Completion" that includes minimization of "Earliness" and "Tardiness" objective functions. In chapter 5, we will demonstrate that the Two-Stage GA cannot address the minimization of Earliness and Tardiness objectives. So we made some changes to the chromosome encoding to allow GA to intentionally delay the starting of the jobs if needed. We have solved several examples and showed that adding the intentional delay allows the Two-Stage GA to minimize E/T objectives while maintaining its ability to minimize other objectives like makespan. The modified Two-Stage GA presented in this chapter is a multi-objective scheduling that minimizes total earliness and tardiness for the jobs that have due dates while minimizing the makespan and total completion time for jobs with no due dates.

This chapter also addressed two other unrealistic assumptions of many FJSPs. First, the classic FJSP assumes no relationship between the jobs, so they can be scheduled to start and finish independently. As a matter of fact, this is unlike real-world job shops where most of the products have subassemblies in several levels. This means that to start producing the final assembly product, all its subassembly jobs must be completed, but not necessarily simultaneously. Furthermore, products usually have more than one level of subassemblies, so the sequence of job completion should start from the lowest level of the product tree structure.

Secondly, most FJSPs assume all the operations can be done in-house. However, in the real world, a job shop relies on outsourcing some production steps due to lacking in-house capabilities or resource constraints. For example, a machine shop specializing in CNC manufacturing relies on its vendors for welding or surface coating of its products. Hence, in chapter 5, we expanded the FJSP model presented in chapter 3 with sequence

dependent setup, machine release date, and lag time to address E/T scheduling with outsourcing capability and multi-level subassembly structure.

1.4.3. Research Novelty

This research is unique and novel in many aspects. The developed Two-Stage GA approach is unique and highly efficient and has already attracted the attention of several researchers. This research is also trying to lessen the gap between FJSP research and the real world by including several realistic features that are being missed by most studies. For example, many scheduling algorithms in the literature consider only a single objective function (usually makespan), while in this research, we considered multi-objective FJSPs with up to 10 objective functions and provided rigorous numerical examples to study their optimization behavior that to the best of our knowledge is unique. This FJSP model also considers a sequence dependent setup that can be attached or detached, machine release date, and process lag time, making it even more unique. Additionally, we considered FJSPs with lot streaming, multi-level subassembly product structure, and outsourcing capability, plus allowing the intentional delay at starting the jobs to address E/T objectives.

1.4.4. Publications

Below is the list of publications as a result of this research:

1. An Efficient Two-Stage Genetic Algorithm for Flexible Job-Shop Scheduling (Rooyani and Defersha (2019)),
2. An Efficient Two-Stage Genetic Algorithm for a Flexible Job-shop Scheduling Problem with Sequence Dependent Attached/Detached Setup, Machine Release Date and Lag-Time (Defersha and Rooyani (2020)),
3. A Two-Stage Multi-Objective Genetic Algorithm for a Flexible Job Shop Scheduling Problem with Lot Streaming (Rooyani and Defersha (2022)),

4. A Two-Stage GA to Address Earliness and Tardiness FJSP in the Presence of Subassembly Requirements and Outsourcing Options (under preparation),

"An Efficient Two-Stage Genetic Algorithm for Flexible Job-Shop Scheduling" is a conference paper presented at a well-regarded 9^{th} IFAC Conference on Manufacturing Modelling, Management and Control in 2019 at Berlin, Germany. This conference paper was published by Elsevier Ltd. under Rooyani and Defersha (2019) and cited 16 times by the time this thesis was finalized. The second paper, which is the core of this research, is "An Efficient Two-Stage Genetic Algorithm for a Flexible Job-shop Scheduling Problem with Sequence Dependent Attached/Detached Setup, Machine Release Date and Lag-Time". This paper is published by Computers & Industrial Engineering Journal under Defersha and Rooyani (2020). This paper is cited 40 time in less than 3 years, referred by Lei et al. (2022) as a "*state-of-the-art meta-heuristic algorithm*" who compared their algorithm performance with ours. The third paper, "A Two-Stage Multi-Objective Genetic Algorithm for a Flexible Job Shop Scheduling Problem with Lot Streaming", expands on the work of the second paper and is published in an open access journal under Rooyani and Defersha (2022) to promote the research to a broader readership. The fourth paper, "A Two-Stage GA to Address Earliness and Tardiness FJSP in the Presence of Subassembly Requirements and Outsourcing Options", is under preparation.

The remainder of this Ph.D. dissertation is organized as follows:

- Chapter 2 reviews the contemporary literature on the different features and objective functions of FJSP, solution methods, and GA for FJSP,
- Chapter 3 presents the Two-Stage GA model and evaluates its performance in solving comprehensive FJSP model with attached or detached sequence-dependent setup, machine release date, and lag time in addition to the classic FJSP benchmark problems,
- Chapter 4 includes the extended model of chapter 3 with considering the multi-objective (10 objective functions) lot-streaming FJSP with rigorous numerical examples,

- Chapter 5 presents the modified model of chapter 3 to address the earliness and tardiness objectives in the presence of subassembly requirements and outsourcing options,
- Finally, the research conclusion and future research are provided in chapter 6.

Chapter 2

Literature Review

2.1. Introduction

As described in Chapter 1, Flexible Job Shops are able to produce more variety of products in smaller batch sizes and shorter lead times than assembly lines. In order to take full advantage of these capabilities, the company needs to develop an effective scheduling system. So production scheduling plays a vital role in improving manufacturing productivity in today's competitive manufacturing world. Hence, developing efficient, effective, and accurate scheduling becomes crucial in achieving the best results (Tamilarasi and Anantha kumar (2010)). This makes scheduling an essential part of any manufacturing company and, in general, any supply chain management system to the degree that (Metaxiotis et al., 2003) state production scheduling has become necessary for the survival of manufacturing firms in today's competitive marketplace. Indeed, scheduling ultimately determines the operational performance of the manufacturing system (Wiers and van der Schaaf (1997)).

Scheduling is defined by many researchers (like Demir and Kürşat Işleyen (2013) and Bagheri and Zandieh (2011)) as a critical decision-making process of allocating available and limited resources to perform a set of tasks over a period of time in order to produce the desired outputs at the desired time. Allahverdi et al. (2008) state the first systematic approach to scheduling problems was undertaken in the mid-1950s by

Johnson (1954) who proposed optimal scheduling for two machines (also for three machines with some restrictions). However, Johnson (1954) considered only a two-machine scenario since the production scheduling optimization problem concerns a large number of variables and constraints. In fact, production scheduling is considered one of the very difficult combinatorial optimization problems known to the research community, which most of them fall into the class of non-deterministic polynomial-time hard (NP-hard) combinatorial problems (Gen and Lin (2014)). They are called NP-hard since exact algorithms cannot guarantee finding their optimum solutions in a finite time. To this end, scheduling has become an active field in operations research, and countless articles can be found in the area dealing with a wide variety of scheduling problems.

Nevertheless, it is surprising to know that there are not many researchers who addressed practical problems. Fuchigami and Rangel (2018) has performed an interesting literature review on practical scheduling research and case studies published between 1992 to 2016. After systematic research among hundreds of scheduling studies, they have only found 46 practical papers. Publication years of these papers indicate an increasing interest in this field, where 6 and 9 out of 46 papers were published in 2015 and 2016, respectively. Among different manufacturing layouts, 31 papers (67%) have addressed flow shop (including hybrid flow shop), followed by job shop (classic and flexible) with 10 cases (22%).

Among all the different scheduling problems, the Classic and Flexible Job shop Scheduling Problem (JSP and FJSP) is one of the most popular also most difficult optimization problems (Demir and Kürşat Işleyen (2013)). For the first time, Brucker and Schlie (1990) proposed a polynomial algorithm to solve an FJSP with only two jobs. Shi et al. (2018) reviewed several Intelligent Algorithms for Solving FJSP and concluded that researchers are usually more enthusiastic about applying sophisticated improvement strategies. As a result, algorithms are becoming more complicated, although there are less complicated techniques to improve GA performance, like improving GA operators or enhancing GA search mechanisms.

This chapter presents a brief literature review of the Flexible Job Shop Scheduling

Problem concerning problem features, objective functions, and solution methods.

2.2. Flexible Job Shop Scheduling Problem Features

The classic Flexible Job Shop Scheduling Problem, as it initially appeared in Brucker and Schlie (1990), involves a number of independent jobs and a number of machines where each job has a certain number of operations to be processed in a fixed sequence. An operation can be assigned to a machine from a set of available machines. Processing times are known for every eligible pair of operations and machines, whereas setup time is not considered or simply added together with processing time. The problem is determining the operations' assignment and sequencing to minimize makespan.

Lal and Durai (2014) listed important hypotheses in the FJSP as follow:

- All the jobs and machines are ready at time zero,
- Every job has a predefined routing (operation sequence),
- Each machine can only perform one operation at a time,
- The operation processing is non-preemptive (cannot be stopped after starting),
- There is no relation between jobs (independent of each other); Similarly, machines can work independently of each other,
- There is no setup time before any operation on any machine
- The moving time between machines is not considered (negligible).

As can be seen, the classic FJSP is very basic, and it is hard to imagine finding such a simple real-life job shop manufacturing system. Thus, many variations of FJSPs are addressed in the literature to incorporate features relevant to actual industrial applications. As discussed in Chapter 1, one of the main objectives of this research is to remove several of these hypotheses to make it more practical to address real-world scheduling problems in flexible job shop manufacturing systems. Chaudhry and Khan (2016), Amjad et al. (2018), Xie et al. (2019) and Gao et al. (2019) have presented extensive literature reviews on FJSP. In this research, the machines do not have to be ready at time zero (machine release dates), the job can be split into batches and processed

in sublots (lot streaming), jobs can be related (subassembly and sequence-dependent setup), machines can have attached or detached setup time, moving between operations may need some delay time (time lag) and some operations can be outsourced.

In this section, we present a brief literature review of FJSP problem features that are common in all our research models before we provide more details of this research in chapters 3, 4 and 5.

MacCarthy and Liu (1993) have studied the "gap" between theory and practice of scheduling and concluded that classical scheduling has a very limited view of the total scheduling environment. In a quantitative study of scheduling studies carried out by Reisman et al. (1997), from a total of 184 reviewed papers, only 5 (less than 3%) dealt with realistic production settings. A more recent study done by Chaudhry and Khan (2016) reports only 12 FJSP papers (6.28%) out of 191 FJSP papers have considered industrial applications, and the other 179 papers (93.72%) were pure research-oriented and algorithm development. Also, they pointed out that most researchers just considered a simple FJSP, and only 70 papers (35.53%) considered different scenarios such as setup time, transportation times, maintenance, machine breakdown, job/machine ready times, fuzzy/uncertain processing times, overlapping operations, and re-entrant flexible job shop.

Table 2.1 listed the scheduling constraints/features that have been considered by researchers based on Fuchigami and Rangel (2018) survey. It also indicates that the constraints which rule the real-world scheduling either have not been properly studied or have not been studied at all. Setup time is one of the few important real-world manufacturing factors that are being relatively well addressed in the scheduling literature. Allahverdi et al. (2008) reviewed more than 300 scheduling papers with setup times or costs published between 1999 to 2006. They noticed an increasing interest in scheduling problems with setup times or costs. They also listed different manufacturing environments that studies considered, including single-machine, parallel-machine, flow shop, job shop, and open shop problems with about 80, 70, 100, 20, and 10 papers, respectively.

Table 2.1: List of constraints considered in practical scheduling researchers according to Fuchigami and Rangel (2018) survey

Constraint	No. of papers	Percentage
Setup times	14	42.42%
No waiting between production stages	4	12.13%
Presence of eligible machines (resources that can only handle a specific set of jobs)	3	9.09%
Different release dates & jobs dynamic arrival	3	9.09%
Batch processing scheduling	2	6.06%
Possibility of returning jobs to previous stages	2	6.06%
Adjacent constraint (no in parallel jobs execution)	2	6.06%
Blocking constraint (no intermediate storage)	1	3.03%
Presence of job availabilities intervals (the times when job processing is permitted)	1	3.03%
The learning effect (processing time depends on the job position in the sequence)	1	3.03%
Total	**33**	**100%**

2.2.1. Machine Release Date, Lag Time and Sequence Dependent Setup Time

Sequence dependent setup time is one of the most important setup-relevant features. The importance of this feature in manufacturing was noted well before the appearance of FJSP in literature (see for instance, Conway et al. (1967), Panwalkar et al. (1973), Baker (1974) and Wortman (1992)). Panwalkar et al. (1973) stated that sequence dependent setup in job scheduling is very common. Disregarding this feature could hinder manufacturers' competitive advantage (Wortman, 1992). However, its consideration was mainly limited to flow shop scheduling. In the past few decades, a considerable number of papers dealing with sequence dependent setup in FJSP were published. Another attribute of setup is the nature of being attached (or non-anticipatory: setup can be performed once the job arrives) or detached (or anticipatory: setup can be performed before the job arrival) based on technological requirements. Attached sequence dependent setup time in FJSP were considered in Guimarães and Fernandes (2006), Saidi-Mehrabad and Fattahi (2007), Bagheri and Zandieh (2011), Mousakhani (2013), Rossi (2014), and Kress et al. (2019). However, the assumption of the attached setup may adversely affect solution quality as it hinders maximal concurrency (Zhang and Gu, 2009). Özgüven et al. (2012) proposed two models: one with sequence dependent attached setup and the other with detached setup, whereas Abdelmaguid (2015) proposed a model with only detached setup time.

The assumption that all machines are available at time zero may not hold in practice, unlike the way classic FJSPs assumed. Instead, machines can have different release dates, at which time they will be free from the previous schedule. Additionally, some operations may need lag time, like for drying, cooling, cleaning, or other ancillary operations, before they can advance to the next operation. In this research, we started from the model proposed by Defersha and Chen (2010b), which already has all these features as described below:

- Machines can have a "Release Date/Time",

- There is a possibility for "Lag Time" after each operation,
- Operations can have "Setup Time" that is "Sequence Dependent" meaning it depends on the previous operation that was completed on the same machine,
- The setup of each operation can be either "Attached" or "Detached" based on technological requirements.

2.2.2. Lot Streaming

Lot streaming is a technique that splits a production lot of a job into several independent sublots and allows a sublot to be transferred from one machine to the next without waiting for the other sublots. In doing so, it enables the simultaneous processing of the sublots of a given job on multiple machines, thereby reducing the completion time of the job. The approach has been used as a strategy for a time-based competition in today's global market (Chang and Chiu, 2005). Even the Lean Manufacturing methodology emphasizes the "one-piece-flow" concept to improve productivity. One-piece-flow is basically splitting the batch as much as possible or theoretically to single pieces that is lot streaming. Since its formal introduction in Reiter (1966), Lot streaming has been an active topic of research, and many articles have been published on its application for scheduling in a variety of shop configurations. The following subsections briefly review recent articles on lot streaming based on those shop configurations. Comprehensive reviews of publications on lot streaming can be found in Chang and Chiu (2005) and Cheng et al. (2013).

Pure Flow Shop Lot Streaming (PFS-LS)

The majority of early publications on lot streaming are for pure flow shops. However, recent literature indicates that PFS-LS still continues to attract the attention of the research community. A tabu-search based three-stage algorithm for PFS-LS was developed by Buscher and Shen (2008). The three stages of the algorithm involve (i) predetermining sublot sizes, (ii) developing a schedule based on the predetermined sublot sizes, and (iii) varying the sizes of the sublots to improve the solution quality. Defersha and Chen

(2010a) developed linear programming hybridized genetic algorithm with variable sublots to minimize makespan. The authors demonstrated that in the presence of setup time, variable sublot could bring substantial improvement in makespan compared to consistent or equal-sized sublots. A genetic algorithm for PFS-LS with limited buffer capacities and equal-sized sublots was developed by Ventura and Yoon (2013) to minimize earliness and tardiness. Han et al. (2014) developed a multi-objective genetic algorithm (given the number and size of sublots) to minimize the makespan, total flowtime, machine idle time, and earliness time. Meng et al. (2018a) developed an improved migrant birds optimization for minimizing makespan with equal sublots and sequence dependant setup time. A bee colony algorithm was proposed in Gong et al. (2018) to minimize makespan and earliness in a blocking PFS-LS with no intermediate buffer between adjacent stages. The author combined setup time with processing time and assumed that the number and size of sublots are determined before scheduling. An exact heuristic based on dynamic programming and Lagrangian relaxation was developed by Alfieri et al. (2021) for a two-machine PFS-LS to minimize total flowtime. A convex programming technique for a single-job two-machine PFS-LS was developed by Fang et al. (2021) with a due date criterion and minimization of total energy consumption. The energy consumption was optimized by varying the processing speed of the sublots. Wang et al. (2022) developed an algorithm for PFS-LS with intermingling and variable sublots having detached setups and demonstrated that the assumption of a detached setup could reduce makespan.

Hybrid Flow Shop Lot Streaming (HFS-LS)

Early and some recent publications in HFS-LS (e.g., Tsubone et al. (1996), Kim et al. (1997), Zhang et al. (2005), Liu (2008), Cheng et al. (2016), and Wang et al. (2019)) are limited to a special case where there are only two stages. To the best of our knowledge, the first major research effort in lot streaming in the general hybrid flexible flow shop with more than two stages was reported in Defersha and Chen (2012b). The authors developed a parallel genetic algorithm with makespan criterion, sequence dependant setup time, and machine release date. The authors also demonstrated that lot streaming

could bring greater makespan reduction in hybrid flows shop than in pure flow shop as the former allows the overlapping of operations not only across stages but also within the parallel machines of a given stage. Nejati et al. (2014) developed a genetic algorithm for HFS-LS in the presence of work-shift constraint. The authors assumed that the processing of a sublot cannot be started if the remaining time of the work-shift does not allow the sublot to be completed, in which case the sublot has to wait for the next work-shift. Techniques based on migrant birds optimization were developed in Zhang et al. (2017) and Wang et al. (2020) to minimize total flowtime and makespan, respectively. Chen et al. (2020) proposed a genetic algorithm to minimize makespan and energy utilization. The minimization of energy utilization is achieved via machine selection, where each stage may have unrelated parallel machines with different power consumption and processing speed. Energy-aware multi-objective HFS-LS was presented in qing Li et al. (2020) to minimize average sojourn time, energy consumption, earliness, and tardiness. Zhang et al. (2022) developed an evolutionary algorithm (with consistent sublots) to minimize makespan and the number of sublots in the presence of setup and transportation.

Classical Job Shop Lot Streaming (CJS-LS)

The very first paper in lot streaming (i.e., Reiter (1966)) was for a classical job shop scheduling problem. However, research in CJS-LS is minimal. Hereunder, we reviewed relatively recent articles in the area. Chan et al. (2008) and Wong et al. (2009) considered CJS-LS with the due date criterion, where the authors assumed that all the sublots and the jobs from the same product (with a due date) would be assembled at the end of the line. Methodologies based on a generic algorithm were developed to minimize the total cost of earliness, lateness, and setup in Chan et al. (2004) and Chan et al. (2009). The authors stated that excessive lot splitting could increase setup costs. A CJS-LS problem, where a customer order contains several jobs and shipment can happen only when all the sublots of the jobs of a given order are completed, was considered in Liu (2009). The authors developed a genetic algorithm to solve the considered problem to

minimize makespan, lateness, and flowtime of finished goods. Liu et al. (2013) developed methodologies to maximize the total values of the jobs. The authors assume that the value of a job deteriorates exponentially over time, and the sooner the job completes, the higher its value is. Lei and Guo (2013) develop a bee colony algorithm with makespan criterion in the presence of a single transporter that can transfer one sublot at a time.

Flexible Job Shop Lot Streaming (FJS-LS)

Early research in job shop lot streaming was for the classical job shop. However, in recent years, the research focus on job shop lot streaming has been in FJS-LS. Defersha and Chen (2012a) developed a parallel genetic algorithm for FJS-LS with makespan criterion considering sequence dependant setup time, attached and detached nature of setups, machine release date, and lag time (a delay for cooling, drying, inspection, or other ancillary operations). Demir and Işleyen (2014) and Meng et al. (2018b) implemented FJS-LS through a successive partial transfer of a job from one machine to the next to allow operation overlapping where sublots are not scheduled independently. Bożek and Werner (2018) considered FJS-LS with variable sublot in the two-stage approach. In the first stage, the makespan is minimized with the minimum sizes of the sublots defined for the problem (larger number of sublots). In the second stage, the number of sublots is reduced without affecting makespan to minimize transportation costs. Defersha and Bayat Movahed (2018) developed linear programming hybridized genetic algorithm where the linear programming is periodically used to enhance promising solutions during the search process. Novas (2019) developed a method based on constraint programming to solve flexible job shop lot streaming with the makespan criterion. Daneshamooz et al. (2021) proposed an algorithm based on variable neighborhood search to minimize makespan for a lot streaming problem in a flexible job shop followed by a parallel assembly station. An FJS-LS problem in an "Engineer to Order" environment was presented Li (2022). The authors developed a mathematical model with variable sublot and makespan criterion and then proposed a genetic algorithm based heuristic to solve the model effectively. Though the above review indicates momentum in FJS-LS research, the total number of

publications is minimal, and further research needs to be conducted.

2.2.3. Earliness and Tardiness (E/T) scheduling

Most job shops follow the Make to Order (MTO) manufacturing strategy, where jobs are built based on the customer's order, usually including some sort of due date or customer-required date. In these situations, if the job completes earlier than the customer's required date, they have to be stored as finished goods until their due dates, incurring the inventory or "earliness costs". Similarly, jobs that are finished after their due dates may cause penalties or "tardiness costs". Therefore, one of the main scheduling objectives in these situations will be to meet the due dates of the respective jobs as closely as possible to minimize the sum of earliness and tardiness costs.

Thus, Earliness and Tardiness (E/T) scheduling has become an important research branch in the production scheduling field. Compared to the single machine, flow shop, and parallel machines E/T scheduling problems, there are fewer studies related to job shop scheduling with earliness and tardiness. (Baker and Scudder (1990)) reviewed different variations of the E/T studies since 1990, mainly single-machine and some parallel machines. The (E/T) scheduling research in job shop scheduling started in the late 1980s (Fox and Smith (1984) and (Sadeh and Fox (1996)). Other E/T job shop scheduling studies are Sadeh (1993), who designed a micro-opportunistic factory scheduler to solve a job shop earliness and tardiness scheduling problem by constraint satisfaction and optimization technology. Brandimarte (1993) has used Tabu Search to minimize makespan and weighted tardiness in a flexible job shop. Bergamaschi et al. (1997) has used job release times as model variables mainly because other variables depend on those times.

E/T scheduling, also called JIT-scheduling, since the main idea behind Just-In-Time inventory and supply chain management is to minimize (even zero) inventory level that, in fact, is the result of on-time completion or minimizing earliness and tardiness. Lauff and Werner (2004) performed a survey on the scheduling studies with just-in-time objectives. They reported that most of the papers till then (2004) are devoted to single-machine problems, also, the work on multi-operation scheduling problems, such as shop

scheduling problems, has begun only recently. Zambrano Rey et al. (2015) mentions that the scheduling objective for just-in-time production is translated into minimization of the due date mean-square deviation (MSD), quadratically penalizing inventory (earliness) costs and backlogging (tardiness) costs.

Like other types of JSPs and FJSPs, the Genetic Algorithm has been extensively used in solving E/T job shop scheduling problems. Sakawa and Kubota (2000) is one of the first scholars who presented a genetic algorithm for a job shop scheduling problem with fuzzy processing times and fuzzy due dates. Yang et al. (2010) and Yang et al. (2012) presented an Enhanced Genetic Algorithm (EGA) to solve a classic job shop scheduling (not flexible) problem with due dates and deadlines in the presence of tardiness, earliness, and flowtime penalties. Unlike most studies, they differentiated between due dates and deadlines and defined due dates as customer-desired completion that can be violated at the cost of tardiness, while deadlines come from the manufacturer and must be met. Their EGA utilizes operation-based chromosomes (since it has no machine assignment) and uses a three-stage decoder to fix the randomly generated initial population. The three-stage decoding system first reduces tardiness based on due dates, second ensures deadlines are not violated, and finally reduces earliness based on due dates. Fakhrzad (2013) included sequence-dependent setup times in a classic (not flexible) job shop scheduling with multi-objectives of minimizing the makespan and sum of the earliness and tardiness. They developed a multi-objective hybrid genetic algorithm with a variable neighborhood search algorithm.

Other optimization techniques are also used to solve E/T job shop scheduling problems. Kelbel and Hanzálek (2011) developed two constraint programming algorithms with earliness and tardiness penalties and solved an industrial case study of a lacquer production scheduling and several job-shop scheduling randomly generated problems with earliness/tardiness costs. Kusuma and Maruf (2016) developed an integer linear programming model and used the branch and bound algorithm to minimize total tardiness in a job shop environment. They assumed each job was processed and completed in a single press machine (only one operation). They showed that this mathematical

approach could provide the optimal solution for up to 70 jobs with a different processing time from 16 machines.

Dispatching rules are another approach that is used to solve E/T job shop scheduling. Chen and Matis (2013) developed a dispatching rule called the Weight Biased Modified RRrule (WBMR) that minimizes the mean tardiness of weighted jobs in an m-machine job shop. The WBMR is an extension of the RRrule with linear complexity, considers weighted jobs, and allows for biasing the schedule towards meeting the deadline of high-priority jobs. Liu and Hsu (2015) applied three existing dispatching rules and introduced nine new rules based on different due date costs and the earliness penalty to address a dynamic job shop scheduling problem with multiple delivery dates, where the time between two consecutive delivery dates is a given constant. Ahmadian and Salehipour (2021) presented a metaheuristic algorithm for the JIT–JSP, which decomposes the problem into smaller sub-problems (i.e., smaller instances, each with a few numbers of operations and machines) and then optimizes the sub-problems. Hajibabaei and Behnamian (2021) investigated an FJSP with unrelated parallel machines and sequence dependent setup time by presenting a mixed-integer linear programming model (for smaller problems) and a Tabu Search (TS) (for large-size instances) to minimize the costs of makespan, total weighted tardiness, delivery time and inventory. Machine learning is also used, Chang et al. (2022) utilized a Deep Reinforcement Learning (DRL) to solve the Dynamic FJSP (DFJSP) with random job arrival, with the goal of minimizing penalties for earliness and tardiness.

Recently, Rolim and Nagano (2020) performed a survey that covers problems where the common due date (window) is a given constraint as well as the ones where it is a decision variable. They classified 170 publications according to the objective function of the problem, the due date or due window approach, and the complexity of the available algorithms. They concluded a lot of effort was directed to combining machine scheduling with other phenomena such as resource allocation, learning, deterioration, and rate-modifying activities, especially in the single machine context. However, research on job shops and open shops is still scarce. They also identified a recent trend of combining

pure E/T scheduling problems with other phenomena such as resource allocation, deterioration, learning, machine disruption, maintenance, or even a combination of these effects. For example, Wei et al. (2021) introduced the job shop scheduling problem with the objectives of minimizing non-processing energy consumption, total weighted tardiness and earliness, and makespan based on a machine status switching framework. Also, Fan et al. (2021) addressed minimizing the mean weighted tardiness of a dynamic job shop scheduling problem with Extended Technical Precedence Constraints (ETPC) which is the precedence constraints existing between different jobs in contrast with conventional precedence constraints between operations of the same job. They developed a mathematical model to solve small-sized problems to optimality and an evolved Dispatching Rule (DR) using a Genetic Programming-based Hyper-Heuristic (GPHH) for solving industry-sized problems. Sun et al. (2021) studied many-objective flexible job-shop scheduling problems with transportation and setup times where the objective is to minimize the makespan, total workload, workload of the critical machine, and penalties of earliness/tardiness. They proposed a Hybrid Many-objective Evolutionary Algorithm (HMEA) that includes a tabu search with the neighborhood structure to improve the local search ability and a reference-point-based non-dominated sorting selection to guide the algorithm to search towards the Pareto-optimal front and maintain the diversity of solutions.

2.2.4. Subassembly Requirement

In traditional scheduling problems, including JSP and FJSP, jobs are independent of each other, which is not the case in many practical settings. Products have multi-level structures meaning they have subassemblies of different quantities that go through different manufacturing processes and can even have their own subassemblies. Therefore, in addition to E/T features, we also considered multi-level product structure in our Two-Stage GA FJSP scheduling presented in chapter 5. In our model, each final assembly can have several operations and several immediate subassemblies (child) that are considered different jobs/products with their own operation routings. Each of these

subassemblies can have its own subassemblies, and so on. "Bill of Material (BOM)" of each job/product, whether final assembly or subassembly, indicates its immediate subassemblies and their usage rate. In computer, BOM can be presented by Priority Constraint Matrix (PCM). Row and column numbers of the PCM correspond to the job numbers and their elements, indicating the assembly relationship of the jobs with binary values. We also utilized the PCM in both mathematical formulation and the Two-Stage GA. Job shop scheduling with assembly requirements is called an Assembly Job shop Scheduling Problem or AJSP. In AJSPs, each job can only start after all its subassemblies have completed their production process. Assembly Scheduling Problem (ASP) for the first time studied in 1960s to solve a multi-level assembly scheduling problem under a random environment (Li et al. (2022b)). This section provides a quick review of the previous AJSP studies and different utilized optimization techniques.

Assembly constraints make JSPs more complex (more NP-hard), and therefore in the early stages, heuristic rules of distribution and scheduling are primarily used to solve AJSPs. For example, HUANG (1984) developed a hybrid rule based on the shortest processing time (SPT) and assembly jobs first with SPT as a tiebreaker (ASMF-SPT) to minimize tardiness and process time of an assembly workshop. Adam et al. (1993) applies two priority rules of earliest Job Due Date (JDD)and the earliest operation due date (OPNDD) and four due date procedures constant allowance (CON), total work content (TWK), and critical path processing time (CPPT) in a dynamical multi-level assembly job shops based on changing job mix, workload, and resources. Even in more recent years, Zhong et al. (2020) applied controlled release to address a dynamic FJSP based on a mold industry with assembly requirements. Their Dynamic Flexible Assembly Job Shop Control (DFAJSC) includes three dynamical sub-decisions of release decision, routing decision, and sequencing decision as any disturbance occurs (e.g., machine breakdown, rush order).

Assembly requirement has been studied in different types of scheduling problems. Framinan et al. (2019) published a comprehensive review of the deterministic ASPs and categorized more than 200 ASP studies into four groups, including (1) assembly

scheduling with a dedicated machine, (2) assembly scheduling with flexible machines, (3) distributed assembly scheduling and (4) general layout (job-shop) assembly models. For the categorization purpose, they proposed a unified notation for assembly scheduling models. As can be expected, there are several GA-based ASPs. Chen and Ji (2007) introduced a GA with two minimization objectives of production flowtime and earliness and tardiness costs for scheduling products with a multi-level structure. They used an interesting encoding scheme to generate a feasible solution using product structure and the string of random numbers to avoid any repair mechanism. The jobs with the highest random priority number are selected for scheduling only if all their subassemblies (if they have any) are scheduled. In another study, Na and Park (2014) combined GA with priority rules and local search to minimize total tardiness in the FJSP with a multi-level job structure with due dates. Ming Huang et al. (2015) also addressed JSP with assembly constraints using GA. Their GA uses the common string encoding in which different jobs are represented by different numbers and are repeated the same as their number of operations. To ensure the chromosomes follow assembly constraints, a repairing mechanism switches jobs in the scheduling sequence based on the Priority Constraint Matrix (PCM).

Many other popular optimization techniques have also been utilized to solve AJSP, like Li et al. (2022b), who used a mixed-integer linear program and artificial bee colony to model a Flexible Assembly Job-shop Scheduling Problem with Lot Streaming (FAJSP-LS) as a two-stage problem where the assembly stage is the second stage for the flexible job shop manufacturing. Additionally, there are some studies that used more unique or new algorithms like Sooncharoen et al. (2020) and Li and Feng (2021) who utilized Gray wolf optimization (GWO), which is a new meta-heuristic technique inspired by grey wolves leadership hierarchy and hunting mechanism to address assembly job shop scheduling. Sooncharoen et al. (2020) addressed production scheduling of a Capital goods manufacturing of highly customized products in order to minimize the combination of earliness and tardiness costs. While Li and Feng (2021) developed a multi-objective GWO to consider completion time and total energy consumption minimization, including

non-production energy consumption.

Among all the ASP papers that Framinan et al. (2019) reviewed, only a few papers addressed general layout (job-shop) assembly models. To the best of our knowledge, no assembly FJSP has been studied with all other different features that we have in our model, including detached or attached sequence dependent setup time, outsourcing some operations, and the jobs with and without due dates, in addition to the multi-objective Two-Stage GA as it is described in chapter 5.

2.2.5. Outsourcing Options

Outsourcing is another feature that is common in real-world job shop manufacturing but is not considered in the classic job shop scheduling literature. Classic JSP and FJSP assume all jobs and operations are being processed in-house. However, due to the increasing complexity of products and the specialization of enterprises, many job shops outsource part of their auxiliary processes. Also, for some job shops, meeting a tight deadline is so crucial that they decide to outsource even the processes with internal capability. This is the author's professional experience as well that several companies rely on available and low-cost outsourcing services, especially in industrially advanced regions like southwestern Ontario. So it was a surprise to see that this aspect of job shop scheduling is not well studied yet. This section summarizes almost all the studies we identified under outsourcing in job shop scheduling.

Safarzadeh and Kianfar (2019) provided a comprehensive literature review on outsourcing in scheduling. They have classified these studies based on machine environment types: single machine, parallel machine, flow shop, and job shop. The earliest publication of any reported machine environment type is from 2005. Chung et al. (2005) is one of the only three JSP with outsourcing studies that Safarzadeh and Kianfar identified before 2019, and our literature search also confirms it. The authors of both other two papers are Xiuping Guo and Deming Lei, which were published in 2014 and 2016. We should add that there are prior studies that applied overtime scheduling as a solution to achieve tight due dates rather than outsourcing or subcontracting (Chung et al.).

As we mentioned, to the best of our knowledge Chung et al. (2005) is the first study that addressed a JSP with outsourcing. They presented a 2 phase heuristic algorithm for JSP with due-date constraints and the outsourcing option. The first phase improves the sequence of operations on a bottleneck machine to minimize the maximum lateness of the jobs by scheduling the in-house operations. Then, in the second phase, a branch-and-bound algorithm identifies the operations to be subcontracted on a bottleneck machine for the late jobs. These two phases are repeatedly executed together to obtain good solutions. Another interesting aspect of this research is that they assumed each operation could be subcontracted independently.

Based on Safarzadeh and Kianfar (2019) literature review, most researchers (including themselves) have considered outsourcing the whole job rather than outsourcing only some operations. This is the case for the following two published papers by Guo and Lei. Guo and Lei (2014) presented a two-phase neighborhood search (TPNS) to solve a bi-objective JSP to minimize total tardiness and the outsourcing cost where some jobs (not operations) are outsourced. In the first phase of TPNS, an initial solution pool is generated randomly and improved using four neighborhood structures. The second phase uses outsourcing and scheduling decisions to improve the best solution of the first phase. Then in their 2016 paper, they changed total outsourcing cost from an objective function to a constraint with an upper bound. This way, minimizing total tardiness is the only objective, and outsourcing is just an effective method to achieve this goal. They applied a new metaheuristic algorithm named Shuffled Frog-Leaping Algorithm (SFLA) to minimize total tardiness. Later Safarzadeh and Kianfar (2019) for the first time considered the makespan as the objective for JSP with outsourcing in addition to outsourcing cost. They proposed a Mixed Integer Linear Programming (MILP) and Constraint Programming (CP) to minimize the weighted sum of makespan and total outsourcing cost of a JSP with the option of outsourcing some jobs (not operations). They also developed two problem relaxation approaches to obtain strong lower bounds for some large-scale problems for which exact methods cannot attain optimal solutions in a reasonable time.

There are not many JSP outsourcing papers after 2019 either. Xu et al. (2021) developed a Hybrid Genetic Algorithm and Tabu Search (H-GA-TS) to combine advantages of GA in global search and TS in local search to address minimizing makespan, costs, quality, and carbon emission in distributed FJSP with outsourcing. They used the Fuzzy Analytical Hierarchy Process (FAHP) to transform the multi-objective problem into a single-objective problem. Gao et al. (2022) considered a unique scenario of subcontracting semi-finished parts when the job cannot complete on time. In their study, no operation from the middle of the job routing can be outsourced. Only the first set of operations can be outsourced and the semi-finished part comes to be completed in-house. Considering this outsourcing restriction, they have addressed the no-wait job shop scheduling problem with no waiting or interruption between two consecutive operations of the same job. The problem's main objective is minimizing subcontracting costs while meeting the deadlines and they have proposed two mathematical models of integrated MILP (MILP-IS) and a rolling timeline MILP (MILP-RTL) to solve it. For solving bigger size problems an artificial bee colony algorithm based on a rolling timeline (RTL-ABC) is also developed. Su et al. (2022) proposed a self-organizing neural scheduler (SoNS) for FJSP with the total tardiness objection under the periodic maintenance and mandatory outsourcing (FJSP-PMMO). The mandatory nature of outsourcing is coming from the fact that many manufacturing companies prefer to outsource a portion of their auxiliary processes rather than acquiring the equipment and skillset to be able to focus more on their core processes. It is unlike the other studies that outsourcing is optional as a way to improve tardiness. Li et al. (2022a) also considered a unique scenario in which outsourcing constraints determine available start and end times for outsourcing operations. They presented a sequence-based mathematical model and a hybrid self-adaptive differential evolution algorithm with heuristic strategies (HSDE)to minimize weighted overdue days considering these outsourcing constraints and jobs with different priorities.

2.3. Scheduling Objective (Single and Multi-objective)

Any scheduling problem attempts to optimize one or more objective functions. While scheduling objectives in industries vary, researchers mainly focused on a limited number of objective functions such as minimizing makespan, mean and total tardiness, total completion time, and machine workload. Many researchers, including Bagheri and Zandieh (2011), reported the makespan criterion (minimization of the maximal completion time of all operations) as the most common objective function in solving JSPs. Çaliş and Bulkan (2015) reported that 55% of JSP papers (which utilized AI and were published between 1997 and 2012) had used a single makespan criterion and only 19% considered multi-objective function. Among due date-based criteria, Mousakhani (2013) reported minimization of total tardiness as the most common one. Among practical scheduling studies and case studies that Fuchigami and Rangel (2018) reviewed, makespan or a related function is the most used criterion (54%), then tardiness measures with 26% of the papers. Other less frequent measures were production costs related (15%), minimizing the mean flowtime (11%), lateness (9%), total setup time (7%), electricity cost (7%) and idle time (4%).

Chaudhry and Khan (2016) after closely examining 191 FJSP journal papers published between 1990 to 2014 also reported makespan as the most popular performance measure used in 166 or 84.26% of all reviewed FJSP articles. Makespan was used as the sole objective function in 88 research papers (44.67%) and in combination with another objective function in other 78 papers (39.59%). Since in FJSPs, machine loads are different according to the scheduling scheme, and a suitable scheduling algorithm is expected to reduce the maximum load of machines by balancing operation assignment, the combination of machine load objectives and makespan is the second most common objective that was used in 23% of FJSP papers. Authors listed a total of 55 different objective functions, where 49 of them were only used once. Table 2.2 shows the six performance measures (three single performance measures and three multi-objectives) used by over 75% of published studies. They also concluded that 53% of papers had

used single objective and 47% multi-objective performance measures. Makespan remains popular even in more recently published papers (e.g., Ishikawa et al. (2015), Li and Gao (2016), Shen et al. (2018), Defersha and Bayat Movahed (2018)). However, because of increased environmental awareness in recent years, FJSP problems that integrate traditional performance measures with the minimization of shop-floor energy consumption also started to emerge (e.g., Mokhtari and Hasani (2017); Dai et al. (2019); Meng et al. (2019); Luo et al. (2020a)).

Table 2.2: Various objective functions reported by Chaudhry and Khan (2016)

Performance measure	No. of papers	Percentage
Makespan	88	44.67%
Minimum of makespan, workload of most loaded machine, total workload of machines	46	23.35%
Minimum of makespan and mean tardiness	5	2.54%
Minimum of makespan and production costs	4	2.03%
Total tardiness	3	1.52%
Minimum of mean tardiness	2	1.02%

It is evident that many real-world scheduling objectives like meeting due dates, minimizing flowtime, work-in-process inventory, or maximizing machine utilization are missing from this table (2.2). However, this is not the only issue. According to Lal and Durai (2014) and as we will show in chapter 4, some scheduling performance measures are in tradeoff with each other, which means improving some objectives can have the opposite effect on the rest. For example, finishing jobs too early in a just-in-time environment may cause unwanted excess WIP or finished goods. Therefore, optimizing a wide variety of objectives is vital to achieving good scheduling.

One of the common methods of solving a multifunction scheduling problem is combining all objectives into a single weighted aggregating function like Bagheri and Zandieh (2011) proposed a bi-objective case of minimizing makespan and mean tardiness for FJSP with sequence dependent setup times where the two objectives are combined

into a single weighted function. In Xing et al. (2009) research, users can rank objective preferences with "very important", "important", and "unimportant" weight tags. An interesting modification to this approach was used by Kachitvichyanukul and Sitthitham (2011), who developed a Two-Stage GA that, in the first stage, parallel GAs evolve for each individual objective, and then in the second stage, all are combined into one population and evolve toward a single weighted objective function.

2.4. Solution Methods

As we discussed, FJSP is an extension of the classical JSP in which, during the scheduling process, an operation can be assigned to a machine from a set of available alternatives. Thus, FJSP integrates a routing problem (assigning operations to machines) and sequencing problems (sequencing the operations on each machine) that need to be solved simultaneously. The flexibility of FJSP improves scheduling efficiency (i.e., shorter makespan) while it makes solving FJSPs more complex than JSPs (Defersha and Bayat Movahed (2018)). Additionally, JSP and FJSPs are NP-hard (Wang et al. (2008)) which means exact algorithms cannot guarantee the optimum solution in finite time (Kachitvichyanukul and Sitthitham (2011)).

Sun et al. (2010) list two approaches to solve routing and sequencing problems: concurrent (also known as integrated) and hierarchical approaches. The hierarchical approach reduces the complexity of FJSP by separating the assignment decisions from the sequencing decisions of operations into two sub-problems. Since the sequencing problem is the classical JSP, this approach was more popular at the time FJSP was introduced. Brandimarte (1993) is one of the early researchers that proposed a hierarchical algorithm for the FJSP with two-way information flow between routing and sequencing problems. They tackled both problems with Tabu Search (TS) algorithm for different objective functions and solved benchmark problems to present the algorithm's efficiency. On the other hand, the concurrent (or integrated) approach simultaneously assigns each operation to an eligible machine (routing problem) and prioritizes the operations on

the machines (sequencing problem). As can be guessed, concurrent approaches achieve higher quality results, and researchers developed several techniques using this approach.

The flexible job shop scheduling problem continues to attract many researchers because of its broad applications in real manufacturing settings. From 1990 that Brucker and Schlie (1990) for the first time proposed an algorithm to solve an FJSP with only two jobs, several approaches have been developed by researchers in three categories of "exact algorithms", "dispatching rules", "metaheuristic, and artificial intelligence" as it was introduced in Chapter 1. Among all the different solution techniques, metaheuristics have proven to be a promising method for solving FJSP as an NP-Hard problem. Although the metaheuristic algorithms cannot guarantee finding the global optimum solution for FJSPs, they can find very satisfactory solutions (near optimal) in a reasonable computational time. While finding the optimal solution through the exact algorithms for the same FJSPs can be very time-consuming, to the point that it becomes impractical due to the manufacturing priorities and timeline.

In recent years, researchers have developed different metaheuristic techniques like Genetic Algorithms, Tabu Search, Petri Net, Ant Colony, and also hybrid algorithms (a combination of two or more techniques) that can find good solutions (not necessarily global optimum) for FJSPs. Chaudhry and Khan (2016) identified 197 different applications or techniques to address FJSP in 191 FJSP journal papers (some papers presented more than one technique). The authors have categorized all those techniques/methods into 14 groups of Ant Colony Optimization (ACO), Artificial Bee Colony (ABC), Artificial Immune System (AIS), Evolutionary Algorithms (EA), Greedy Randomized Adaptive Search Procedure (GRASP), integer/linear programming, Neighborhood Search (NS), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS), mathematical programming, deterministic heuristics, hybrid techniques that is the combination of two or more techniques and miscellaneous techniques.

Wang et al. (2008) mention these meta-heuristic algorithms can achieve a satisfactory solution for FJSP, but their performance is highly dependent on the parameters used. Among all different metaheuristic algorithms, the Genetic Algorithm has been

identified as the most effective evolutionary technique for solving JSP and FJSP (Amjad et al. (2018)). From the time that Mesghouni et al. (1997) and Chen et al. (1999) used GA for the first time to solve FJSP, GA became the most popular technique among all different metaheuristic techniques that various researchers have used for solving FJSP, and even its popularity is increasing in the recent years. Chaudhry and Khan (2016) show EA and, more specifically GA, is the most popular technique used by 34% (or 65 papers out of 191) of the publications, whether in hybrid or pure form. In a more recent survey, Amjad et al. (2018) reviewed 190 FJSP papers using a variant of GA from a total of 384 articles found on the FJSP published from 2001 to December 2017. Also, they indicated that 79% of listed articles had been published in the last seven years from 2009 to 2017, which means GA popularity is increasing. Fuchigami and Rangel (2018) also reported similar findings on the popularity and effectiveness of metaheuristic and GA applications. After systematic research among hundreds of scheduling studies published between 1992 to 2016, the authors only found 46 practical papers. More than half of the papers (26 cases or 68%) used meta-heuristics, 17 papers (or 45%) used Mixed Integer Linear Programming (MILP) models, and 16 papers utilized heuristics, including computer simulations and priority rules. MILP is the only exact solution method that has been used, with no case of branch-and-bound or dynamic programming. Within meta-heuristics algorithms, the genetic algorithm by far is the most popular technique (47% of cases) that indicates its effectiveness in practical scheduling problems. Other techniques are Simulated Annealing with 9%, Neural Networks, Ant Colony, Variable Neighborhood Search, and Particle Swarm Optimization, each with a 6% frequency of use.

As discussed in Chapter 1, the Genetic Algorithm belongs to a larger class of Evolutionary Algorithms that use the evolution mechanism of the biological community and have a high degree of parallelism, randomness, adaptiveness, and other advantages of a combinatorial optimization search method. EAs are a set of population-based algorithms that, in addition to GA, include the Memetic Algorithm (MA), Immune Algorithm (IM), and Scatter Search (SS). A review of the research trend in GA-based

FJSP methods can be found in Amjad et al. (2018), whereas that of SI and EA, in general, was presented in Gao et al. (2019). In addition to EA, Sergienko et al. (2009) introduced 6 other combinatorial optimization techniques categories. These 7 categories are (1) sequential algorithms, (2) deterministic local search, (3) stochastic local search, (4) swarm intelligence, (5) evolutionary algorithm, (6) scanning methods, and (7) other special methods, including exact algorithms. Among these methods, 3, 4, and 5 are commonly used in solving FJSP.

Stochastic Local Search (SLS) includes Simulated Annealing (SA), Iterated Local Search (ILS), Greedy Randomized Adaptive Search Procedure (GRASP), and Tabu Search (TS). Following are some examples of the SLS application in FJSPs. Cruz-Chávez et al. (2017) developed an SA-based algorithm for the basic FJSP problem. ILS algorithm that independently and iteratively changes machine assignment and operation sequencing was proposed in Ishigaki and Takaki (2017). Rajkumar et al. (2011) developed GRASP for multi-objective FJSP to minimize (i) makespan, (ii) maximum workload, and (iii) total workload. A TS was developed in Jia and Hu (2014) with these same three objectives. Lot streaming in FJSP using TS was addressed in Fernández Romero et al. (2018).

Swarm Intelligence (SI) refers to a collection of techniques based on insects' social behavior in solving complex problems by interacting with each other and their environment. These techniques include, among a few others, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and Bee Colony Optimization (BCO). A survey of these techniques for various types of discrete optimization can be found in Krause et al. (2013). Their applications in FJSP are also abundant. For instance, ACO-based algorithms for FJSP with sequence dependent setup and transportation time were proposed by Rossi and Dini (2007) and Rossi (2014). Shahgholi Zadeh et al. (2019) developed a BCO algorithm for scheduling dynamic FJSP that involves rescheduling when unexpected changes cause deviation from the primary schedule. A two-level PSO for FJSP was developed by Zarrouk et al. (2019), where the top-level deals with machine assignment and the lower-level deals with operations sequencing.

2.5. Genetic Algorithm for FJSP

GA was introduced in 1975 by Taylor (1994) and since then has gained sprawling popularity in solving a wide range of complex problems in different disciplines. A significantly large number of researchers applied the genetic algorithm to solve very diverse problems in manufacturing systems design and operation. GA is the most popular technique in solving JSP and FJSP. Çaliş and Bulkan (2015) listed GA as the most popular AI (Artificial Intelligence) technique in solving JSP. They reported that 26.4% of JSP papers using AI published between 1997 and 2012 used GA. Falkenauer and Bouffouix (1991) was one of the first researchers who developed a GA to solve JSP. Their chromosome encoding consists of several strings equal to the number of machines, where each string shows the sequencing of operations on the machine. The GA model for FJSP has been initially developed by Mesghouni et al. (1997) and then by Chen et al. (1999). Mesghouni et al. (1997) proposed "parallel jobs representation (PJsR)" to minimize the makespan . In the PJsR, the chromosome is a matrix where the number of rows equals the number of jobs. Each row consists of several genes equal to the number of job operations. Each gene has two elements of the machine selection and the starting time of the operation. Chen et al. (1999) proposed a two-segment chromosomal encoding that many researchers have used afterward, as we reviewed in section 3.4.1. Their chromosome encoding has two strings of chromosome A and chromosome B, where chromosome A defines the routing policy of the problem, and chromosome B defines the sequence of the operations on each machine.

GA has its own shortfalls like it is known that GA results are unstable Hamzaçebi (2008). It also can be shown that different random numbers will result in different final solutions that may be very close but not equal. To address this issue Caponetto et al. (2003) studied the effect of Random Number Generator (RNG) on GA performance. They have introduced using chaotic sequences instead of random ones. They have performed t-tests to show that some chaotic sequences always increase GA performance indexes compared to random sequences. Another algorithm called Improving GAs by

Random Search Technique (IGARSET) is suggested by Hamzaçebi (2008). IGRASET has two searching phases where the first phase is a GA global search, and the second phase is a local search around the found best solution. The local search produces a dx vector from [-α,α] range to adjust moving step sizes for each variable. The value of α becomes smaller by setting the new value at half of the previous value.

Oppacher and Wineberg (1999) have discussed that GA tends to get trapped at local maxima and has difficulty changing search space after convergence has occurred. This is due to the fact that when the GA population evolves, diversity reduces, and the same/similar chromosomes may be reevaluated over and over. To address these deficiencies, Oppacher and Wineberg (1999) have suggested a modified Shifting Balance Genetic Algorithm (SBT) and presented some experimental results showing that SBGA reduces premature convergence issues and improves GA performance under a dynamic environment. SBT divides the population into small, semi-isolated demes to give a better chance for the population to explore. As a result, the size of subpopulations that reach higher fitness will increase and send out more migrants than other subpopulations. So better gene combinations are gradually spread throughout the entire set of subpopulations through interim selection.

Shi et al. (2018) state that the GA population with higher diversity is more likely to escape from local optima. So increasing population diversity will be another method of reducing the likelihood of GA trapping in local optima and improving GA performance. Ho et al. (2007) proposed an architecture for integration between evolution and learning within a random search process of FJSP called LEarnable Genetic Architecture (LEGA). The authors presented many examples that through LEGA, the knowledge extracted from the previous generation by its schemata learning module is used to influence the diversity and quality of offspring. Another common way of improving GA diversity is Parallel GA (PGA) which uses different methods for evolving populations in parallel. PGA is less likely to be trapped in sub-optimal regions than regular GA, which uses a single population. Defersha and Chen (2010b) have proposed an island model PGA to solve a flexible job-shop scheduling problem incorporating sequence dependent

setup time, attached or detached setup time, machine release dates, and time lag requirements. Island-model is a common approach that uses multiple subpopulations that are separately evolving on several processors and periodically exchanging individuals. In addition to regular GA operators, PGA employs a migration operator that determines (1) the number of individuals undergoing migration, (2) the frequency of migration in numbers of generations, and (3) the migration policy directing the type of individuals (best, according to fitness, random, etc.) from one subpopulation to migrate to another. It also directs the type of individuals (worst, random, etc.) to be replaced. We have implemented PGA in our models presented in Chapter 3 and Chapter 4. Some other recent applications of GA for FJSP include Defersha and Bayat Movahed (2018), Lin and Zhang (2019), Zhang et al. (2020), Luo et al. (2020b), and Luo et al. (2020a).

2.5.1. Genetic Operators

The performance of genetic algorithms, to a great extent, depends on the performance of their genetic operators since it is the operators that are responsible for the whole evolution process. First, the selection operators select more fit chromosomes for the reproduction process. The selection operator can be applied in several different schemes, such as (i) Roulette Wheel, (ii) Ranked, (iii) Truncation, (iv) Boltzmann, (v) Elitism, and (vi) K-ways Tournament. Roulette Wheel Selection and Tournament Selection are the two more popular techniques widely used in the GA models for FJSPs. The roulette wheel selection is less popular than it was in the past due to issues like existing a "super individual" which may cause premature convergence (Zhang et al. (2011)). However, in the tournament selection, there is no arithmetical computation based on the fitness value. So "weaker" chromosomes have some chance to be selected, preventing the GA process from permutation. Jinghui Zhong et al. (2005) compared the performance of these two selection operators and concluded that tournament selection outperforms roulette wheel selection. This aligns with our finding about better performance of tournament selection presented in 4.4.3.

Also, as per Miller and Goldberg (1996), an ideal selection scheme should be

simple to code and efficient for both nonparallel and parallel architectures. Finally, it should be able to adjust its selection pressure to tune its performance for different domains. K-ways Tournament selection satisfies all of the above criteria (Miller and Goldberg, 1996), and consequently, we used it in all proposed genetic algorithms in this research. When this selection operator is applied, it holds a tournament competition among k randomly selected individuals. Then, the best individual from the tournament is declared the winner and inserted into the mating pool for the next generation. The contest is repeated until the mating pool reaches the desired population size. In this process, the selection pressure can be increased or decreased by increasing or decreasing the value of k. This adjustment is needed to strike a balance between exploitation (when k is large) and exploration (when k is small).

Once the selection process is completed, the individuals in the mating pool are randomly paired, and crossover operators are applied to each pair. Then, the resulting children undergo different mutation operators and finally constitute the population for the next generation. Unlike a selection operator, which can be applied similarly across problem domains, crossover and mutation operators need to be tailored to match the problem at hand and the solution representation adopted. As per Hong et al. (2000), each problem, even each stage of the genetic process in a single problem, requires appropriately defined crossover and mutation operators. Crossover operators select two chromosomes (parents) from the reproduction pool and generate two offspring, while mutation operators start with only one chromosome and transform it into one child. A.J. and P.D. (2015) classified crossover into three main categories of classical standard, binary, and application-dependant operators and introduced several types of crossover operators from each category. They conclude that each crossover operator has advantages and disadvantages for different types of problems. Hence they recommend reviewing similar solved problems and various crossover operators before selecting or creating a new crossover. In addition to crossover operator type, the probability rate these operators are applying is also important. A.J. and P.D. (2015) reported that most researchers

had set the value of crossover probability between 0.6 and 1 but finding the right probability depends on the type of crossover. The mutation operator is applied with a small probability of introducing more variability into the population and enhancing diversity. However, a large probability may destroy the good chromosomes (Zhang et al. (2011)). High probability rates of crossover and mutation increase diversity and recombination opportunity, whereas low rates improve quality by less disrupting good combination (Mansouri (2005)). Researchers have used many techniques, including ANOVA and Dynamic selection, to set the right value for GA operators. Akgündüz and Tunalı (2011) classify these techniques into two major forms "Parameter Tuning" and "Parameter Control" as described below.

- Parameter Tuning: This is about finding good parameter values (e.g., crossover and mutation probabilities, population size, etc.) before running the algorithm and keeping them unchanged during running GA. Parameter tuning has been widely adopted by researchers. Since GA parameters often interact in a complex way, tuning one parameter at a time may cause suboptimal issues. However, the simultaneous tuning of more parameters needs an enormous amount of experiments. So the best configurations of GA parameters can be determined through several experiments prior to the run of the algorithm. Design of Experiments (DOE) and Analysis of Variance (ANOVA) are common methods of parameter tuning that have been used by Kundakcı and Kulak (2016) and Essafi et al. (2008) among many others.

- Parameter Control: In this method, GA starts with initial parameter values, which will change through different mechanisms during the GA run:

  - Deterministic Parameter Control: Changing GA parameters through some sort of deterministic rules without using any feedback (e.g., a time-varying schedule)

  - Adaptive Parameter Control (APC): Some form of feedback is used to change

GA parameters. Akgündüz and Tunalı (2010) employed APC for a multi-objective GA that modifies GA parameter values for every G number of iterations based on the GA performance of the last G iterations. Their experimental results show that the proposed adaptive GA outperformed non-adaptive algorithms.

– Self-adaptive Parameter Control: This is a more advanced technique that GA parameters are part of the GA chromosomes and will be optimized through GA evolutions. So parameter values that generate better individuals have a higher survival chance and produce offspring.

In terms of GA for FJSP, the crossover and mutation operators can be categorized as assignment and sequencing operators. Assignment operators change the machine assignment of the chromosomes either through exchanging the assignment of selected operations between two parents (in a crossover operator) or altering the assignment of one or a few operations of one individual chromosome (in a mutation operator). This is similar for sequencing operators, except here, job sequencing of chromosomes is changed considering precedence constraints among operations of the same job either as a checkpoint built in the actual operator or done by a repairing mechanism afterward. One way or another, according to the precedence constraint, any operation cannot start unless all previous (precedence) operations of the same job have been completed. There are several types of crossover and mutation operators have been developed for FJSP, such as "Precedence Preserving Order-based Crossover (POX)", "Precedence Preserving Shift Mutation (PPS)", "Position Based Mutation (PBM)", "Machine Based Mutation (MBM)", "Operations Swapping Mutation (OSM)", and "Assignment Altering Mutation (AAM)". We have modified and developed several GA operators that will be described in chapters 3, 4, and 5. Regarding selection operators, as discussed, K-ways Tournament has been utilized in all proposed genetic algorithms in this research due to its broad advantages over the other popular techniques and the result of the empirical analysis done in section 4.4.3.

2.6. Two-stage GA for FJSP

As we mentioned, GA has been used extensively in FJSP as a stand-alone or primary algorithm in hybrid approaches. Each researcher has used a different technique to improve the GA performance. Shi et al. (2018) have classified hundreds of the GA improvement strategies for solving FJSP into five basic categories and compared their performance using some proposed algorithms and benchmark problems. The five categories are "discrete", "multi-population", "mixed", "parallel" and "multistage" structures. The discrete structure uses some sort of discretization method. The multi-population structure has more than one population to improve the population's diversity and avoid premature convergence. A mixed or hybrid structure utilizes operators of one technique or its main idea in another technique. A parallel structure has two or more different populations corresponding to two or more different techniques in a newly obtained algorithm. This differs from multi-population, which uses only one technique to evolve multiple populations. In a multistage structure, there are two or more populations that are evolving one after another. Then Shi et al. (2018) ran several sample problems and concluded that hybridization of GA with other techniques is the best strategy among the five studied structures, followed by the multistage structure, and suggested these two should be the first strategies to consider for improving GA performance in solving FJSP. They also pointed out that, unlike hybridization, which is the most popular improvement GA strategy, there are only a few papers with multistage structures despite its effectiveness.

One of the first studies that presented a multistage GA for FJSP is Zhang and Gen (2005). Their multistage model, in fact, is a way of simplifying solution representation to reduce completion time calculation. It considers the total number of operations for all jobs as K stages and the total number of machines as M states in a multistage solution representation. A novel Two-Stage GA has been developed by Wang et al. (2008) that in the first stage uses an optimal computing budget allocation method to find the fittest GA parameters (number of population, probability of crossover, and mutation) for the given JSP. Then in the second stage, a regular GA uses those optimized parameters to

solve the FJSP. They solved some benchmark problems to illustrate the efficiency of their model. Ma et al. (2009) also utilized similar multistage encoding to Zhang and Gen (2005) and proposed a hybrid of GA with bottleneck shifting. Bottleneck shifting interchanges operation sequences and machine assignment of operations on the critical path to improve the makespan. Their solution encoding has two vectors representing each solution candidate. One vector is for initialization and mutation, and the other is for crossover operation. Al-Hinai and ElMekkawy (2011) presented a Two-Stage GA to provide a robust scheduling algorithm for FJSP in the presence of random machine breakdown. In the first stage, they do not consider any breakdown. They have also used a similar hybrid GA introduced in Al-Hinai and Elmekkawy (2011) and are utilizing a special initial population method (Ini-PopGen). In this approach, half of the initial population is formed randomly, but another half follows the Ini-PopGen technique. At Ini-PopGen, the machine that can complete the job in the shortest time, considering the processing time and its current workload, will be assigned to process the operation. In this approach, jobs are ordered randomly, but the first operation of all jobs in that order will be complete (i.e., o_1 of j_3 then o_1 of j_1 then o_1 of j_2) before going to the next operations (i.e., o_2 of j_3 then o_2 of j_1 then o_2 of j_2). The first stage provides a high-quality population to be fed into the second stage. The second stage considers random machine breakdown with a bi-objective objective of makespan minimization with a robust evaluation function that combines three stability measures. Another interesting application of Two-Stage GA for solving multi-objective JSPs has been developed by Kachitvichyanukul and Sitthitham (2011). At stage one, parallel GAs with independent populations and allowing migration among populations evolve to produce high-quality solutions for each objective function. These populations are combined in stage two and evolve using the weighted aggregating objective function. They have considered a JSP with three objectives: minimizing makespan, minimizing total weighted earliness, and minimizing total weighted tardiness, and used benchmark problems to demonstrate the algorithm's effectiveness. A recent multistage GA for FJSP has been developed by Shi et al. (2018), which in the first stage uses a discrete IWO (Invasive Weed Optimization)

approach, then better individuals will be fed as the initial population of the second stage GA.

The quality of the initial population of an evolutionary algorithm is a crucial factor that affects its ability to find good solutions with less computational time (Bajer et al., 2016; Rahnamayan et al., 2007). If the initial population is constituted of good individuals, it may lead the search toward promising regions of the problem space from the get-go. Kacem et al. (2001) created a method referred to as Approach by Localization (AL) to generate a promising initial population in solving FJSP using evolutionary optimization. A variant of the AL method was later proposed in Pezzella et al. (2008). The approach was then extended in Defersha and Chen (2010b) to account for sequence dependent setup time and machine release date. Considering this fact to create a high-quality initial population, in chapter 3, we describe our highly efficient Two-Stage GA for solving FJSP, that the first stage generates a high-quality population that will be fed as the initial population into the second stage. The first stage of genetic search is different from the described common approach of GA for FJSP, which determines both operation sequencing and machine assignment through genetic search. This stage has a solution encoding that only dictates the sequence in which operations are considered for assignment. For the assignment problem, machines are assigned through an evaluation (decoding) process that starts from the first operation in the GA coding and finds machines that can complete each operation the soonest by considering process time and also operations that are already assigned to the machine. Then the second stage starts from the high-quality population created by the first stage and follows the common approach of the genetic algorithm for FJSP to enable searching the entire solution space and includes solutions that might have been excluded because of the greedy nature of the first phase.

The algorithm has been successfully tested on benchmark FJSP problems and also on more complicated random generated problems that include attached or detached sequence dependent setup time, process lag time, and machine release date in order to demonstrate the performance improvement better. In order to do so, we developed

a random problem generator code to produce several FJSPs incorporating attached or detached sequence-dependent setup, machine release dates, and time lag requirements as per the model presented by Defersha and Chen (2010b). We solved these more complex problems with both proposed Two-Stage GA and regular GA to illustrate how good the performance of the proposed algorithm is in solving more realistic and complex FJSPs compared to regular GA.

As was discussed earlier, in addition to multi-staging, Shi et al. (2018) listed multi-population and parallel as GA improvement strategies. However, they concluded that multi-staging outperforms multi-population and parallel. We also came to the same result as shown in chapters 3 and 4, we improved the performance of the proposed Two-Stage GA by using high-performance parallel computation. However, we found that the sequential version of the Two-Stage GA (using a single CPU) outperformed a parallel implementation of the regular genetic algorithm that uses many CPUs.

Chapter 3

Two-Stage GA for FJSP

3.1. Introduction

As we discussed in chapter 1 the solution encoding of many regular genetic algorithms (RGAs) developed in the literature for flexible job-shop scheduling problem (FJSP) has two parts: one part encodes the assignment decision and the other the sequencing decision. The genetic search determines both the assignment and the sequencing of the operations simultaneously through a random process guided by the principles of natural selection and evolution. In this research, we develop a Two-Stage Genetic Algorithm (2SGA) with the first stage being different from typical RGAs. The first stage of 2SGA has a solution encoding that only dictates the sequence in which the operations are considered for assignment. Whenever an operation is considered for the assignment, the machine that can complete this operation the soonest is selected while taking into account its process time and all operations that are already assigned to this machine. The order in which the operations are assigned to machines determines their sequence. The second stage, starting from the solutions of the first stage, follows the common approach of RGAs for FJSP to enable the algorithm to search the entire solution space by including solutions that might have been excluded because of the greedy nature of the first stage. Initially, we developed the Two-Stage Genetic Algorithm for a regular FJSP and tested the proposed algorithm by solving many benchmark problems and several

large-size problems that is published at Rooyani and Defersha (2019).

This observed high performance of the algorithm is attributed to some of its distinct features. One of these features is its inherent ability to create a high-quality initial population. The quality of the initial population that can be created by the proposed solution encoding/decoding is by far better than the one that can be generated using a specialized initialization technique that appeared in Kacem et al. (2001) and later used by many researchers (e.g., Tang et al. (2011); Pezzella et al. (2008); Defersha and Chen (2010b)). This feature enables the algorithm to find highly improved solutions from the get-go of the search. The greedy nature of the first stage of the search accelerates convergence and shortens the computational time required to reach reasonable solutions by many folds. The second stage further improves the solution of the first stage. With these features, the proposed algorithm outperforms the existing approach both in terms of convergence speed and final solution quality.

Further, we applied the Two-Stage GA on a comprehensive FJSP model with sequence-dependent setup, machine release date, and lag time between operations. We generated several examples and run them to illustrate the performance of the proposed two-stage algorithm greatly exceeds that of the common approach of genetic algorithm for FJSP. We evaluated this improved performance in several ways. Many published articles assessed the performance of their proposed algorithms by solving selected benchmark problems (e.g., Ishikawa et al. (2015); Li and Gao (2016); Shen et al. (2018)). However, such evaluations alone may be insufficient as the benchmark problems are usually small in size and do not pose the computational challenge often encountered when solving large-size problems. There is also no guarantee that methodologies that perform very well in small-size problems would replicate their performance superiority when solving large-size problems. Moreover, in industrial settings, algorithms that arrive at reasonable solutions very rapidly may be preferred over those which achieve the same or better solutions at the expense of excessive computational time and cost. To this end, in addition to using benchmark problems in both our papers, we demonstrated the performance superiority of the proposed algorithm in solving large-size problems having

up to 140 jobs and 80 machines. We also show that the performance of the proposed algorithm can be further improved using High-performance Parallel Computation (HPC). However, the more interesting result we found was that the sequential version of the proposed algorithm (using a single CPU) outperformed a parallel implementation of the regular genetic algorithm that uses many CPUs. The results of this study is published in Defersha and Rooyani (2020).

In this chapter, we present the finding and results of both papers organized as follows. In Section 3.2 the variant of the FJSP problem for which we proposed our algorithm is described in detail. The various components of the proposed Two-Stage GA are presented in Section 3.3. Section 3.4 provides several numerical studies for both classic FJSP and more comprehensive problems. Discussion and conclusion are in Sections 3.5.

3.2. Problem Description

Many mathematical models appeared in the literature for FJSP problems. The purpose of our model is to develop a new Two-Stage Genetic Algorithm (2SGA) that can be tailored to efficiently solve many of those mathematical models. As it was mentioned, we initially developed and tested the Two-Stage GA on a classic FJSP and published it at Rooyani and Defersha (2019). To describe a classic FJSP, consider a job shop having M machines ($m = 1, 2, ..., M$) and a total number of J ($j = 1, 2, ..., J$) independent jobs which are to be scheduled. Job j has O_j number of operations to be processed in a fixed order which are indexed as $o = 1, 2, \ldots, O_j$. Operation o of job j has a set of eligible machines. This feature is represented by a binary data $P_{o,j,m}$ which is equal to 1 if machine m can process operation o of job j, 0 otherwise. Among a set of eligible machines for an operation, only one will be selected for its processing. If machine m is selected for processing operation o of job j, this operation will be processed on this machine with a processing time $B_j \times T_{o,j,m}$ where B_j is the batch size and $T_{o,j,m}$ is the unit processing time. R_m denotes the maximum number of production runs of

machine m where production runs are indexed by $r = 1, 2, ..., R_m$. Theoretically, R_m is equal to the total number of operations that can be assigned to machine m and can be increased as schedule demands. Runs are assigned to operations in sequence and each operation is assigned to exactly one production run. Thus, the assignment of operations to production runs determines both their assignments and sequences. A machine can process only one operation at a time (at each run). An operation cannot be preempted and all the machines and jobs are ready to process at time zero. This makes a classic FJSP that we solved and demonstrated the superiority of the Two-Stage Genetic Algorithm at Rooyani and Defersha (2019).

Furthermore, in Defersha and Rooyani (2020) we tested and illustrated the performance of the Two-Stage GA on more complicated FJSPs. We considered the model that appeared in Defersha and Chen (2010b) because of its comprehensive nature. The model incorporates sequence-dependent setup time, attached/detached nature of setups, machine release date, and operation lag time. In this FJSP, the machine release date is coded by D_m and is the time when machine m will become available for processing jobs for the current schedule. Each operation has a lag time of $L_{o,j}$ (a delay for cooling, drying, inspection, etc.) from the completion time of operation $o-1$ of the same job j, and after the required setup is completed. A setup can be attached (non-anticipatory or inseparable) or detached (anticipatory or separable). An attached setup cannot be performed before the arrival of the job on the machine. Whereas, a detached setup can be performed before the job arrives. We should note both form of setups will take the machine time and should be considered in the time between two consequent machine runs. The nature of a setup being attached or detached is represented by a binary data $A_{o,j}$. This binary data is equal to 1 if the setup of operation o of job j is attached and 0 if it is detached. Setup time is sequence dependent and denoted by $S_{o,j,m,o',j'}$ where operation o' of job j' is operation processed on machine m immediately before operation o of job j. If operation o of job j is the first operation to be processed on machine m, the setup time is represented by $S^*_{o,j,m}$. The setup time $S_{o,j,m,o',j'}$ (or $S^*_{o,j,m}$) can be overlapped with the processing time of operation $o-1$ of job j if the setup is detached and

machine m is available for setup. Given the above problem, the task is to assign each operation to a production run on one of its eligible machines and to determine the start and completion time of each production run. The completion time of the production run r on machine m is represented by $\widehat{c}_{r,m}$. Whereas, the completion time of an operation o of job j on an eligible machine m is denoted as $c_{o,j,m}$. If an operation o of job j is assigned to the r^{th} run of machine m, the variable $c_{o,j,m}$ has the same value as $\widehat{c}_{r,m}$. The objective is to minimize the makespan of the schedule, which is the largest completion time. 3.2.1 describes the Mixed Integer Linear Programming (MILP) formulation of this problem as it is reported in Defersha and Chen (2010b). Also, all the model parameters and variables are listed in List of Acronyms Section. To facilitate the discussion about solution encoding and decoding procedures in the following sections, a data set for a small problem instance is given in Tables 3.1 and 3.2. The problem has 5 jobs to be processed using 4 machines. The release dates for machine-1 and machine-4 are assumed to be $D_1 = 840$ and $D_4 = 120$ minutes, respectively. Machines 2 and 3 are available at time zero. We refer to this problem as Problem-1 in the later sections of this chapter.

Table 3.1: Data for Jobs for Problem-1

					(Eligible machine, Processing Time) = (m, $T_{o,j,m}$)			
j	B_j	o	$A_{o,j}$	$L_{o,j}$	i	ii	iii	iv
1	45	1	NA	NA	(1, 6.75)	(2, 6.25)		
		2	1	0	(1, 4.25)	(4, 4.50)		
		3	1	0	(1, 1.50)	(2, 1.25)	(3, 1.75)	(4, 1.50)
		4	0	40	(3, 2.00)	(4, 1.25)		
2	35	1	NA	na	(1, 5.00)	(4, 5.00)		
		2	1	0	(1, 6.75)	(2, 6.50)	(3, 6.75)	(4, 6.50)
		3	1	0	(1, 6.00)	(2, 5.25)		
		4	1	40	(1, 3.25)	(4, 3.75)		
3	40	1	NA	NA	(1, 7.00)	(2, 6.25)	(3, 6.25)	(4, 6.25)
		2	1	0	(1, 4.00)	(2, 4.00)		
		3	0	40	(1, 5.50)	(3, 5.50)		
4	30	1	NA	NA	(1, 3.75)	(3, 3.25)	(4, 3.25)	
		2	0	0	(1, 6.25)	(4, 6.75)		
5	50	1	NA	NA	(2, 3.25)	(4, 3.75)		
		2	1	0	(1, 2.50)	(2, 2.50)	(4, 2.75)	
		3	0	0	(1, 4.50)	(2, 4.75)	(3, 4.25)	(4, 4.50)

NA = Not Applicable

Machine release dates in minutes: $D_1 = 840$, $D_2 = D_3 = 0$, $D_4 = 120$.

Table 3.2: Sequence Dependent Setup Time Data

			Setup Time $S^*_{o,j,m}$ and $S_{o,j,m,o',j'}$	
j	o	m	($S^*_{o,j,m}$)	$\cdots(j', o', S_{o,j,m,o',j'}) \cdots$
1	1	1	(60)	(1,1,10)(1,2,40)(1,3,60)(2,1,90)(2,2,90)(2,3,120)(2,4,90)(3,1,120)(3,2,120)(3,3,120)(4,1,90)(4,2,120)(5,2,120)(5,3,90)
		2	(80)	(1,1,20)(1,3,40)(2,2,120)(2,3,90)(3,1,120)(3,2,120)(5,1,90)(5,2,90)(5,3,120)
	2	1	(60)	(1,1,60)(1,2,10)(1,3,60)(2,1,90)(2,2,120)(2,3,120)(2,4,120)(3,1,120)(3,2,90)(3,3,90)(4,1,90)(4,2,120)(5,2,90)(5,3,90)
		4	(60)	(1,2,20)(1,3,60)(1,4,60)(2,1,90)(2,2,120)(2,4,90)(3,1,90)(4,1,90)(4,2,120)(5,1,90)(5,2,90)(5,3,120)
	3	1	(60)	(1,1,60)(1,2,40)(1,3,20)(2,1,90)(2,2,120)(2,3,90)(2,4,90)(3,1,90)(3,2,120)(3,3,120)(4,1,90)(4,2,90)(5,2,90)(5,3,90)
		2	(80)	(1,1,40)(1,3,20)(2,2,120)(2,3,120)(3,1,90)(3,2,120)(5,1,90)(5,2,120)(5,3,90)
		3	(60)	(1,3,20)(1,4,60)(2,2,120)(3,1,120)(3,3,90)(4,1,90)(5,3,120)
		4	(40)	(1,2,40)(1,3,20)(1,4,60)(2,1,90)(2,2,120)(2,4,90)(3,1,90)(4,1,120)(4,2,90)(5,1,120)(5,2,90)(5,3,90)
	4	3	(80)	(1,3,40)(1,4,10)(2,2,120)(3,1,90)(3,3,120)(4,1,90)(5,3,90)
		4	(80)	(1,2,40)(1,3,60)(1,4,20)(2,1,90)(2,2,90)(2,4,90)(3,1,90)(4,1,90)(4,2,90)(5,1,90)(5,2,90)(5,3,90)
2	1	1	(60)	(1,1,90)(1,2,90)(1,3,90)(2,1,10)(2,2,40)(2,3,60)(2,4,60)(3,1,120)(3,2,90)(3,3,120)(4,1,120)(4,2,90)(5,2,120)(5,3,120)
		4	(80)	(1,2,120)(1,3,90)(1,4,90)(2,1,20)(2,2,60)(2,4,60)(3,1,120)(4,1,90)(4,2,120)(5,1,90)(5,2,120)(5,3,120)
	2	1	(40)	(1,1,90)(1,2,120)(1,3,120)(2,1,40)(2,2,10)(2,3,60)(2,4,60)(3,1,120)(3,2,120)(3,3,120)(4,1,120)(4,2,90)(5,2,90)(5,3,120)
		2	(80)	(1,1,120)(1,3,90)(2,2,10)(2,3,40)(3,1,120)(3,2,90)(5,1,120)(5,2,90)(5,3,90)
		3	(80)	(1,3,120)(1,4,120)(2,2,10)(3,1,120)(3,3,90)(4,1,120)(5,3,120)
		4	(40)	(1,2,90)(1,3,120)(1,4,120)(2,1,40)(2,2,10)(2,4,40)(3,1,90)(4,1,90)(4,2,120)(5,1,120)(5,2,90)(5,3,90)
	3	1	(80)	(1,1,90)(1,2,120)(1,3,90)(2,1,40)(2,2,40)(2,3,10)(2,4,60)(3,1,120)(3,2,90)(3,3,90)(4,1,120)(4,2,90)(5,2,90)(5,3,90)
		2	(80)	(1,1,120)(1,3,120)(2,2,60)(2,3,20)(3,1,90)(3,2,120)(5,1,90)(5,2,120)(5,3,90)
	4	1	(40)	(1,1,90)(1,2,90)(1,3,120)(2,1,60)(2,2,60)(2,3,40)(2,4,20)(3,1,120)(3,2,90)(3,3,90)(4,1,120)(4,2,120)(5,2,90)(5,3,90)
		4	(60)	(1,2,90)(1,3,120)(1,4,90)(2,1,40)(2,2,60)(2,4,10)(3,1,90)(4,1,120)(4,2,120)(5,1,90)(5,2,90)(5,3,120)
3	1	1	(40)	(1,1,90)(1,2,90)(1,3,90)(2,1,90)(2,2,90)(2,3,120)(2,4,120)(3,1,10)(3,2,60)(3,3,40)(4,1,120)(4,2,90)(5,2,120)(5,3,120)
		2	(80)	(1,1,90)(1,3,120)(2,2,120)(2,3,90)(3,1,10)(3,2,60)(5,1,120)(5,2,120)(5,3,90)
		3	(80)	(1,3,120)(1,4,90)(2,2,90)(3,1,10)(3,3,40)(4,1,90)(5,3,90)
		4	(60)	(1,2,120)(1,3,120)(1,4,90)(2,1,120)(2,2,120)(2,4,120)(3,1,20)(4,1,120)(4,2,90)(5,1,120)(5,2,120)(5,3,90)
	2	1	(60)	(1,1,120)(1,2,120)(1,3,120)(2,1,90)(2,2,120)(2,3,120)(2,4,120)(3,1,40)(3,2,20)(3,3,60)(4,1,90)(4,2,90)(5,2,90)(5,3,90)
		2	(60)	(1,1,90)(1,3,120)(2,2,120)(2,3,120)(3,1,60)(3,2,20)(5,1,120)(5,2,120)(5,3,90)
	3	1	(60)	(1,1,90)(1,2,90)(1,3,90)(2,1,120)(2,2,120)(2,3,90)(2,4,120)(3,1,60)(3,2,60)(3,3,10)(4,1,90)(4,2,120)(5,2,90)(5,3,120)
		3	(80)	(1,3,90)(1,4,120)(2,2,90)(3,1,60)(3,3,10)(4,1,120)(5,3,120)
4	1	1	(60)	(1,1,90)(1,2,120)(1,3,90)(2,1,90)(2,2,90)(2,3,90)(2,4,120)(3,1,120)(3,2,90)(3,3,120)(4,1,10)(4,2,40)(5,2,120)(5,3,90)
		3	(80)	(1,3,120)(1,4,90)(2,2,120)(3,1,90)(3,3,90)(4,1,20)(5,3,90)
		4	(40)	(1,2,90)(1,3,120)(1,4,90)(2,1,120)(2,2,90)(2,4,120)(3,1,90)(4,1,10)(4,2,60)(5,1,90)(5,2,90)(5,3,90)
	2	1	(80)	(1,1,120)(1,2,120)(1,3,120)(2,1,90)(2,2,120)(2,3,90)(2,4,90)(3,1,90)(3,2,90)(3,3,120)(4,1,40)(4,2,10)(5,2,90)(5,3,120)
		4	(80)	(1,2,90)(1,3,120)(1,4,90)(2,1,90)(2,2,120)(2,4,90)(3,1,120)(4,1,60)(4,2,20)(5,1,90)(5,2,90)(5,3,120)
5	1	2	(60)	(1,1,90)(1,3,120)(2,2,120)(2,3,90)(3,1,120)(3,2,120)(5,1,10)(5,2,40)(5,3,40)
		4	(40)	(1,2,90)(1,3,90)(1,4,120)(2,1,120)(2,2,120)(2,4,90)(3,1,90)(4,1,90)(4,2,90)(5,1,20)(5,2,60)(5,3,40)
	2	1	(40)	(1,1,90)(1,2,120)(1,3,90)(2,1,90)(2,2,120)(2,3,90)(2,4,90)(3,1,120)(3,2,120)(3,3,90)(4,1,120)(4,2,120)(5,2,10)(5,3,60)
		2	(40)	(1,1,120)(1,3,90)(2,2,120)(2,3,90)(3,1,120)(3,2,120)(5,1,40)(5,2,20)(5,3,60)
		4	(60)	(1,2,120)(1,3,90)(1,4,90)(2,1,120)(2,2,120)(2,4,90)(3,1,120)(4,1,120)(4,2,90)(5,1,60)(5,2,10)(5,3,40)
	3	1	(80)	(1,1,90)(1,2,120)(1,3,90)(2,1,90)(2,2,90)(2,3,90)(2,4,90)(3,1,120)(3,2,90)(3,3,120)(4,1,90)(4,2,120)(5,2,60)(5,3,10)
		2	(40)	(1,1,90)(1,3,90)(2,2,90)(2,3,90)(3,1,120)(3,2,120)(5,1,40)(5,2,60)(5,3,20)
		3	(80)	(1,3,90)(1,4,90)(2,2,90)(3,1,120)(3,3,120)(4,1,120)(5,3,10)
		4	(60)	(1,2,120)(1,3,120)(1,4,90)(2,1,90)(2,2,90)(2,4,90)(3,1,120)(4,1,120)(4,2,120)(5,1,60)(5,2,60)(5,3,20)

$S^*_{o,j,m}$ is the setup time for o of job j on machine m if this operation is the first operation on this machine.

$S_{o,j,m,o',j'}$ is the setup time for operation o of job j on machine m if operation o' of job j' is the last operation on machine m.

3.2.1. Mathematical Model

Model Parameters and Variables:

Following the problem description in section 3.2, below is the MILP model for the FJSP that we used to test the Two-Stage Genetic Algorithm performance in Defersha and Rooyani (2020) as it was originally developed by Defersha and Chen (2010b):

Indexes and Input Data:

B_j Batch size of job j

$T_{m,j,o}$ Unit process time of operation o on machine m

D_m Release date of machine m.

R_m Maximum number of production runs of machine m where production runs are indexed by $r = 1, 2, ..., Rm$.

$S^*_{o,j,m}$ Setup time of operation o of job j on machine m if operation o is the first operation to be processed on machine m

$S_{o,j,m,o',j'}$ Setup time of operation o of job j on machine m where operation o' of job j' is operation processed immediately before

Ω Large positive number.

Variables:

Continuous Variables:

C_{max} Makespan of the schedule.

$\hat{c}_{m,r}$ Completion time of run r of machine m.

$c_{o,j,m}$ Completion time of operation o of job j on an eligible machine m.

Binary Variables:

$A_{o,j}$ A binary data equal to 1 if the setup of operation o of job j is attached (non-anticipatory), or 0 if this setup is detached (anticipatory).

$x_{m,r,j,o}$ Binary variable that takes the value 1 if the run r on machine m is assigned to operation o of job j, 0 otherwise.

$z_{m,r}$ Binary variable that is equal to 1 if run r of machine m has been assigned to any operation, 0 otherwise.

Objective Minimize the makespan:

$$Z = c_{max} \tag{3.1}$$

Subject to:

$$c_{max} \geq c_{o,j,m}; \quad \forall(o,j,m) \tag{3.2}$$

$$\hat{c}_{r,m} \geq c_{o,j,m} + \Omega \cdot x_{r,m,o,j} - \Omega; \ \ \forall(r,m,o,j) \tag{3.3}$$

$$\hat{c}_{m,r} \leq c_{o,j,m} - \Omega \cdot x_{r,m,o,j} + \Omega; \ \ \forall(r,m,o,j) \tag{3.4}$$

$$\hat{c}_{1,m} - B_j \cdot T_{o,j,m} - S^*_{o,j,m} - \Omega \cdot \ x_{1,m,o,j} + \Omega \geq D_m; \ \ \forall(m,o,j) \tag{3.5}$$

$$\hat{c}_{r,m} - B_j \cdot T_{o,j,m} - S_{o,j,m,o',j'} - \Omega \cdot (x_{r,m,o,j} + x_{r-1,m,o',j'}) + 2\Omega \geq \hat{c}_{r-1,m};$$

$$\forall(r,m,o,j,o',j') | (r > 1) \,\&\, ((o,j) \neq (o',j')) \tag{3.6}$$

$$\hat{c}_{1,m} - B_j \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \Omega \cdot (x_{1,m,o,j} + x_{r',m',o-1,j}) + 2\Omega \geq \hat{c}_{r',m'} + L_{o,j};$$

$$\forall(m,r',m',o,j) | ((1,m) \neq (r',m')) \,\&\, (o > 1) \tag{3.7}$$

$$\hat{c}_{r,m} - B_j \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \Omega \cdot (x_{r-1,m,o',j'} + x_{r,m,o,j} + x_{r',m',o-1,j})$$

$$+3\Omega \geq \hat{c}_{r',m'} + L_{j,o};$$

$$\forall (r, m, r', m', o, j, o', j') | (r > 1) \,\&\, (o > 1) \,\&\, ((r, m) \neq (r', m')) \,\&\, ((o, j) \neq (o', j')) \quad (3.8)$$

$$x_{r,m,o,j} \leq P_{o,j,m}; \;\; \forall (r, m, o, j) \quad (3.9)$$

$$\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,j} = 1; \;\; \forall (o, j) \quad (3.10)$$

$$\sum_{j=1}^{J} \sum_{o=1}^{O_j} x_{r,m,o,j} = z_{r,m}; \;\; \forall (r, m) \quad (3.11)$$

$$z_{r+1,m} \leq z_{r,m}; \;\; \forall (r, m) \quad (3.12)$$

$$x_{r',m,o',j} \leq 1 - x_{r,m,o,j}; \;\; \forall (r, r', m, o, j, o') | (o' > o) \,\&\, (r' < r) \quad (3.13)$$

$$x_{r',m,o',j} \leq 1 - x_{r,m,o,j}; \;\; \forall (r, r', m, o, j, o') | (o' < o) \,\&\, (r' > r) \quad (3.14)$$

$$x_{r,m,o,j} \textit{ and } z_{r,m} \textit{ are binary} \quad (3.15)$$

The objective function of this model (Eq. (3.1)) is minimizing the makespan that is calculated through Eq. (3.2). Eqs. (3.3) and (3.4) calculate the completion time of each step of the production process ($c_{o,j,m}$) that is equal to the completion time of the specific run of machine m that the job is assigned. Eqs. (3.5) and (3.6) determine the completion time of the first run ($\hat{c}_{1,m}$) and subsequent runs of the machine ($\hat{c}_{r,m}$) based on its previous run. Eqs. (3.5) and (3.6) considered setup time regardless of its detached or attached nature since both types take time from the machine between two consequent runs. While we only consider the attached setup time ($A_{j,o}$ =1) when we calculate the

completion time of each run of the machine based on the previous operation of the job through Eqs. (3.7) and (3.8). Eqs. (3.9) and (3.10) assure that operation o of job j has been assigned to one run of the capable machine m ($P_{o,j,m} = 1$). Eqs. (3.11) and (3.12) assure only one operation, at most, is assigned to run r of machine m only if the previous run of that machine is already assigned. Eqs. (3.13) and (3.14) secure the feasibility of machine assignment by stating if two operations of a specific job are assigned to a specific machine, the earlier operation is assigned to an earlier run and not the other way.

3.3. Two-Stage Genetic Algorithm (2SGA)

Since its introduction in 1970 by Taylor (1994), genetic algorithm has gained sprawling popularity in solving a wide range of complex problems in different disciplines. A significantly large number of researchers have also applied genetic algorithms to solve very diverse problems in manufacturing systems design and operation. It has also been widely preferred by researchers to solve FJSP problems. In a classical job shop scheduling problem, an operation of a job can only be assigned to a designated machine. Given the assignments of operations to machines, the problem is to determine the sequences of the operations on each machine. In a flexible job-shop scheduling problem (FJSP), on the other hand, an operation can be assigned to one of a set of eligible machines. The problem is, therefore, to simultaneously determine both the assignment of operations to machines and their sequences. The common approach to the genetic search is to determine both the assignment and the sequencing of the operations simultaneously through a random process guided by the principles of natural selection and evolution. We develop a Two-Stage Genetic Algorithm with the first stage having solution encoding and decoding different from the common approaches discussed in the previous section. In our proposed Two-Stage Genetic Algorithm approach, the first stage encoding only dictates the sequence in which the operations are considered for assignment. While in the second stage, starting from the solutions of the first stage, the encoding follows the

common approach of the genetic algorithm for FJSP and the genetic search determines both the assignment and the sequencing of the operations simultaneously.

3.3.1. Solution Encoding and Decoding

Solution encoding is the kernel of a GA to solve a problem at hand. Gao et al. (2006) stated that primary determinants of a GA's success or failure are the coding by which its genotypes represent solutions and the interaction of the coding with the GA's recombination and mutation operators. In the past several decades, three closely interrelated solution encodings have been widely used in literature to solve many FJSP variants. In the following subsections, we present those commonly used solution encodings, and the modifications we propose to accelerate the genetic algorithm by dividing the search into two stages.

As we discussed in a classical job shop scheduling problem, an operation of a job can only be assigned to a designated machine, so we have only determined the sequences of the operations on each respective machine (sequencing problem). In a flexible job-shop scheduling problem, on the other hand, operations should be assigned to one eligible machine out of a set (assignment problem) prior to the sequencing problem. Accordingly, the solution encodings of many genetic algorithms developed in literature have two parts: one part encodes the assignment decision and the other the sequencing decision. So the genetic search can determine both assignment and sequencing of the operations simultaneously through a random process guided by the principles of natural selection and evolution. This solution encoding has been proposed by one of the early studies of GA for FJSP by Chen et al. (1999). Figure 3.1 depicts three interrelated solution representations that appeared in many articles, reporting the use of a genetic algorithm for FJSP. These solution representations correspond to an arbitrarily generated solution for Problem-1 which has a total of 16 operations.

The chromosomal encoding in Figure 3.1-(a) has two segments. The number of genes in each segment is equal to the total number of operations. The left-hand-side segment (LHS-Segment) has genes that are arranged in the natural order of the jobs

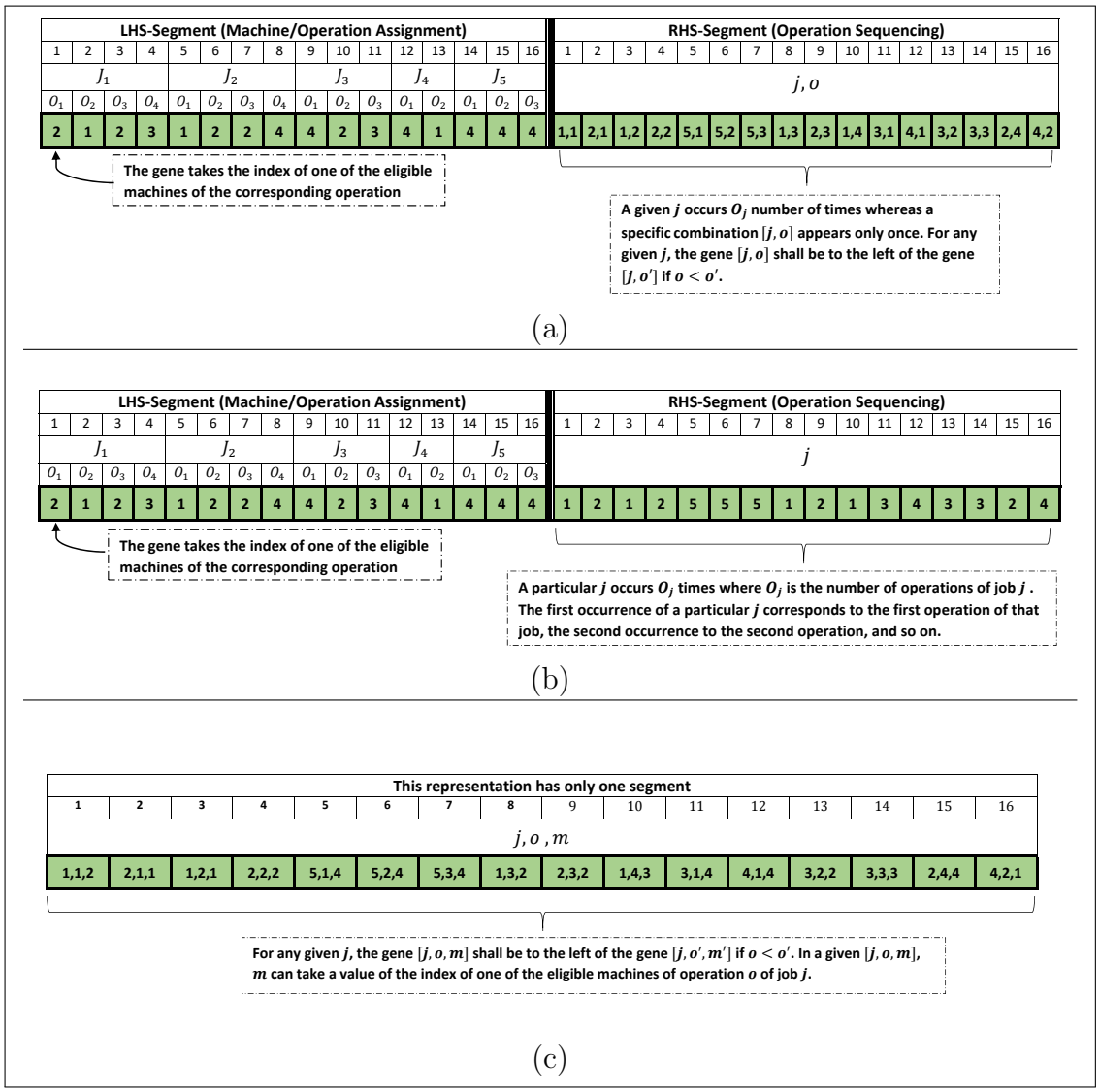


Figure 3.1: Three commonly used solution representations in applying GA for FJSP

and operations. Each gene corresponds to an operation and can take a value equal to the index of one of the eligible machines of the corresponding operation. In effect, the LHS-Segment encodes the machine-to-operation assignments of a particular solution of the FJSP at hand. The decision of sequencing the operations on each machine is encoded in the right-hand-side segment (RHS-Segment). In this segment, each gene takes a pair of values (j, o). A particular j occurs O_j number of times where a specific combination (j, o) appears only once. For any given j, the gene with a value (j, o) shall be located to the left of the genes with values (j, o') if $o < o'$. If two operations are assigned on

the same machine by the LHS-Segment, the operation that occurs earlier in the RHS-Segment takes precedence over the one that occurs later. Representations that have similar structural interpretations as the one discussed above can be found in Lei (2012) Chaudhry et al. (2013), Demir and Işleyen (2014), and Ishikawa et al. (2015).

The solution representation in Figure 3.1-(b) is very similar to the one shown in Figure 3.1-(a). The difference is that a gene in the RHS-Segment of the encoding in Figure 3.1-(b) can take only a single value j, the index of one of the jobs. A particular j occurs O_j number of times, where the first occurrence corresponds to the first operation of that job, the second occurrence corresponds to the second operation, and so on. With this modification, unlike the one shown in Figure 3.1-(a), a random permutation of the RHS-Segment of Figure 3.1-(b) always represents a feasible operation sequencing. Solution representation with a similar structure as the one shown in Figure 3.1-(b) appeared in many articles which include Gao et al. (2006, 2007, 2008), Gholami and Zandieh (2009), Lei (2010), Wang et al. (2010), Zhang et al. (2011), Ida and Oka (2011), Teekeng and Thammano (2012), and Driss et al. (2015).

The solution encoding in Figure 3.1-(c), combines the LHS- and RHS-Segments of the previous solution encoding into one segment where a gene can take a triple (j, o, m) where m is the index of one of the eligible machine to process operation o of job j. A gene that contains a particular j occurs O_j number of times. Moreover, for a given j, a gene with a value (j, o, m) must occur earlier in the sequence than the gene with value (j, o', m') if $o < o'$ regardless of the values of m and m'. This solution representation was first proposed in Kacem (2003) and subsequently used in many other articles including Zribi and Borne (2005), Defersha and Chen (2010b), Pezzella et al. (2008), De Giovanni and Pezzella (2010), Al-Hinai and Elmekkawy (2011), and Rohaninejad et al. (2015).

All the solution encodings in Figure 3.1 represent the same assignment and sequencing decision shown in Table 3.3. Nevertheless, due to their differences in structure, crossover and mutation operators need to be tailored to conform to the respective representation. The determination of the actual starting and finishing time of the operations and the calculation of the makespan is accomplished through a decoding procedure which

is also referred to as fitness evaluation. The decoding procedure corresponding to the solution representation in Figure 3.1-(c) for an FJSP problem with sequence-dependent setup time, attached/detached nature of setups, machine release date, and lag time was presented in Defersha and Chen (2010b). For better comprehension, the general steps of this decoding procedure are outlined in Figure 3.2. In this decoding procedure, the variable l is an index for the location of a gene in the solution encoding in Figure 3.1-(c). This index runs from 1 to the total number of operations. The notation Gene[l] represents the content of the gene at location l in the chromosome. The variable r_m is a counter for the number of runs of machine m so far assigned to operations and it increases by one every time an operation is assigned to this machine. In this decoding process, whenever an operation is assigned to a machine, the completion time is determined using the procedure outlined in Figure 3.3. In Chapter 3 we will describe the novel Two-Stage Genetic Algorithm structure that uses two different solution encodings in two stages (described in 3.3.1). Stage one has a similar structure to the RHS-Segment of the representation in Figure 3.1-(a) (without the LHS-Segment). Hence the assignment of the operations to machines is not directly encoded. But stage two utilizes the chromosome encoding as Figure 3.1-(c).

Table 3.3: Assignment and sequencing decision for the solution encoded in Figure 3.1

Machine	Assignment and Sequencing
1	$(J2, O1)(J1, O2)(J4, O2)$
2	$(J1, O1)(J2, O2)(J1, O3)(J2, O3)(J3, O2)$
3	$(J1, O4)(J3, O3)$
4	$(J5, O1)(J5, O2)(J5, O3)(J3, O1)(J4, O1)(J2, O4)$

In this research, we develop a Two-Stage Genetic Algorithm with the first stage having solution encoding and decoding different from the common approaches discussed in the previous section. Unlike those common approaches, the solution representation does not explicitly encode operations assignment and sequencing. This solution representation is illustrated in Figure 3.4-(a). It has a similar structure to the RHS-Segment of the representation in Figure 3.1-(a) (without the LHS-Segment). Hence, in this representation, the assignment of the operations to machines is not directly encoded. Without

Step 1. Set $l = 1$. Set $r_m = 0; \forall m$.

Step 2. Set (j, o, m) = Gene[l] of the chromosome in Figure 3.1-(c).

Step 3. Set $r_m = r_m + 1$.

Step 4. Assign operation o of job j to the $(r_m)^{th}$ run of machine m.

Step 5. Calculate the completion time $c_{o,j,m}$ using the procedure described in Figure 3.3.

Step 6. If l is equal to the total number of operations, go to Step 8; otherwise, go to Step 7

Step 7. Set $l = l + 1$. Go to Step 2.

Step 8. Calculate the makespan of the schedule as $c_{max} = \max\{c_{o,j,m};\ \ \forall(o, j, m)\}$.

Figure 3.2: A decoding procedure for the solution representation in Figure 3.1-(c).

the assignment decision, we also cannot see the sequencing of operation on the machines directly from the chromosome. Instead, the chromosome provides the order (from left to the right of the chromosome) in which we consider the operations for assignment and sequencing during the decoding procedure outlined in Figure 3.5. In this decoding procedure, whenever we consider an operation for assignment, we choose the machine that can complete this operation sooner while taking into account the operations that we already assigned to the various machines. This greedy nature of the first stage accelerates the search and enables the genetic algorithm to find good-quality solutions in a short computational time.

The second stage, starting from the final population of the first stage, follows the standard approach of genetic algorithm for FJSP using a solution representation shown in Figure 3.4-(b) where both operation assignment and sequencing are directly encoded. Therefore, during this stage of the search, both assignment and sequencing are determined through a random process. Thus, the second stage enables the algorithm to search the entire solution space by including solutions that might have been excluded because of the greedy nature of the first stage. During the transition from the first to the second stage, all the final solutions of the first stage are augmented by their

If an operation o of job j is to be assigned on machine m, its completion time $c_{o,j,m}$ is calculated based on one of the following four cases:

- **Case 1:**
  (a) Operation o of job j is the first operation to be assigned on machine m (i.e., $r_m = 1$), and
  (b) $o = 1$.

  $c_{o,j,m} = D_m + S^*_{o,j,m} + B_j \times T_{o,j,m}$

- **Case 2:**
  (a) Operation o of job j is the first operation to be assigned on machine m (i.e., $r_m = 1$),
  (b) $o > 1$, and
  (c) Operation $o-1$ of job j is assigned on machine m'.

  $c_{o,j,m} = \max\{D_m + (1 - A_{o,j}) \times S^*_{o,j,m} \quad , \quad c_{o-1,j,m'} + L_{o,j}\} + B_j \times T_{o,j,m} + A_{o,j} \times S^*_{o,j,m}$

- **Case 3:**
  (a) Operation o of job j is not the first operation to be assigned on machine m (i.e., $r_m > 1$),
  (b) Operation o' of job j' is the operation to be processed immediately before operation o of job j on machine m (i.e., Operation o' of job j' was assigned to run $r_m - 1$ of machine m), and
  (c) $o = 1$.

  $c_{o,j,m} = c_{o',j',m} + S_{o,j,m,o',j'} + B_j \times T_{o,j,m}$

- **Case 4:**
  (a) Operation o of job j is not the first operation to be assigned on machine m (i.e., $r_m > 1$),
  (b) Operation o' of job j' is assigned immediately before operation o of job j on machine m (i.e., Operation o' of job j' was assigned to run $r_m - 1$ of machine m),
  (c) $o > 1$, and
  (d) Operation $o-1$ of job j is assigned on machine m'.

  $c_{o,j,m} = \max\{c_{o',j',m} + (1 - A_{o,j}) \times S_{o,j,m,o',j'} \quad , \quad c_{o-1,j,m'} + L_{o,j}\} + B_j \times T_{o,j,m} + A_{o,j} \times S_{o,j,m,o',j'}$

Figure 3.3: Calculation of completion time $c_{o,j,m}$ in an FJSP with sequence dependent setup, detached/attached nature of setup, machine release date and lag time (adopted from Defersha and Chen (2010b))

respective operation assignments as determined by the decoding procedure in Figure 3.5 and become the initial population for the second stage. This stage uses the decoding procedure outlined in Figure 3.2.

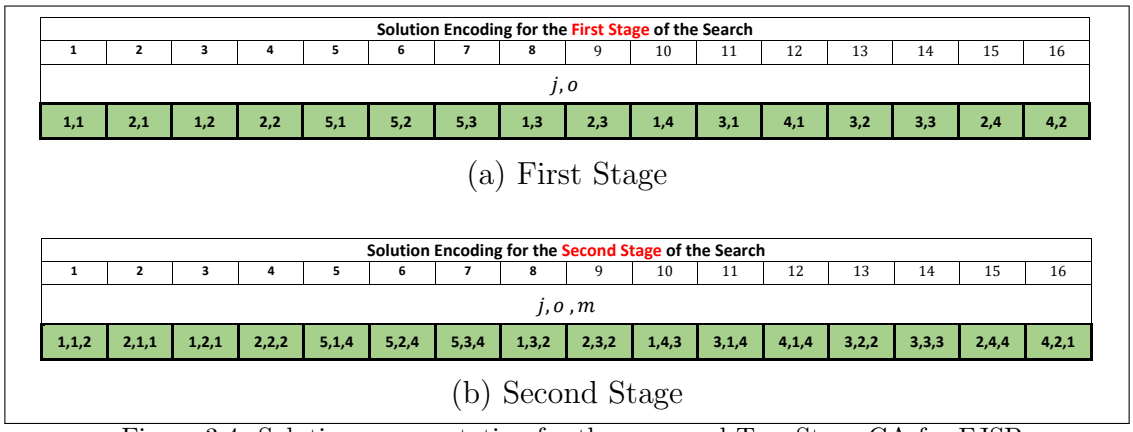


Figure 3.4: Solution representation for the proposed Two-Stage GA for FJSP

3.3.2. Genetic Operators

Imitating the natural evolution of organisms is the underlying mechanism of the genetic algorithm. In doing so, it evolves a population of solutions using various genetic operators, which can be classified into three main categories: selection, crossover, and mutation. The selection operator is used to mimic Darwin's principle of ***the survival of the fittest***, so better-fit individuals have more chances to survive and be carried forward to the next generation leaving behind the less fit ones. This operator can be applied in several different schemes, such as (i) Roulette Wheel, (ii) Ranked, (iii) Truncation, (iv) Boltzmann, (v) Elitism, and (vi) K-ways Tournament. As discussed at 2.5.1, K-ways Tournament selection is superior to the rest of the popular selection techniques and therefore, we used it in the Two-Stage Genetic Algorithm. At k-way tournament k chromosomes are randomly selected, then the tournament selects the chromosome with the best fitness and inserts it into the reproduction pool. This process will be repeated until the reproduction pool size is equal to the population size. In this process, the selection focus of the fittest can be increased or decreased by increasing or decreasing the value of k. This adjustment is needed to strike a balance between exploitation (when

Step 1. Set $l = 1$. Set $r_m = 0; \forall m$.

Step 2. Set $(j, o) = \text{Gene}[l]$ of the chromosome in Figure 3.4-(a).

Step 3. Temporarily set $r_m = r_m + 1$ for each eligible machine m of operation o of job j (for each m such that $p_{o,j,m} = 1$). Using the procedure described in Figure 3.3, calculate the completion time of operation o of job j corresponding to each of these machines.

Step 4. Using the results from Step 3, select the machine that can complete the operation sooner.
Say this machine is machine m^*.

Step 5. Assign operation o of job j to the $(r_{m^*})^{th}$ run of machine m^*.

Step 6. Assign the completion time calculated in Step 3 corresponding to machine $m = m^*$ to the variable $c_{o,j,m}$.

Step 7. Set $r_m = r_m - 1$ corresponding to all the other machines considered in Step 3 but not selected to processes operation o of job j in Step 4.

Step 8. If l is equal to the total number of operations, go to Step 10; otherwise go to Step 9

Step 9. Set $l = l + 1$. Go to Step 2. .

Step 10. Calculate the makespan of the schedule as $c_{max} = \max\{c_{o,j,m};\ \ \forall(o, j, m)\}$.

Figure 3.5: A decoding procedure for the solution representation in Figure 3.4-(a) - first stage of the Two-Stage Genetic Algorithm.

k is large) and exploration (when k is small).

Once the selection process is completed, the individuals in the mating pool are randomly paired, and on each pair, crossover operators are applied. Then, the resulting children undergo different mutation operators and finally constitute the population for the next generation. In the proposed Two-Stage Genetic Algorithm, we used several crossover and mutation operators that are tailored to FJSP and the proposed Two-Stage Genetic Algorithm. These operators are Single-Point Crossover Operator (SCO), Job Crossover Operator(JCO), Assignment Crossover Operator (ACO), Operations Swapping Mutation (OSM), and Assignment Altering Mutation (AAM).

The SCO first selects an arbitrary crossover point on the parent chromosomes. Then, it creates a child chromosome by copying all the genes that lie to the left of the crossover point from one parent and completes the remaining genes in the sequence they appear in the other parent. Figure 3.6 illustrates the creation of Child-1 using this operator. Child-2 is created by preserving the part of the chromosome of Parent-2 that lies to the left of the crossover point. SCO is applied with a probability p_1. The JCO, applied with a probability p_2, randomly selects some jobs and then copies all the genes that correspond to these jobs from Parent-1 to Child-1, as shown in Step-1 of Figure 3.7. In Step-2, it completes the child by copying the missing genes in the order they appear in Parent-2. At the same time, this operator also creates Child-2 (not shown in the figure) by starting Step-1 from Parent-2. Here, it is essential to note that both SCO and JCO ensure that a gene with a specific value $(j,\ o)$ appears only once, and earlier in the chromosome than a gene with values $(j,\ o')$ if $o < o'$. ACO exchanges the machine assignment of the operations between two parent chromosomes with a probability p_3. A two-step creation of Child-1 using this operator has been illustrated in Figure 3.8, whereas Child-2 is created by starting Step-1 from Parent-2. Once a child chromosome is created, it will be subjected to mutation operators. OSM randomly peaks two adjacent genes on a child chromosome and swaps their location as long they belong to different jobs. AAM randomly selects a gene and changes the machine assignment of the corresponding operation to another eligible machine. The

mutation operators AAM and OSM are applied with probabilities p_4 and p_5, respectively. In the proposed Two-Stage GA, the operators SCO, JCO, and OSM are all applicable during both the first and second stages, except that each gene in a chromosome has three elements during the second stage. In contrast, ACO and AAM are applicable only during the second stage of the search.

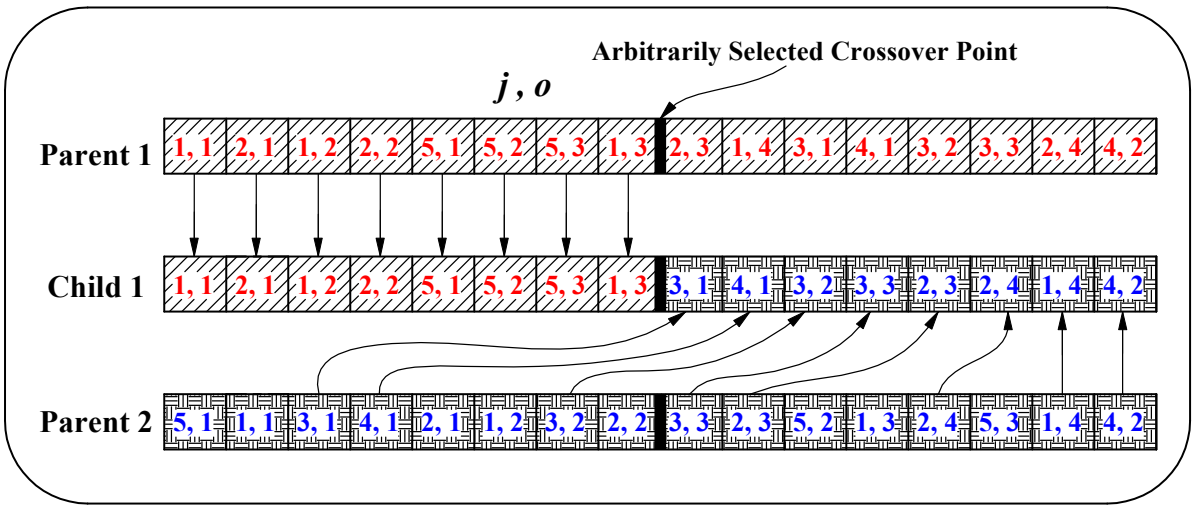


Figure 3.6: Single-Point Crossover Operator (SCO)

3.4. Numerical Studies

In this section, we present numerical studies to shed light on the features of the proposed Two-Stage GA and demonstrate its superiority over the Regular Genetic Algorithm (RGA). Solution quality, convergence speed, and algorithm robustness were used as the bases of comparison.

3.4.1. Solution Encoding/Decoding Comparison

The solution encoding and the accompanying decoding procedure shown in Figures 3.4-a and 3.5, respectively, are the key inventions that enable the design of an efficient Two-Stage GA for FJSP. In this section, we illustrate and compare a schedule that results from this new encoding-decoding pair to that from the conventional one.

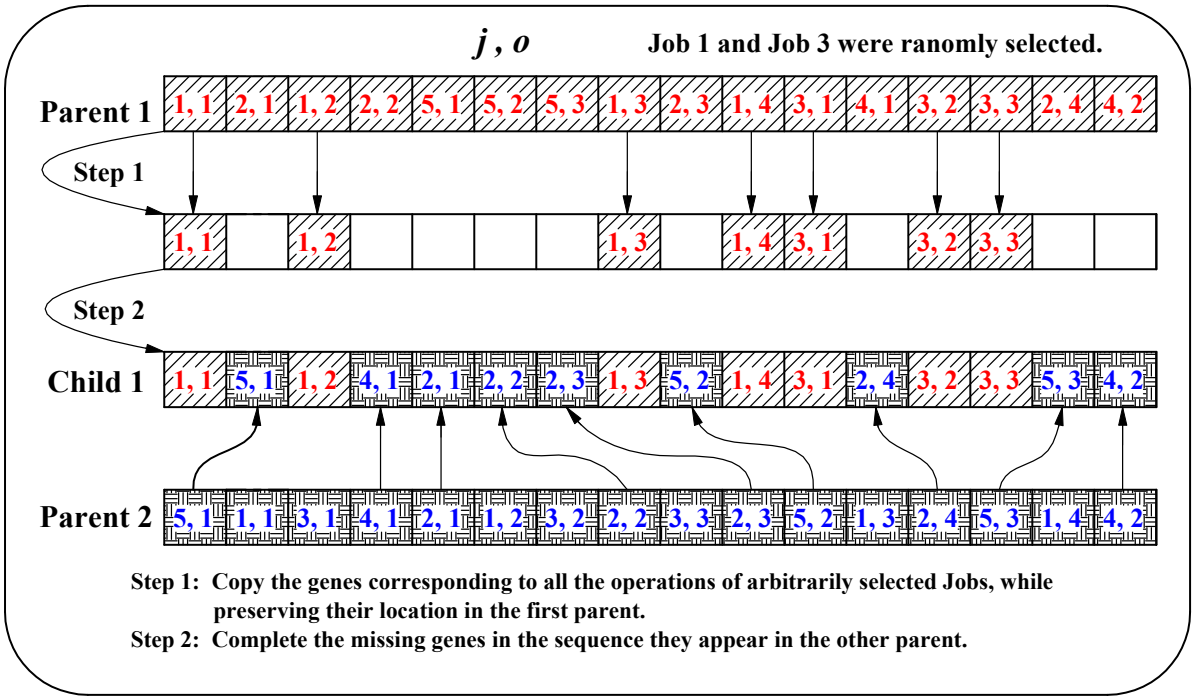


Figure 3.7: Job Crossover Operator (JCO)

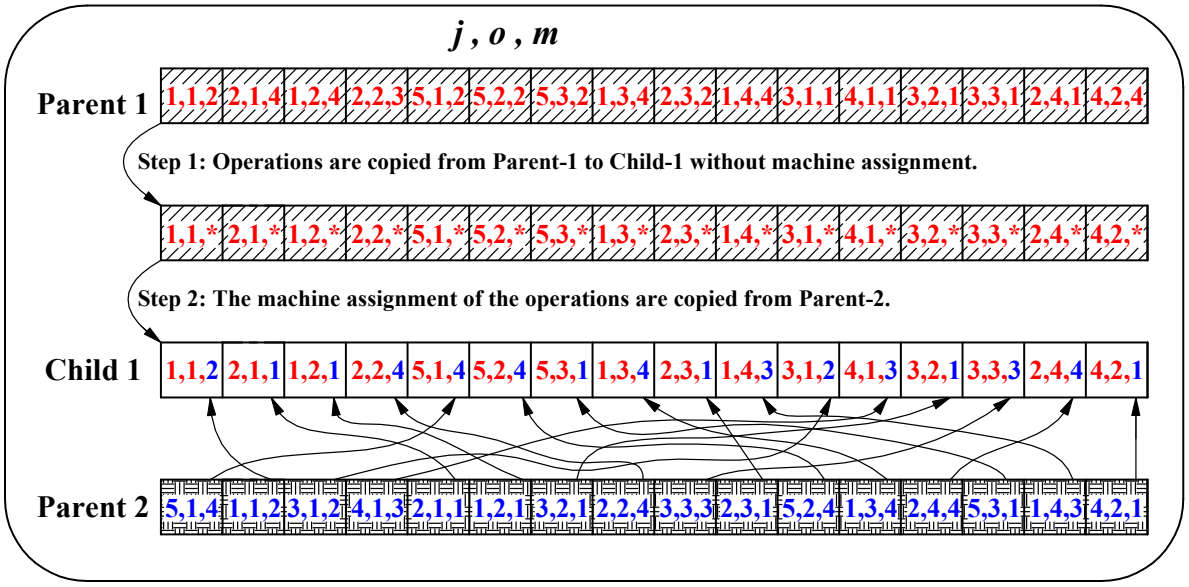


Figure 3.8: Assignment Crossover Operator (ACO)

A randomly generated solution based on the conventional approach with chromosome encoding of (o, j, m) as it is presented in Figure 3.1-c for Problem-1 was decoded using its corresponding decoding procedure. A partial illustration of the decoding procedure is outlined in the second column of Table 3.4. The gene at location $l = 1$ has a value $(j, o, m) = (1, 1, 2)$. Hence operation 1 of job 1 is the first operation to be assigned to machine 2. Its completion time will be calculated under case-1 of the procedure described Figure 3.3 using the equation $c_{o,j,m} = D_m + S^*_{o,j,m} + B_j \times T_{o,j,m}$. The value of the gene at $l = 2$ is $(j, o, m) = (2, 1, 1)$. In this case, the first operation of job 2 is the first operation for machine 1, and its completion time will be calculated under case-1 of Figure 3.3. At location $l = 3$ the gene has a value $(j, o, m) = (1, 2, 1)$. In this case, machine 1 was previously assigned to another operation $(j', o') = (2, 1)$, and operation $o = 2$ is not the first operation for the job $j = 1$. Hence, the completion time needs to be calculated under case 4 of Figure 3.3 using the equation $c_{o,j,m} = \max\{c_{o',j',m}+(1-A_{o,j})\times S_{o,j,m,o',j'} \ , \ c_{o-1,j,m'}+L_{o,j}\}+B_j\times T_{o,j,m}+A_{o,j}\times S_{o,j,m,o',j'}$. This process continues until all the genes at location $l = 16$ are processed. The resulting schedule is depicted in a Gantt chart in Figure 3.9-i, where the makespan is 2400 minutes. The Gantt chart also shows other features of the problem considered, such as machine release date, overlapping of detached setups with operations, and lag time. The detailed numerical values of the Gantt chart are given in Table 3.5.

The same randomly generated solution, shown in Figure 3.4-a, where only the machine assignments are stripped-off, was decoded using the procedure in Figure 3.5. A partial illustration of the decoding procedure is outlined in the third column of Table 3.4. The value of the gene at location $l = 1$ is $(j, o) = (1, 1)$. Hence, operation $o = 1$ of job $j = 1$ is the first operation to be considered for assignment. Machines 1 and 2 are the two alternative machines for this operation. Since $o = 1$ and the machines were not assigned previously, the completion time is calculated under Case 1 of Figure 3.3. As it can be seen from row-1-column-3 of Table 3.4, machine 2 can complete this operation sooner. Hence, the operation is assigned to machine 2. The processing of the gene at locations $l = 2$, $l = 3$ and $l = 4$ are detailed in this table where the operations

$(j2, o1)$ and $(j2, o1)$ are assigned to machine 4, and operation $(j2, o1)$ is assigned to machine 3. This procedure continues until all the genes are processed. The resulting schedule is shown in Figure 3.9-ii with a makespan of 1637.5 minutes, which is about 38% reduction compared to the one resulting from the conventional approach. This improvement leads to a high-quality initial population and a rapid convergence rate of the proposed algorithm, as it is illustrated in the following sections.

3.4.2. Initial Population Quality

The quality of the initial population of an evolutionary algorithm is a crucial factor that affects its ability to find good solutions with less computational time (Bajer et al., 2016; Rahnamayan et al., 2007). If the initial population is constituted of good individuals, it may lead the search toward promising regions of the problem space from the get-go. In this section, we compare the qualities of the initial populations generated for different problem sizes by (i) a purely random approach, (ii) Approach by Localization (AL), and (iii) our new solution encoding-decoding mechanism. AL is the method initially created by Kacem et al. (2001) to generate a promising initial population in solving FJSP using evolutionary optimization, then was extended in Defersha and Chen (2010b) to account for sequence dependent setup time and machine release date (as it is used here). The comparison is accomplished by plotting the histograms of the makespan of the initial populations. Figure 3.10-a is the histogram for one thousand initial solutions for Problem-1. The average makespan of the randomly generated population is 2801 minutes. This average was reduced by 17.2 % and 38.1% by using the AL initialization method and our new approach, respectively. This clearly shows that our encoding-decoding scheme greatly improves the quality of the initial population. A much more interesting result was obtained when we repeat this analysis by increasing the problem size (see Figure 3.10). As we move from Problems 1, 2, 3, to 4, the reductions of the average makespan by the AL method fall from 17.2, 9.2, 6.4, to 3.76%. This confirms that the improvement using the AL method is rapidly lost as the problem size increases. Whereas, in using our new approach, the average makespan reductions in the initial populations of Problems 1,

Table 3.4: Partial illustration of the decoding procedure for the solutions representations depicted in Figures 3.1-c and 3.4-a, corresponding to RGA and the first stage of 2SGA, respectively.

l	RGA decoding	2SGA first stage decoding
1	**Step 2:** Set $(j,o,m)=(1,1,2)$; **Step 3:** Increase the run counter of machine 2 by 1, i.e., $r_2=r_2+1=0+1=1$; **Step 4:** Assign operation 1 of job 1 to run 1 of machine 2; **Step 5:** Calculate the completion time $c_{1,1,2}$ under Case 1 ($r_m=1,\ \ o=1$)) in the procedure described in Figure 3.3: $c_{1,1,2}=D_2+S^*_{1,1,2}+B_1\times T_{1,1,2}=0+80+45\times 6.25=361.2$; **Step 6:** Since $l<16$, go to the next step; **Step 7:** Set $l=l+1=1+1=2$. Got to Step 2;	**Step 2:** Set $(j,o)=(1,1)$; **Step 3:** Temporarily increase the run counters of each of the two alternative machines for operation 1 of job 1 (machines 1 and 2) by 1, i.e., $r_1=r_1+1=0+1=1$ and $r_2=r_2+1=0+1=1$; **Step 4:** Calculate the completion times $c_{1,1,1}$ and $c_{1,1,2}$ under Case 1 ($r_m=1,\ \ o=1$)) in the procedure described in Figure 3.3: $c_{1,1,1}=D_1+S^*_{1,1,1}+B_1\times T_{1,1,1}=840+60+45\times 6.75=$ **1203.75**; $c_{1,1,2}=D_2+S^*_{1,1,2}+B_1\times T_{1,1,2}=0+80+45\times 6.25=$ **361.2**; **Steps 5 and 6:** Since $c_{1,1,2}<c_{1,1,2}$, machine 2 can complete this operation sooner. Assign operation 1 of job 1 to run 1 of machine 2 with a completion time of $c_{1,1,2}=361.2$; **Step 7:** Reduce the run counter of the unselected machine by 1, i.e., $r_1=r_1-1=1-1=0$; **Step 8:** Since $l<16$, go to the next step; **Step 9:** Set $l=l+1=1+1=2$. Got to Step 2;
2	**Step 2:** Set $(j,o,m)=(2,1,1)$; **Step 3:** Increase the run counter of machine 1 by 1, i.e., $r_1=r_1+1=0+1=1$; **Step 4:** Assign operation 1 of job 2 to run 1 of machine 1; **Step 5:** Calculate the completion time $c_{1,2,1}$ under Case 1 ($r_m=1,\ \ o=1$)) in the procedure described in Figure 3.3: $c_{1,2,1}=D_1+S^*_{1,2,1}+B_1\times T_{1,2,1}=840+60+35\times 5.00=1075.00$; **Step 6:** Since $l<16$, go to the next step; **Step 7:** Set $l=l+1=2+1=3$. Got to Step 2;	**Step 2:** Set $(j,o)=(2,1)$; **Step 3:** Temporarily increase the run counters of each of the two alternative machines for operation 1 of job 1 (machines 1 and 4) by 1, i.e., $r_1=r_1+1=0+1=1$ and $r_4=r_4+1=0+1=1$; **Step 4:** Calculate the completion times $c_{1,2,1}$ and $c_{1,2,4}$ under Case 1 ($r_m=1,\ \ o=1$)) in the procedure described in Figure 3.3: $c_{1,2,1}=D_1+S^*_{1,2,1}+B_2\times T_{1,2,1}=840+60+35\times 5.0=$ **1075.00**; $c_{1,2,4}=D_4+S^*_{1,2,4}+B_2\times T_{1,2,4}=120+80+35\times 5.0=$ **375.00**; **Steps 5 and 6:** Since $c_{1,2,4}<c_{1,2,1}$, machine 4 can complete this operation sooner. Assign operation 1 of job 2 to run 1 of machine 4 with a completion time of $c_{1,2,4}=375.00$; **Step 7:** Reduce the run counter of the unselected machine by 1, i.e., $r_1=r_1-1=1-1=0$; **Step 8:** Since $l<16$, go to the next step; **Step 9:** Set $l=l+1=2+1=3$. Got to Step 2;
3	**Step 2:** Set $(j,o,m)=(1,2,1)$; **Step 3:** Increase the run counter of machine 1 by 1, i.e., $r_1=r_1+1=1+1=2$; **Step 4:** Assign operation 1 of job 2 to run 2 of machine 1; **Step 5:** Calculate the completion time $c_{2,1,1}$ under Case 4 ($r_m>1,\ \ o>1$)) in the procedure described in Figure 3.3: $c_{2,1,1}=\max\{c_{1,2,1}+(1-A_{2,1})\times S_{2,1,1,1,2},\ c_{1,1,2}+L_{2,1}\}+B_j\times T_{2,1,1}+A_{2,1}\times S_{2,1,1,1,2}=\max\{1075.00+(1-1)\times 90,\ 361.25+0\}+B_j\times 4.25+A_{2,1}\times 90=1356.25$; **Step 6:** Since $l<16$, go to the next step; **Step 7:** Set $l=l+1=3+1=4$. Got to Step 2;	**Step 2:** Set $(j,o)=(1,2)$; **Step 3:** Temporarily increase the run counters of each of the two alternative machines for operation 1 of job 1 (machines 1 and 4) by 1, i.e., $r_1=r_1+1=0+1=1$ and $r_4=r_4+1=1+1=2$; **Step 4:** Calculate the completion times $c_{2,1,1}$ under Case 2 ($r_m=1,\ \ o>1$)) and $c_{2,1,4}$ under Case 4 ($r_m>1,\ \ o>1$)) in the procedure described in Figure 3.3: $c_{2,1,1}=\max\{D_1+(1-A_{2,1})\times S^*_{2,1,1},\ c_{1,1,2}+L_{2,1}\}+B_1\times T_{2,1,1}+A_{2,1}\times S^*_{2,1,1}=\max\{840+(1-1)\times 60,\ 361.25+0\}+45\times 4.5+1\times 60=$ **1091.25** $c_{2,1,4}=\max\{c_{1,2,4}+(1-A_{2,1})\times S_{2,1,4,1,2},\ \ c_{1,1,2}+L_{2,1}\}+B_1\times T_{2,1,4}+A_{2,1}\times S_{2,1,4,1,2}=\max\{375+(1-1)\times 90,\ \ 361.25+0\}+45\times 4.5+1\times 90=$ **667.50**; **Steps 5 and 6:** Since $c_{2,1,4}<c_{2,1,1}$, machine 4 can complete this operation sooner. Assign operation 2 of job 1 to run 2 of machine 4 with a completion time of $c_{2,1,4}=667.50$; **Step 7:** Reduce the run counter of the unselected machine by 1, i.e., $r_1=r_1-1=1-1=0$; **Step 8:** Since $l<16$, go to the next step; **Step 9:** Set $l=l+1=3+1=4$. Got to Step 2;
4	**Step 2:** Set $(j,o,m)=(2,2,2)$; **Step 3:** Increase the run counter of machine 2 by 1, i.e., $r_2=r_2+1=1+1=2$; **Step 4:** Assign operation 2 of job 2 to run 2 of machine 2; **Step 5:** Calculate the completion time $c_{2,2,2}$ under Case 4 ($r_m>1,\ \ o>1$)) in the procedure described in Figure 3.3: $c_{2,2,2}=\max\{c_{1,1,2}+(1-A_{2,2})\times S_{2,2,2,1,1},\ \ c_{1,2,1}+L_{2,2}\}+B_2\times T_{2,2,2}+A_{2,2}\times S_{2,2,2,1,1}=\max\{361.25+(1-1)\times 120\ ,\ 1075.00+0\}+35\times 6.5+1\times 120=1422.50$; **Step 6:** Since $l<16$, go to the next step; **Step 7:** Set $l=l+1=3+1=4$. Got to Step 2;	**Step 2:** Set $(j,o)=(2,2)$; **Step 3:** Temporarily increase the run counters of each of the four alternative machines for operation 2 of job 2 (machines 1, 2, 3 and 4) by 1, i.e., $r_1=r_1+1=0+1=1$, $r_2=r_2+1=1+1=2$, $r_3=r_3+1=0+1=1$ and $r_4=r_4+1=2+1=3$; **Step 4:** Calculate the completion times $c_{2,2,1}$ and $c_{2,2,3}$ under Case 2 ($r_m=1,\ \ o>1$)) and $c_{2,2,2}$ and $c_{2,2,4}$ under Case 4 ($r_m>1,\ \ o>1$)) in the procedure described in Figure 3.3: $c_{2,2,1}=\max\{D_1+(1-A_{2,2})\times S^*_{2,2,1},\ \ c_{1,2,4}+L_{2,2}\}+B_2\times T_{2,2,1}+A_{2,2}\times S^*_{2,2,1}=\max\{840+(1-1)\times 40,\ \ 375+0\}+35\times 6.75+1\times 40=$ **1116.25**; $c_{2,2,2}=\max\{c_{1,1,2}+(1-A_{2,2})\times S_{2,2,2,1,1},\ \ c_{1,2,4}+L_{2,2}\}+B_2\times T_{2,2,2}+A_{2,2}\times S_{2,2,2,1,1}=\max\{361.25+(1-1)\times 120,\ \ 375.00+0\}+35\times 6.5+1\times 120=$ **722.50**; $c_{2,2,3}=\max\{D_3+(1-A_{2,2})\times S^*_{2,2,3},\ \ c_{1,2,4}+L_{2,2}\}+B_j\times T_{2,2,3}+A_{2,2}\times S^*_{2,2,3}=\max\{0+(1-1)\times 80,\ \ 375.00+0\}+35\times 6.75+1\times 80=$ **691.25**; $c_{2,2,4}=\max\{c_{2,1,4}+(1-A_{2,2})\times S_{2,2,4,2,1},\ \ c_{1,2,4}+L_{2,2}\}+B_2\times T_{2,2,4}+A_{2,2}\times S_{2,2,4,1,2}=\max\{667.5+(1-1)\times 90,\ \ 375.00+0\}+35\times 6.50+1\times 90=$ **985.00**; **Steps 5 and 6:** Since $c_{2,2,3}<c_{2,1,m}|m\in\{1,2,4\}$, machine 3 can complete this operation sooner. Assign operation 2 of job 2 to run 1 of machine 3 with a completion time of $c_{2,2,3}=691.25$; **Step 7:** Reduce the run counter of the unselected machine by 1, i.e., $r_1=r_1-1=1-1=0$, $r_2=r_2-1=2-1=1$ and $r_4=r_4-1=3-1=2$; **Step 8:** Since $l<16$, go to the next step; **Step 9:** Set $l=l+1=4+1=5$. Got to Step 2;

This process continues until all the operation are assigned when $l=16$.

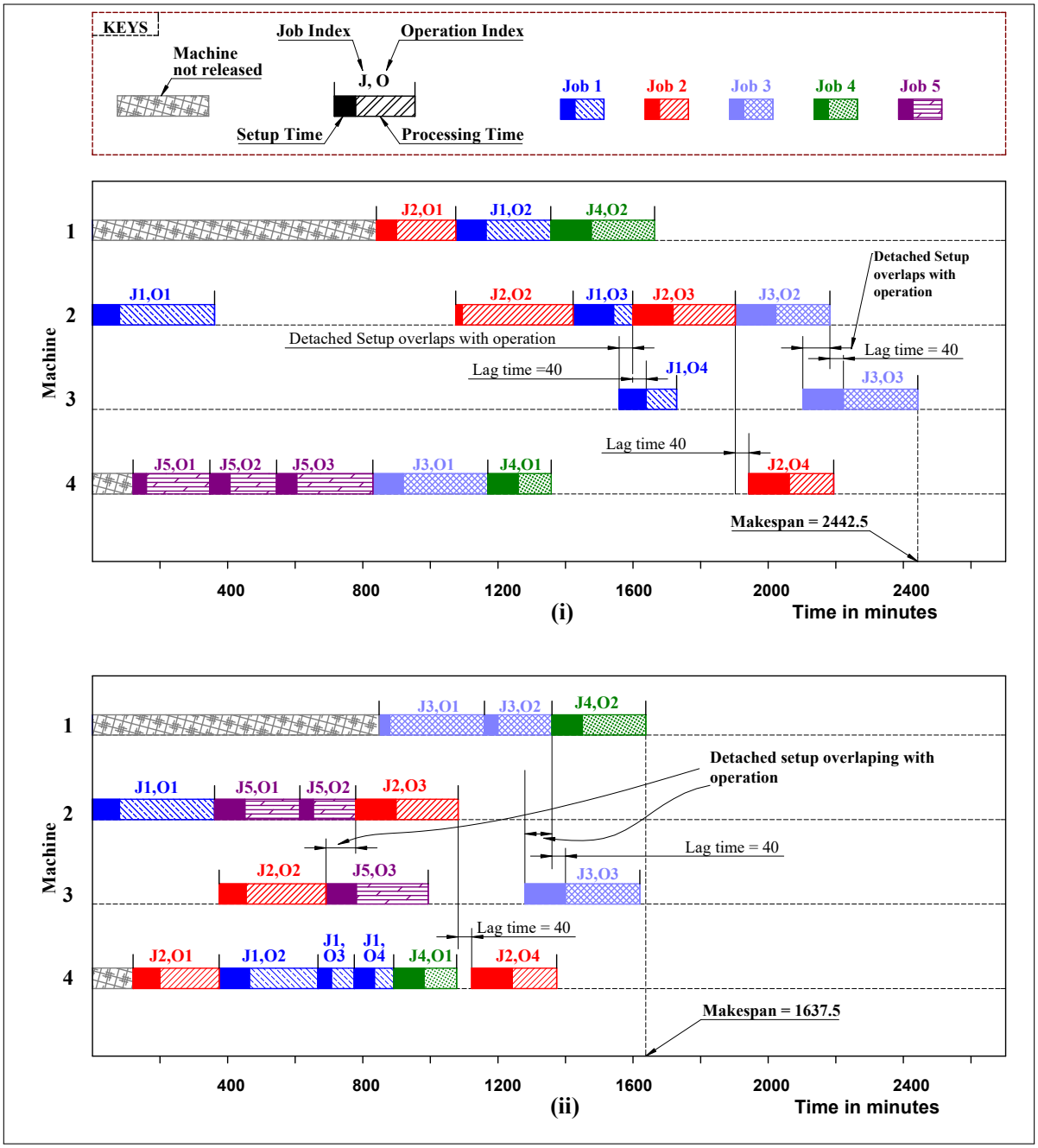


Figure 3.9: Gantt chart of the schedule corresponding to (i) the chromosome in Fig 3.1-c and the decoding procedure in Figure 3.2, and (ii) the chromosome in Fig 3.4-a and the decoding procedure in Figure 3.5.

Table 3.5: The numerical values of the schedules shown in the Gantt chart of Figure 3.9

Machine	Run	Details for Fig 3.9-(i)				Details for Fig 3.9-(ii)			
m	r	(j, o)	SB	SE/PB	PE	(j, s, o)	SB	SE/PB	PE
1	1	(2,1)	840.0	900.0	1075.0	(3,1)	840.0	880.0	1160.0
	2	(1,2)	1075.0	1165.0	1356.3	(3,2)	1160.0	1200.0	1360.0
	3	(4,2)	1356.3	1476.3	1663.8	(4,2)	1360.0	1450.0	1637.5
2	1	(1,1)	0.0	80.0	361.3	(1,1)	0.0	80.0	361.3
	2	(2,2)	1075.0	1195.0	1422.5	(5,1)	361.3	451.3	613.8
	3	(1,3)	1422.5	1542.5	1598.8	(5,2)	613.8	653.8	778.8
	4	(2,3)	1598.8	1718.8	1902.5	(2 3)	778.8	898.8	1082.5
	5	(3,2)	1902.5	2022.5	2182.5	NA	NA	NA	NA
3	1	(1,4)	1558.8	1638.8	1728.8	(2,2)	375.0	455.0	691.3
	2	(3,3)	2102.5	2222.5	2442.5	(5,3)	691.3	781.3	993.8
	3	NA	NA	NA	NA	(3,3)	1280.0	1400.0	1620.0
4	1	(5,1)	120.0	160.0	347.5	(2,1)	120.0	200.0	375.0
	2	(5,2)	347.5	407.5	545.0	(1,2)	375.0	465.0	667.5
	3	(5,3)	545.0	605.0	830.0	(1,3)	667.5	707.5	775.0
	4	(3,1)	830.0	920.0	1170.0	(1,4)	775.0	835.0	891.3
	5	(4,1)	1170.0	1260.0	1357.5	(4,1)	891.3	981.3	1078.8
	6	(2,4)	1942.5	2062.5	2193.8	(2,4)	1122.5	1242.5	1373.8

Note: SB, SE, PB, PE stand for setup begins, setup ends, processing begins, and processing ends, respectively.
NA means that a run is Not Assigned to an operation.

2, 3, and 4 are 38.1, 47.71, 52.13, and 49.47%, respectively, which enables the proposed algorithm to archive a high-quality solution from the beginning of the search.

However, based on the histograms in Figure 3.10, one may argue that in our approach, the diversity of the initial population is very poor, in particular as the problem size increases. Nevertheless, it is important to note that the histograms show only the distribution of the phenotype (makespan). The genotype (the sequence of the genes) of each individual in the initial population in our approach is created with the same level of randomness as that of the randomly generated population. The initial population is the same as that of the randomly generated one, except that the machine assignments are stripped off from each chromosome as they are determined by the decoding procedure.

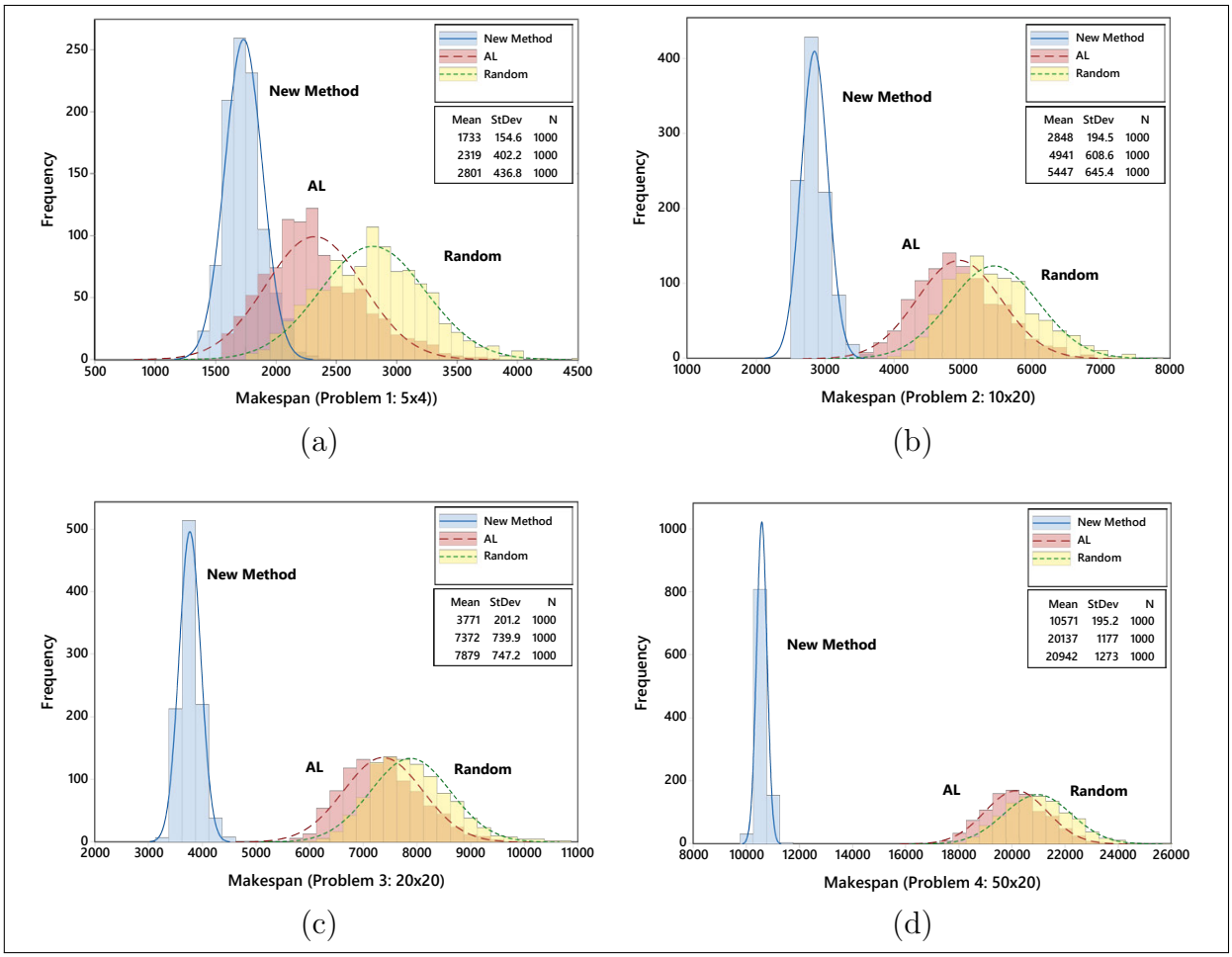


Figure 3.10: Distribution of the makespan of initial populations generated by (i) a purely random approach, (ii) AL, and (iii) our new method.

3.4.3. Preliminary Performance Assessment using Benchmark Problems

We have chosen two sets of benchmark problems to illustrate the performance of our proposed Two-Stage Genetic Algorithm (2SGA). Brandimarte (1993) introduced a set of 10 problems with the number of jobs ranging from 10 to 20, the number of machines ranging from 4 to 15, and the number of operations for each job ranging from 5 to 15. Hurink et al. (1994) developed 40 benchmark FJSP problems based on the classical job shop scheduling instances of Adams et al. (1988). Each benchmark problem has three different versions referred to as "edata", "rdata", and "vdata" in the order of increasing the number of alternative machines for the operations. These instances are well documented in Behnke and Geiger (2012). In this section, we used all the 40 "vdata" problem instances (la01-la40) to provide a preliminary performance comparison of 2SGA and RGA. On each problem instance, each algorithm was executed 40 times using 40 arbitrarily generated GA parameters shown in Table 3.6. In effect, we conducted a total of $40 \times 40 \times 2 = 3200$ experimental runs (i.e. 1600 test runs using each algorithm). Each experimental run needs from 1 to 30 minutes depending on the problem size and the GA parameter selected. Such a large number of experimental runs were possible using several hundreds of concurrently available CPUs on a High-Performance-Computation (HPC) cluster. We were able to conduct a very large number of test runs simultaneously. The results of these test runs are summarized in Table 3.7.

The first four columns of this table provide the features of the benchmark problems. An asterisk next to an upper bound (UP) indicates that the UP is also the global optimal solution. The next four columns are for the computational results when solving these benchmark problems using RGA. Each row in these four columns is a summary of 40 test runs using the 40 GA parameter sets given in Table 3.6. Columns 5 and 6 indicate the best and average objective function values. Column 7 provides the Global Optimal Hit Rate (GOHR), which is the number of times RGA was able to find the global optimal solution as we vary the GA parameter sets. The Average Optimality Gap (AOG) is

indicated in Column 8. Columns 9 through 12 provide similar information for 2SGA. The last column is for 2SGA's Win Count, which is the number of times 2SGA gets a better solution than RGA. From this table, it can be seen that 2SGA outperforms RGA in many test runs. RGA was able to reach a global optimal solution for only 192 times, whereas 2SGA gets an optimal solution for 504 times. The overall global optimality gaps are 2.11% for RGA and 0.8% for 2SGA. Summing up the values in column 13, we found that 2SGA was able to reach a better solution than RGA for 1177 times out of a total of 1600 runs. Though not shown in the table, out of these 1600 test runs, RGA wins only 149 times and both algorithms reach the same solutions for 274 times. By comparing the average objective function values in Columns 6 and 10, we can see that based on this performance measure 2SGA outperforms RGA in all the 40 benchmark problems.

For solving MK series benchmark problems introduced by Brandimarte (1993), we followed Li and Gao (2016) who reported 21 studies (including themselves) that solved these problems with different approaches. Table 3.8 shows the number of jobs (N) and the number of machines (M) of each problem and lists the best makespan reported by each of those 21 studies. Columns 25 and 26 show the results of our regular GA and 2SGA approaches, respectively. The researches listed in column 3 to column 24 and their approaches are: Tabu Search(TS) by Mastrolilli and Gambardella (2000), Learnable Genetic Architecture(LEGA) by Ho et al. (2007), Hierarchical Optimization(HO) by Zribi et al. (2007), Genetic Algorithm(GA) by Pezzella et al. (2008), Multi-Agent Tabu Search Systems(MAS) by Ennigrou and Ghédira (2008), Hybrid Genetic and Variable Neighborhood Descent Algorithm (HGVNA)by Gao et al. (2008), Artificial Immune Algorithm (AIA) by Bagheri et al. (2010), Variable Neighbourhood Search (VNS) by Amiri et al. (2010), Parallel Variable Neighborhood Search (PVNS) by Yazdani et al. (2010), Hybrid Tabu Search (HTS) by Li et al. (2011), Discrepancy Search (DS) by Ben Hmida et al. (2010), Hybrid Chaos Particle Swarm Optimization and GA (HAT) Tang et al. (2011), Artificial Bee Colony (ABC) by Wang et al. (2012), Evolutionary

Table 3.6: Randomly Generated Algorithm Parameters:

			Crossover Probabilities			Mutation Probabilities	
Parameter Set	Population Size	Tournament Size Factor	p_1	p_2	p_3	p_4	p_5
1	2000	0.005	0.95	0.80	0.95	0.15	0.15
2	3000	0.010	0.80	0.65	0.95	0.05	0.10
3	2000	0.010	0.95	0.80	0.80	0.05	0.10
4	1000	0.005	0.80	0.65	0.65	0.10	0.05
5	1000	0.010	0.95	0.65	0.65	0.10	0.15
6	3000	0.015	0.65	0.95	0.95	0.10	0.15
7	3000	0.015	0.95	0.95	0.80	0.10	0.15
8	3000	0.015	0.65	0.65	0.95	0.10	0.15
9	1000	0.015	0.65	0.80	0.65	0.10	0.10
10	1000	0.005	0.95	0.95	0.65	0.15	0.05
11	2000	0.010	0.80	0.80	0.95	0.15	0.10
12	1000	0.005	0.65	0.65	0.80	0.15	0.10
13	1000	0.005	0.80	0.65	0.80	0.10	0.10
14	2000	0.005	0.80	0.95	0.95	0.10	0.15
15	2000	0.015	0.80	0.80	0.65	0.05	0.10
16	3000	0.015	0.80	0.80	0.95	0.05	0.15
17	1000	0.005	0.80	0.95	0.80	0.05	0.10
18	1000	0.010	0.80	0.65	0.80	0.10	0.15
19	2000	0.010	0.65	0.95	0.80	0.15	0.15
20	2000	0.015	0.95	0.80	0.80	0.05	0.15
21	3000	0.010	0.80	0.65	0.65	0.15	0.15
22	1000	0.015	0.80	0.80	0.80	0.05	0.05
23	3000	0.010	0.95	0.80	0.65	0.05	0.15
24	3000	0.015	0.80	0.65	0.80	0.05	0.05
25	1000	0.015	0.80	0.65	0.80	0.05	0.15
26	3000	0.005	0.65	0.95	0.95	0.15	0.05
27	3000	0.010	0.80	0.65	0.80	0.05	0.10
28	3000	0.005	0.65	0.65	0.95	0.10	0.15
29	2000	0.010	0.65	0.65	0.95	0.05	0.10
30	1000	0.005	0.80	0.65	0.80	0.05	0.15
31	2000	0.015	0.65	0.80	0.65	0.15	0.05
32	1000	0.010	0.80	0.95	0.65	0.15	0.15
33	1000	0.010	0.95	0.95	0.65	0.05	0.15
34	3000	0.005	0.65	0.80	0.80	0.15	0.10
35	3000	0.015	0.80	0.80	0.65	0.15	0.05
36	3000	0.010	0.65	0.95	0.95	0.15	0.15
37	3000	0.005	0.80	0.80	0.65	0.15	0.05
38	2000	0.005	0.95	0.95	0.65	0.10	0.15
39	1000	0.015	0.80	0.65	0.65	0.05	0.10
40	1000	0.005	0.95	0.65	0.95	0.05	0.05

Table 3.7: Performance Comparison of RGA and 2SGA using 40 benchmark problems from Hurink et al. (1994)

Benchmark Problem (vdata)				RGA				2SGA				
Prob.	size $N \times M$	Lower Bound LP	Upper Bound UP	Best Obj.	Ave. Obj.	GOHR	AOG%	Best Obj.	Ave. Obj.	GOHR	AOG%	2SGA's Win Count
la01	10 × 5	570	570*	571	576.70	0	1.18	570*	572.25	1	0.48	38
la02	10 × 5	529	529*	530	535.35	0	1.02	529*	532.33	3	0.63	26
la03	10 × 5	477	477*	479	484.90	0	1.66	477*	481.38	1	0.92	30
la04	10 × 5	502	502*	504	509.58	0	1.51	502*	505.50	1	0.70	28
la05	10 × 5	457	457	462	467.68	0	2.34	458	462.68	0	1.24	36
la06	15 × 5	799	799*	800	803.80	0	0.60	799*	800.70	13	0.21	35
la07	15 × 5	749	749*	750	755.65	0	0.89	749*	750.98	7	0.26	37
la08	15 × 5	765	765*	766	767.90	0	0.38	765*	766.45	11	0.19	27
la09	15 × 5	853	853*	855	858.08	0	0.59	853*	854.48	11	0.18	39
la10	15 × 5	804	804*	805	807.58	0	0.44	804*	805.55	8	0.19	32
la11	20 × 5	1071	1071*	1071*	1074.48	1	0.32	1071*	1071.95	22	0.09	36
la12	20 × 5	936	936*	936*	938.53	2	0.27	936*	936.75	22	0.08	31
la13	20 × 5	1038	1038*	1039	1041.35	0	0.32	1038*	1038.80	23	0.08	36
la14	20 × 5	1070	1070*	1071	1072.83	0	0.26	1070*	1071.00	20	0.09	33
la15	20 × 5	1089	1089*	1090	1092.33	0	0.31	1089*	1090.13	12	0.01	36
la16	10 × 10	717	717*	717*	717.69	38	0.09	717*	717.00	40	0.00	2
la17	10 × 10	646	646*	646*	646	40	0.00	646*	646.00	40	0.00	0
la18	10 × 10	663	663*	663*	663	40	0.00	663*	663.00	40	0.00	0
la19	10 × 10	617	617*	617*	644.03	2	4.38	617*	619.55	28	0.41	36
la20	10 × 10	756	756*	756*	756	40	0.00	756*	756.00	40	0.00	0
la21	15 × 10	800	806	813	856.48	NA	7.02	803†	825.58	NA	3.20	37
la22	15 × 10	733	739	748	781.08	NA	6.56	736†	751.10	NA	3.29	38
la23	15 × 10	809	815	824	858.90	NA	6.17	813†	834.00	NA	3.09	38
la24	15 × 10	773	777	787	822.85	NA	6.45	775†	796.45	NA	3.03	37
la25	15 × 10	751	756	765	804.48	NA	7.12	753†	777.55	NA	3.54	38
la26	20 × 10	1052	1054	1056	1077.03	NA	2.38	1053†	1064.55	NA	1.19	33
la27	20 × 10	1084	1085	1087	1106.05	NA	2.03	1085†	1099.13	NA	1.40	30
la28	20 × 10	1069	1070	1074	1092.68	NA	2.21	1070	1083.60	NA	1.37	34
la29	20 × 10	993	994	996	1019.20	NA	2.64	995	1007.03	NA	1.14	31
la30	20 × 10	1068	1069	1073	1095.03	NA	2.53	1070	1083.15	NA	1.42	34
la31	30 × 10	1520	1520*	1522	1528.25	0	0.54	1520*	1525.98	2	0.39	25
la32	30 × 10	1657	1658	1659	1665.93	NA	0.54	1659	1665.78	NA	0.53	25
la33	30 × 10	1497	1497*	1499	1505.20	0	0.55	1498	1504.28	0	0.49	25
la34	30 × 10	1535	1535*	1536	1542.55	0	0.49	1535*	1540.80	1	0.38	23
la35	30 × 10	1549	1549*	1551	1556.33	0	0.47	1550	1555.85	0	0.44	24
la36	15 × 15	948	948*	948*	990.80	2	4.51	948*	948.8	35	0.08	38
la37	15 × 15	986	986*	988	1045.4	0	6.02	986*	992.65	21	0.67	40
la38	15 × 15	943	943*	943*	957.50	16	1.54	943*	943.00	40	0.00	24
la39	15 × 15	922	922*	931	978.98	0	6.18	922*	927.58	22	0.60	40
la40	15 × 15	955	955*	955*	972.13	15	1.79	955*	955.00	40	0.00	25
Total or Average						196	2.11			504	0.80	1177

* Global Optimal solution; †Lower than know upper bound; NA - Global Hit Rate not applicable since global optimal solution is not known.

Algorithm (EA) by Chiang and Lin (2013), Hybrid Harmony Search (HHS) by Yuan et al. (2013), Hybrid Harmony and Large Neighborhood Search (HS) by Yuan and Xu (2013), Heuristic by Ziaee (2014), Two-Stage Artificial Bee Colony (TABC) by Gao et al. (2015), Hybrid GA and Tabu Search (HGTS) by Palacios et al. (2015), Multi-objective Memetic Algorithms (MA2) by Yuan and Xu (2015) and Hybrid GA and Tabu Search (HA) by Li and Gao (2016).

To have a better comparison, table 3.9 shows the minimum and maximum of all those 21 reported makespans, along with our regular GA and 2SGA results and conversion CPU time. This shows both regular GA and 2SGA perform in the range of other studies. Columns 10 and 11 show the deviation of 2SGA makespan from the min and max values of other approaches, defined as:

dev = [(2SGA-max or min)/2SGA] * 100

As can be seen, for every problem, the Two-Stage Genetic Algorithm outperformed several approaches. Also, Two-Stage Genetic Algorithm has much better performance in terms of both conversion time and final result than regular GA.

3.4.4. Performance Comparison by Solving Large Size Problems

In the previous section, we demonstrated the superiority of 2SGA over RGA using benchmark problems. However, such a comparison is inadequate as the problem instances are small in size and do not represent the complexity of actual industrial problems. Methodologies that fare very well in small problem instances may not replicate their performance when problem sizes increase. Moreover, in industrial settings, algorithms that arrive at reasonable solutions very rapidly may be preferred over those which achieve the same or better solutions at the expense of excessive computational time and cost. In this section, using large-size problems, we illustrate that a Two-Stage Genetic Algorithm not only converges very quickly but also it arrives at a better solution than RGA. The problems considered have 50 to 140 jobs and 20 to 80 machines (see Problems 4 to 8 in Table 3.10). We run the algorithms 40 times for each problem using the 40 GA parameter

sets given in Table 3.6. Figures 3.11-a and -b show the convergence of RGA and 2SGA, respectively, while solving Problem 5. The average convergence is presented in Figures 3.11-c. The qualities (makespans) of the final solutions are given in Figure 3.11-d. As can be seen from these convergence histories, 2SGA arrives at better solutions in all the test runs than RGA. Similar results were obtained while solving Problem 6 (see Figures 3.11-e, -f, -g, and -h) and in all the other large-size problems. Figure 3.12 provides a summary of the final solution qualities for Problems 2 to 8. As it can be seen in Figure 3.12-a, the difference in the average makespan from the 40 test runs increases with problem size, and an average improvement as high as 18% is achieved in using 2SGA. Furthermore, the standard deviation of the makespan of the final solutions is lower in the case of 2SGA as shown in Figure 3.12-b, suggesting that 2SGA is more robust to the GA parameter settings than RGA.

In addition to achieving improved solution qualities and robustness to parameter settings, another yet very significant advantage of 2SGA over RGA is its fast convergence. To illustrate this advantage, in Table 3.11, we provide the time snapshots of the convergence histories of both algorithms while solving Problems 7 and 8. In Problem 7, the makespan that was achieved in just less than 5 minutes by 2SGA was not reached after more than 4 hours of computation by RGA. This advantage is further amplified in Problem 8. The solution that was obtained by 2SGA at time $t = 0$, because of its improved initial solution (see Section 3.4.2), was already better than the solution obtained by RGA after more than 9 hours of computation. The solution that was obtained in less than 30 minutes using 2SGA was not achieved after 72 hours of computation using RGA. This result clearly demonstrates the superiority of 2SGA over RGA and its potential in solving large-size industrial problems using short computational time.

3.4.5. Impact of Dividing the Search into Stages

In developing the Two-Stage Genetic Algorithm, we hypothesize that the greedy nature of the first stage of the algorithm will enable it to find good solutions with minimal computation. The second stage, following the RGA approach, will improve the solution

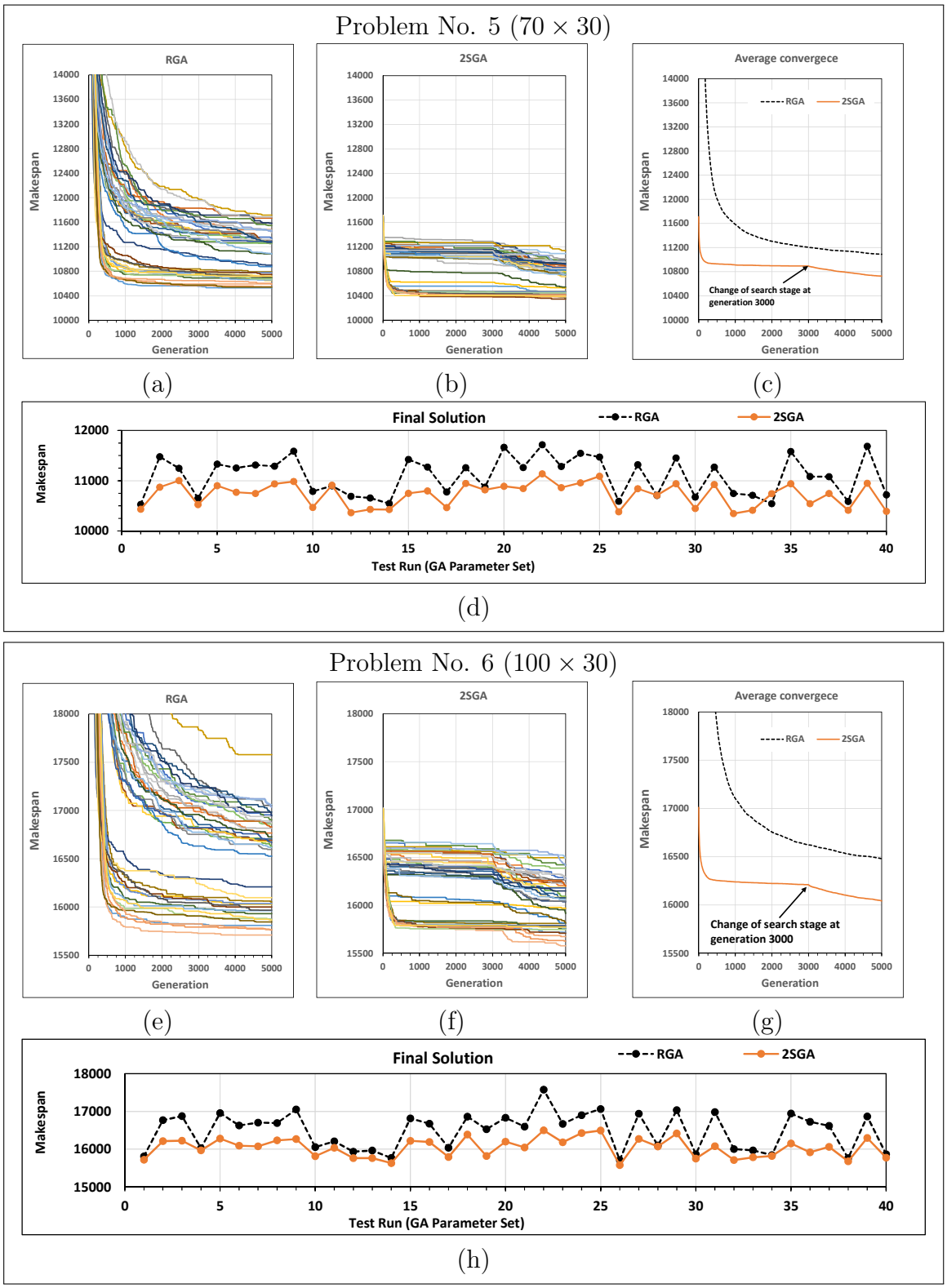


Figure 3.11: Convergence behaviours of RGA and 2SGA in solving Problems 5 and 6 (Similar convergence behaviours were observed in solving all the other large problems).

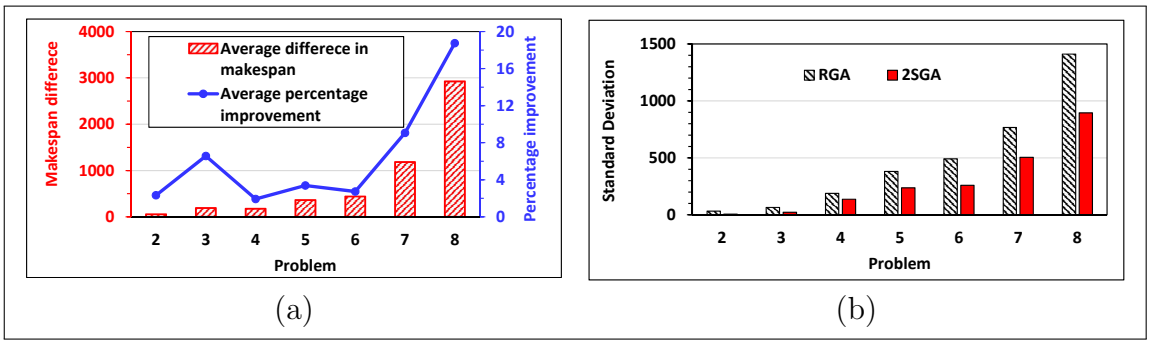


Figure 3.12: Difference in average makespan, percentage improvement and standard deviation of the final solutions from 40 test runs of RGA and 2SGA in solving Problem 2 to 8.

of the first stage by searching neighborhood solutions that might have been excluded because of the greedy nature of the first stage. The impact of this phenomenon was observed in many test runs we conducted. For example, the convergence graphs in Figures 3.11-b, -c, -f and -g, where a change of stage occurs at generation 3000, clearly shows that solution improvement were possible by allowing the algorithm to run into its second stage after the first stage converges. Figure 3.13 further illustrates this phenomenon when Two-Stage Genetic Algorithm was executed in three different modes while solving Problem 5. We differentiate these three modes by running the algorithm with (i) Stage-1 only, (ii) Stage-2 only, and (iii) Stage-1 followed by Stage-2. As can be seen in this figure (a, b, and c), the algorithm achieved further improvement in solution quality when Stage-2 followed Stage-1. Figure 3.13-d shows that the two-stage approach improved the final solution qualities in all the 40 test runs in Problem 5. Similar results were observed in all the large-size problems considered in this study.

3.4.6. Comparison with Parallel RGA and Further Performance Improvement

One of the very promising performance improvement strategies for a genetic algorithm is parallel computation. To this end, many researchers reported successful implementations of parallel genetic algorithms in different domains. Defersha and Chen (2010b) developed a Parallel RGA (P-RGA) to solve the same problem considered in this article.

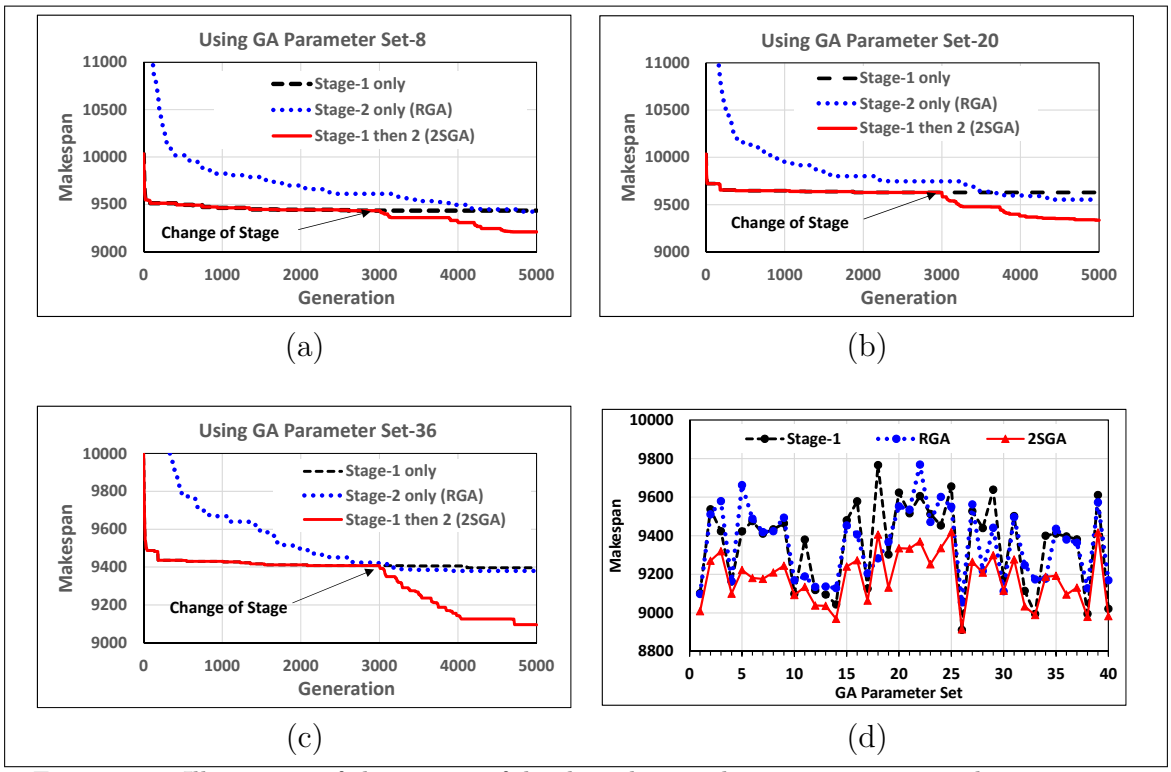


Figure 3.13: Illustration of the impact of dividing the search into two stages on the convergence behaviour of the proposed algorithm in solving Problem 5.

In this section, we present a comparison between P-RGA and the Sequential Two-Stage Genetic Algorithm (S-2SGA) and demonstrate the performance improvement of the latter using parallelization. For the parallel implementation of the GAs, we adopt the island model with dynamically and randomly connected topology, initially proposed in Defersha and Chen (2008). We used a total of 48 concurrently available processors (CPUs) in the parallel implementations. Figure 3.14 shows the convergence behavior of RGA and 2SGA both in sequential and parallel implantations while solving Problem 5. A total of ten test runs were conducted using the first ten sets of the GA parameters given in Table 3.6 while keeping the population size at 1000 in each CPU. Additional parameters pertinent to parallelization, such as topology density, migration frequency, and migration replacement policy, were set based on the recommendation in Defersha and Chen (2010b). From the convergence graphs and the final solution quality plots, it is clear to see that parallelization improves the performances of the algorithms. However, the more interesting result in this analysis is that S-2SGA (with only one CPU)

outperforms the P-RGA (with 48 CPUs) both in terms of convergence speed and final solution quality.

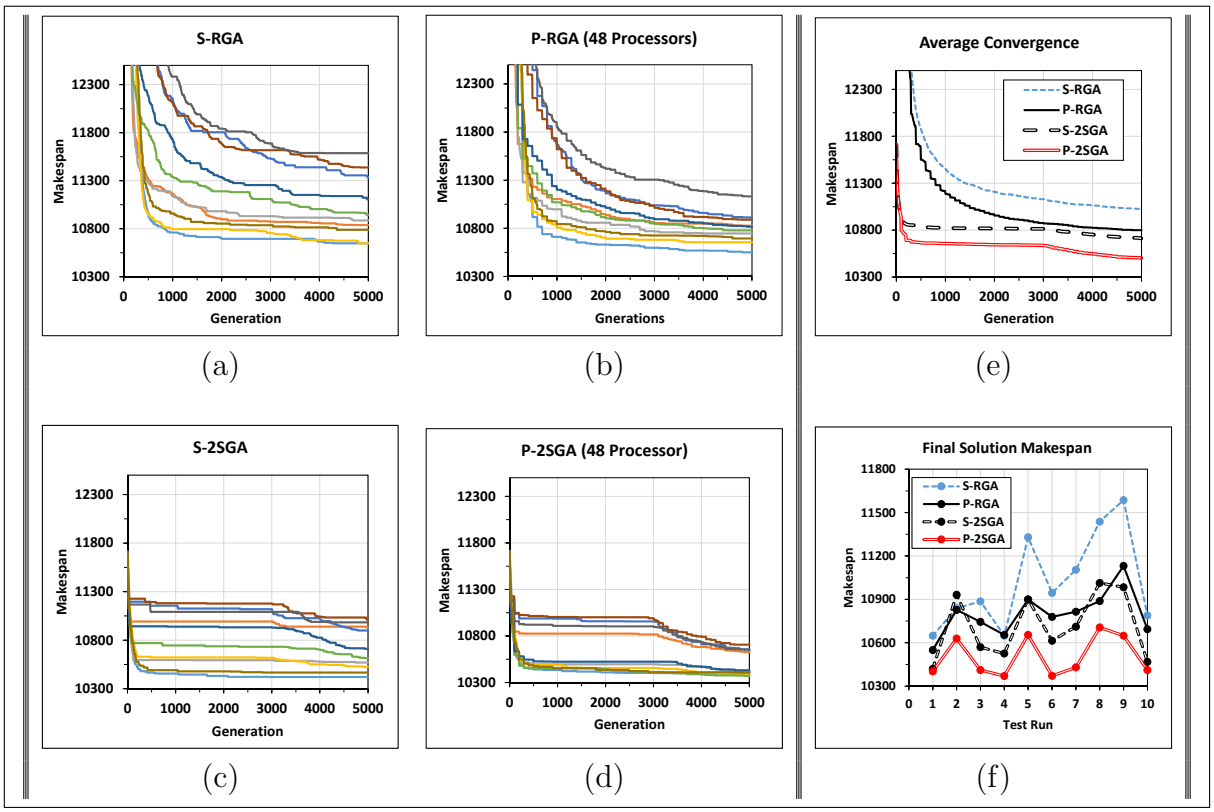


Figure 3.14: Comparison of Parallel RGA with Sequential 2SGA and further improvement of 2SGA using parallel computing.

3.5. Discussions and Conclusion

Regular genetic algorithms for FJSP attempt to determine both the assignment and the sequencing of the operations simultaneously and randomly through out the entire search process. In this research, we develop a Two-Stage Genetic Algorithm with the first stage being different from the regular approach. The first stage has a solution encoding that only dictates the sequence in which the operations are considered for assignment. Whenever an operation is considered for assignment, the machine that can complete this operation the soonest is selected while taking into account the operations that are already assigned. The order in which the operations are assigned determines

their sequence. The second stage, starting from the solutions of the first stage, follows the regular approach to enable the algorithm to search the neighborhood solution by including solutions that might have been excluded because of the greedy nature of the first stage. The algorithm was tailored to solve a comprehensive FJSP problem, which includes a sequence dependent setup time, attached or detached nature setups, machine release date, and lag time. Extensive numerical studies were conducted to assess the performance of the proposed algorithm. The findings of these numerical studies are summarized below.

Initial solution quality: The solution encoding and the corresponding decoding procedure proposed in this research enable the creation of high-quality initial solutions, which are by far better than those created using a specialized initialization technique that appeared in the literature. In the largest problem instance considered in this research, up to 49.47% average makespan reduction in the initial population was observed in using the proposed encoding/decoding procedure, whereas the specialized algorithm from literature resulted only in a 3.76% reduction.

Evaluation using benchmark problems: A total of 40 benchmark problems were solved using the proposed and regular genetic algorithms under 40 different parameter settings. In effect, each algorithm was executed for a total of 1600 test runs on the benchmark problems. The proposed algorithm was able to reach a better solution than the regular approach for 1177 times. Whereas the regular GA wins only 149 times, and both algorithms achieve the same solutions 274 times. The result clearly shows the superiority of the proposed algorithm in solving benchmark problems.

Evaluation using large-size problems: Methodologies that fare very well in small problem instances may not replicate their performance when problem sizes increase. In this study, using large-size problems having up to 140 jobs and 80 machines, we showed that the proposed algorithm not only converges very quickly but also it arrives at a better solution than the regular genetic algorithm. When solving the largest problem in this research, we observed that a solution that was obtained in less than 30 minutes using the proposed algorithm was not achieved after 72 hours of computation using the

regular genetic algorithm.

Impact of dividing the search into stages: The greedy nature of the first stage of the proposed algorithm enables it to find good solutions with minimal computation. The second stage, following the approach of the regular genetic algorithm, improves the solution of the first stage by searching neighborhood solutions that might have been excluded because of the greedy nature of the first stage. The impact of this phenomenon was observed in many test runs.

Comparison with parallel genetic algorithm: One of the very promising performance improvement strategies for a genetic algorithm is parallel computation. Numerical results showed that the performance of the proposed algorithm could be further improved using parallelization. However, the more interesting finding in our numerical study is that the sequential version of the proposed algorithm (with only one CPU) outperforms the parallel version of the regular genetic algorithm (with 48 CPUs) both in terms of convergence speed and final solution quality.

Table 3.8: Reported solutions for benchmark problems by Brandimarte (1993)

Problem	N	M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Reg GA	2SGA
MK01	10	6	40	40	41	40	40	40	40	40	40	40	40	40	40	40	40	40	42	40	40	40	40	42	41
MK02	10	6	26	29	28	26	32	26	26	26	26	26	26	26	26	26	26	26	28	26	26	26	26	27	27
MK03	15	8	204	N/A	204	204	N/A	204	204	204	204	204	204	204	204	204	204	204	204	204	204	204	204	204	204
MK04	15	8	60	67	67	60	67	60	60	60	60	62	60	60	60	61	60	60	75	60	60	60	60	68	63
MK05	15	4	173	176	177	173	188	172	173	173	173	172	173	173	172	173	172	172	179	173	172	172	172	175	173
MK06	10	15	58	67	61	63	85	58	63	59	60	65	58	60	60	65	59	58	69	60	57	59	57	71	64
MK07	20	5	144	147	154	139	154	139	140	140	141	140	139	140	139	140	139	139	149	139	139	139	139	144	141
MK08	20	10	523	523	523	523	523	523	523	523	523	523	523	523	523	523	523	523	555	523	523	523	523	559	523
MK09	20	10	307	320	321	311	N/A	307	312	307	307	310	307	307	307	311	307	307	342	307	307	307	307	336	311
MK10	20	15	198	229	219	212	N/A	197	214	207	208	214	197	205	208	225	202	205	242	202	198	202	197	240	205

Note Column (method) labels:
1 = Tabu Search(TS); 2 =Learnable Genetic Architecture(LEGA); 3 = Hierarchical Optimization(HO); 4 = Genetic Algorithm(GA); 5 = Multi-Agent Tabu Search Systems(MAS); 6 = Hybrid Genetic and Variable Neighborhood Descent Algorithm (HGVNA); 7 = Artificial Immune Algorithm (AIA); 8 = Variable Neighbourhood Search (VNS); 9 = Parallel Variable Neighborhood Search (PVNS); 10 = Hybrid Tabu Search (HTS); 11 = Discrepancy Search (DS); 12 = Hybrid Chaos Particle Swarm Optimization and GA (HAT); 13 = Artificial Bee Colony (ABC); 14 = Evolutionary Algorithm (EA); 15 = Hybrid Harmony Search (HHS); 16 = Hybrid Harmony and Large Neighborhood Search (HS); 17 = Heuristic; 18 = Two-Stage Artificial Bee Colony (TABC); 19 = Hybrid GA and Tabu Search (HGTS); 20 = Multi-objective Memetic Algorithms (MA2); 21 = Hybrid GA and Tabu Search ;

Table 3.9: Comparison between proposed 2SGA with regular GA and minimum and maximum of other methods

Problem	N	M	Min	Max	Reg GA	GA t.	2SGA	2SGA t.	Dev. from Min	Dev. from Max
MK01	10	6	40	42	42	00:02	41	00:02	2.44%	-2.44%
MK02	10	6	26	32	27	00:44	27	00:01	3.7%	-18.52%
MK03	15	8	204	204	204	00:15	204	00:01	0%	0%
MK04	15	8	60	75	68	02:30	63	00:50	4.76%	-19.05%
MK05	15	4	172	188	175	02:06	173	00:08	0.58%	-8.67%
MK06	10	15	57	85	71	06:34	64	00:12	10.94%	-32.81%
MK07	20	5	139	154	144	03:00	141	00:04	1.42%	-9.22%
MK08	20	10	523	555	523	04:46	523	00:06	0%	-6.12%
MK09	20	10	307	342	336	15:01	311	02:19	1.29%	-9.97%
MK10	20	15	197	242	240	13:03	205	05:53	3.9%	-18.05%

Table 3.10: General characteristics of comprehensive problems

			Number of operations per job		Number of alternative machines per operation	
Problem	No. of Jobs	No. of Machins	Min	Max	Min	Max
1	5	4	2	4	2	4
2	20	10	7	9	5	8
3	20	20	8	10	5	10
4	50	20	10	15	5	10
5	70	30	12	20	7	12
6	100	30	15	20	7	12
7	100	50	15	30	7	15
8	140	80	20	40	4	15

Table 3.11: Comparison of 2SGA and RGA in solving very large problems

Problem 7 (100x50)				Problem 8 (140x80)			
2SGA		RGA		2SGA		RGA	
Time	Makespan	Time	Makespan	Time	Makespan	Time	Makespan
00:00:00	14557	00:00:00	29921	00:00:00	17383	00:00:00	36731
00:02:20	13882	00:04:20	28778	00:07:14	17003	01:23:50	36731
00:04:45	13576	00:10:03	22261	00:14:48	16578	01:28:56	36483
00:10:16	13113	00:12:22	20506	00:26:14	16048	01:44:13	34844
00:15:19	12858	00:15:09	18501	00:28:07	15955	02:33:06	29581
00:20:15	12698	00:20:14	16618	01:01:45	15195	04:02:31	23779
00:30:15	12556	00:25:02	15370	01:28:31	14879	05:01:32	21376
00:37:01	12537	00:32:42	14577	02:01:46	14581	09:07:17	17396
00:36:37	12555	00:45:08	14328	02:30:21	14455	11:14:49	17025
03:18:20	12535	01:00:04	14171	03:00:58	14381	20:07:39	16571
03:42:01	12532	01:30:11	14012	04:03:50	14304	67:33:23	16049
04:31:24	12528	02:00:31	13938	06:00:18	14233	71:57:29	16009
*	*	02:15:24	13902	12:00:56	14229	*	*
*	*	03:33:39	13895	10:19:20	14229	*	*
*	*	03:51:51	13882	30:58:00	14228	*	*
*	*	04:04:29	13830	33:04:34	14225	*	*
*	*	04:41:06	13757	37:24:18	14225	*	*

* The termination criterion was meet.

Chapter 4

Multi-Objective Lot Streaming Extension

4.1. Introduction

Chapter 3 presented an efficient Two-Stage Genetic Algorithm that we developed initially for a classic FJSP (published in Rooyani and Defersha (2019)) and then applied it on a comprehensive FJSP that incorporates (1) sequence dependent setup time, (2) attached and detached nature of setup, (3) machine release date, and (4) lag time (published in Defersha and Rooyani (2020)). As it was described, the high performance of the two-stage algorithm was achieved by a systematically designed solution representation and a greedy decoding mechanism of the first stage. This approach enables the algorithm to find highly improved solutions from the get-go that rapidly converge to promising regions of the search space. The second stage removes the greedy nature of the first stage and follows the regular approach of a genetic algorithm for FJSP and attempts to improve the solutions found in the first stage. The superiority of the Two-Stage Genetic Algorithm specifically in solving very large-size problems was demonstrated by solving the examples of up to 80 machines and 140 jobs. In this chapter, we extend the application of the algorithm of chapter 3 to solve a lot streaming problem in FJSP that appeared in Defersha and Chen (2012a) while at the same time expanding the problem

to incorporate multiple objective functions. The objective function terms included are the minimization of (1) makespan, (2) maximum sublot flowtime, (3) total sublot flowtime, (4) maximum job flowtime, (5) total job flowtime, (6) maximum sublot finish-time separation, (7) total sublot finish-time separation, (8) maximum machine load, (9) total machine load, and (10) maximum machine load difference. The result of this research is published in Rooyani and Defersha (2022).

In addition to expanding the single objective lot streaming FJSP to a multi-objective one and customizing the Two-Stage Genetic Algorithm to solve it, an added contribution of this research is its provision of many numerical studies. At the outset of the numerical studies, all ten objective function terms are illustrated using a small prototype problem. The importance of multi-objective optimization in small versus large size problems is examined and contrasted. The capability of the Two-Stage Genetic Algorithm to jointly optimize all the objective function terms is evaluated. The need to optimize both the maximum and the total of a performance measure (such as flowtime) was examined. The relevance of two newly proposed objective function terms (sublot finish-time separation and maximum workload difference) in providing better solution quality is assessed. The quality of the initial population and the convergence behavior of the Two-Stage Genetic Algorithm is contrasted against the regular genetic algorithm with respect to each of the objective function terms. Further algorithm enhancement through high-performance parallel computation is considered. Algorithm components and parameters are empirically studied. In particular, three different selection operators are examined, and the Analysis of Variance (ANOVA) on mutation and crossover probabilities is conducted.

The remainder of this chapter is organized as follows. The proposed multi-objective FJSP lot streaming model is presented in Section 4.2. The adaptation of the Two-Stage Genetic Algorithm to solve the multi-objective lot streaming model is detailed in Section 4.3. Section 4.4 provides extensive numerical studies. Conclusion, discussion and future research are in Section 4.5.

4.2. Mathematical Modeling

4.2.1. The Basic Problem

The main objective of this work is to expand the single objective FJSP lot streaming model presented in Defersha and Chen (2012a) to a multi-objective one and develop a Two-Stage Genetic Algorithm based on the work in Defersha and Rooyani (2020). However, for a better comprehension, we first present the basic single-objective problem and its mathematical model as presented in Defersha and Chen (2012a).

Problem description and notations

Consider a job shop consisting of M machines where machines with common functionalities are grouped into a department (e.g., turning machines in a turning department). Assume that the system is currently processing jobs from the previous schedule, and each machine m (where $m = 1, \ \ldots, \ M$) has a release date D_m at which time it will be available for the next scheduling. Consider also a total of J independent jobs to be scheduled next in the system where a job is a batch of identical parts. The number of parts in a batch of job j (where $j = 1, \ \ldots, \ J$) is given by B_j, and this batch is to be split into S_j number of unequal sublots (transfer batches). A decision variable $b_{s,j}$ is used to denote the size of sublot s (where $s = 1, \ \ldots, \ S_j$) of job j. Each sublot of job j is to undergo O_j number of operations in a fixed sequence such that each operation o (where $o = 1, \ \ldots, \ O_j$) can be processed by one of several eligible machines. $T_{o,j,m}$ is unit-processing-time for operation o of job j on machine m. Operation o of a sublot of job j can be started on an eligible machine m after lag time $L_{o,j}$ and after the setup is performed. The lag time $L_{o,j}$ is a waiting time that may be required either for cooling, drying, or for some other purpose. The setup time for an operation o of job type j on machine m depends on the preceding operations and is denoted by $S_{o,j,m,o',j'}$, where operation o' of a sublot of job j' is the preceding operation on machine m. If operation o of sublot s of job j is the first operation to be processed on machine m, the setup

time is represented as $S^*_{o,j,m}$. The setup time $S_{o,j,m,o',j'}$ (or $S^*_{o,j,m}$) for operation o of a sublot of job j can be overlapped with the processing time of operation $o-1$ of the same sublot if the setup is a detached setup and machine m is available for setup. The problem is to determine the size of each sublot, assign the operation of each sublot to one of the eligible machines and determine the sequence and starting time of the assigned operations on each machine. The objective is to minimize the makespan of the schedule. We next introduce some additional notations and then present a mixed-integer linear programming (MILP) formulation for Flexible Assembly Job-shop Scheduling Problem with Lot Streaming (FJSP-LS).

Additional Parameters:

R_m Maximum number of production runs of machine m where production runs are indexed by r or $u = 1, 2, ..., R_m$; Each of these production runs can be assigned to at most one operation of one sublot. Thus the assignment of the operations to production runs of a given machine determines the sequence of the operations on that machine;

$P_{o,j,m}$ A binary data equal to 1 if operation o of job j can be processed on machine m, 0 otherwise;

$A_{o,j}$ A binary data equal to 1 if the setup of operation o of of job j is attached (non-anticipatory), or 0 if this setup is detached (anticipatory);

Ω Large positive number.

Variables:

Continuous Variables:

c_{max} Makespan of the schedule

$c_{o,s,j}$ Completion time of operation o of sublot s of job j;

$\widehat{c}_{r,m}$ Completion time of the r^{th} run of machine m;

$b_{s,j}$ Size of sublot s of job j

Binary Integer Variables:

$x_{r,m,o,s,j}$ A binary variable which takes the value 1 if the r^{th} run on machine m is for operation o of sublot s of job j, 0 otherwise;

$y_{r,m,o,j}$ A binary variable which takes the value 1 if the r^{th} run on machine m is for operation o of any one of the sublots of job j, 0 otherwise;

$\gamma_{s,j}$ A binary variable that takes the value 1 if sublot s of job j is non-zero ($b_{s,j} \geq 1$), 0 otherwise,

$z_{r,m}$ A binary variable that takes the value 1 if the r^{th} potential run of machine m has been assigned to an operation, 0 otherwise;

MILP model for FJSP-LS

Following the problem description and using the notations given above, the MILP mathematical model for the FJSP-LS is presented below.

Minimize:

$$Objective = c_{max} \tag{4.1}$$

Subject to:

$$c_{max} \geq c_{o,s,j} \ ; \quad \forall(o,s,j) \tag{4.2}$$

$$\widehat{c}_{r,m} \geq c_{o,s,j} + \Omega \cdot x_{r,m,o,s,j} - \Omega \ ; \quad \forall(r,m,o,s,j) \tag{4.3}$$

$$\widehat{c}_{r,m} \leq c_{o,s,j} - \Omega \cdot x_{r,m,o,s,j} + \Omega \ ; \quad \forall(r,m,o,s,j) \tag{4.4}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^{*}_{o,j,m} - \Omega \cdot x_{1,m,o,s,j} + \Omega \geq D_m \ ; \quad \forall(m,o,s,j) \tag{4.5}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j}) + 2\Omega \ \geq \widehat{c}_{r-1,m} \ ; \quad \forall(r,m,o,s,j,o',j') | (r > 1) \tag{4.6}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \Omega \cdot (x_{1,m,o,s,j} + x_{r',m',o-1,s,j}) + 2\Omega \geq \widehat{c}_{r',m'} + L_{o,j}; \quad \forall (m, r', m', o, s, j) | \{((1, m) \neq (r', m')) \wedge (o > 1)\} \tag{4.7}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j}) + 3\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \; ; \forall (r, m, r', m', o, s, j, o', j') | \{(r > 1) \wedge (o > 1) \wedge (r, m) \neq (r', m') \wedge (o, j) \neq (o', j')\} \tag{4.8}$$

$$y_{r,m,o,j} \leq P_{o,j,m} \;\; ; \quad \forall (r, m, o, j) \tag{4.9}$$

$$y_{r,m,o,j} = \sum_{s=1}^{S_j} x_{r,m,o,s,j} \;\; ; \quad \forall (r, m, o, j) \tag{4.10}$$

$$\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,s,j} = \gamma_{s,j} \;\; ; \quad \forall (o, s, j) \tag{4.11}$$

$$b_{s,j} \leq B_j \cdot \gamma_{s,j} \;\; ; \quad \forall (s, j) \tag{4.12}$$

$$\gamma_{s,j} \leq b_{s,j} \;\; ; \quad \forall (s, j) \tag{4.13}$$

$$\sum_{s=1}^{S_j} b_{s,j} = B_j \;\; ; \quad \forall (j) \tag{4.14}$$

$$\sum_{j=1}^{J} \sum_{s=1}^{S_j} \sum_{o=1}^{O_j} x_{r,m,o,s,j} = z_{r,m} \;\; ; \quad \forall (r, m) \tag{4.15}$$

$$z_{r+1,m} \leq z_{r,m} \;\; ; \quad \forall (r, m) \tag{4.16}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j}; \quad \forall (r, r', m, o, o', s, j) | \{(o' > o) \wedge (r' < r)\} \tag{4.17}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j}; \quad \forall (r, r', m, o, o', s, j) | \{(o' < o) \wedge (r' > r)\} \tag{4.18}$$

$$x_{r,m,o,s,j}, \;\; y_{r,m,o,j}, \;\; \gamma_{s,j} \text{ and } z_{r,m} \text{ are binary} \tag{4.19}$$

The complete description and the meanings of the objective function in Eq. (4.1) and the constraints in Eqs. (4.2)-(4.19) can be found in Defersha and Chen (2012a).The expansion of this single objective FJSP lot streaming model into a multi-objective one is presented in the following section.

4.2.2. Multi-Objective Model for FJSP-LS

As it was stated previously, one of the objective of this work is to expand the single objective FJSP lot streaming presented in Defersha and Chen (2012a) to a multi objective approach. In this section, we present notations of additional continuous variables and the MILP formulation of the proposed multi-objective FJSP scheduling with lot streaming.

Additional Continuous Variables

The definitions of the additional continuous variables is given below. Further explanations for some of the variable definitions are also given as we discus the equations that uses those variables.

$e_{s,j}$ Entry time of sublot s of job j;

$\hat{e}_j$ Entry time of job j (minimum of $e_{s,j}$ for all s of job j);

$d_{s,j}$ Departure time of sublot s of job j;

$\hat{d}_j$ Departure time of job j (maximum of $d_{s,j}$ for all s of job j);

$f_{s,j}$ Flowtime of sublot s of job j;

$\hat{f}_j$ Flowtime of job j;

f_{max} Maximum sublot flowtime;

$\hat{f}_{max}$ Maximum job flowtime;

f_{total} Total sublot flowtime;

$\hat{f}_{total}$ Total job flowtime;

$\hat{g}_j$ Minimum sublot departure time of job j;

$\hat{h}_j$ Sublot finish separation time of job j;

$\hat{h}_{max}$ Maximum sublot finish separation time;

$\hat{h}_{total}$ Total sublot finish separation time;

$l_{m,o,s,j}$ Workload on machine m because of the setup and processing of operation o of sublot s of job j;

$\hat{l}_m$ Workload on machine m;

$\hat{l}_{min}$ Minimum machine workload;

$\hat{l}_{max}$ Maximum machine workload;

$\hat{l}_{total}$ Total machine workload;

$\hat{l}_{diff}$ Maximum machine workload difference;

Objective Functions and Additional Constraints

The objective of the proposed multi-objective model is to minimize the function given in Eq. (4.20) subject to the constraints in the original model in Eqs. (4.2) to (4.19) and newly added constraints in Eqs. (4.21) to (4.56). The objective function terms and the additional constraints are discussed in the following sections.

$$\textbf{Minimize: } Z_i \ \forall i \ \in \{1, 2, \cdots, 10\}. \tag{4.20}$$

Makespan (Z_1)

Makespan is defined as the maximum completion time of a given schedule. Its minimization is a widely used objective function in scheduling research. The essence of minimizing makespan is to finish production as soon as possible to expedite delivery of products to customers and/or to quickly free up resources for the upcoming production and other tasks such as development and maintenance. The first objective function (Z_1) of the proposed multi-objective model is makespan (c_{max}) as shown in Eq. (4.21).

$$Z_1 = c_{max} \tag{4.21}$$

Maximum and Total Sublot Flowtime (Z_2 and Z_3)

The entry time ($e_{s,j}$) to the shop floor of a sublot of a job is the time the setup of its first operation begins if the setup is attached. If the setup of the first operation is detached, $e_{s,j}$ is the time at which the actual processing of the first operation begins as setup can be completed before the raw material is admitted to the shop floor. The constraints in Eqs. (4.22) to (4.25) are used to set the value of this variable. The departure time of the sublot ($d_{s,j}$) is simply the completion time of the last operation of the sublot as enforced by the constraint in Eq. (4.26). The flowtime of sublot s of job j, denoted as $f_{s,j}$, is the interval between the time the sublot enters the shop floor to the time its last operation is finished. Its value is set by the constraint in Eq. (4.27). The constraint in Eq. (4.28) along with the objective function will enforce f_{max} to assume the maximum flowtime of all the sublots ($\max_{\forall(s,j)} f_{s,j}$). The total flowtime of all the sublots is calculated by the constraint in Eq. (4.29). The objective function terms Z_2 and Z_3 are the values of f_{max} and f_{total} as shown in Eqs. (4.30) and (4.31), respectively. The minimization of flowtime can lead to stable or uniform utilization of resources and a rapid turn-around of jobs, and it is particularly important in real-life situations where reducing inventory or holding cost is of primary concern (Sang and Duan, 2012).

$$e_{s,j} \geq c_{1,s,j} - b_{s,j} \cdot T_{1,j,m} - S^*_{1,j,m} \cdot A_{1,j} - \Omega \cdot (1 - x_{1,m,1,s,j}); \quad \forall(s,j,m) \tag{4.22}$$

$$e_{s,j} \leq c_{1,s,j} - b_{s,j} \cdot T_{1,j,m} - S^*_{1,j,m} \cdot A_{1,j} + \Omega \cdot (1 - x_{1,m,1,s,j}); \quad \forall(s,j,m) \tag{4.23}$$

$$\begin{aligned} e_{s,j} \geq\ & c_{1,s,j} - b_{s,j} \cdot T_{1,j,m} \\ & - S_{1,j,m,o',j'} \cdot A_{1,j} - 2\Omega \cdot (1 - x_{r,m,1,s,j} - y_{r-1,m,o',j'}); \quad \forall(s,j,r,m) | r > 1 \end{aligned} \tag{4.24}$$

$$\begin{aligned} e_{s,j} \leq\ & c_{1,s,j} - b_{s,j} \cdot T_{1,j,m} \\ & - S_{1,j,m,o',j'} \cdot A_{1,j} + 2\Omega \cdot (1 - x_{r,m,1,s,j} - y_{r-1,m,o',j'}); \quad \forall(s,j,r,m) | r > 1 \end{aligned} \tag{4.25}$$

$$d_{s,j} = c_{O_j,s,j}; \quad \forall(s,j) \tag{4.26}$$

$$f_{s,j} = d_{s,j} - e_{s,j}; \quad \forall(s,j) \tag{4.27}$$

$$f_{max} \geq f_{s,j}; \quad \forall(s,j) \tag{4.28}$$

$$f_{total} = \sum_{j=1}^{J} \sum_{s=1}^{S_j} f_{s,j} \tag{4.29}$$

$$Z_2 = f_{max} \tag{4.30}$$

$$Z_3 = f_{total} \tag{4.31}$$

Maximum and Total Job Flowtime (Z_4 and Z_5)

In the presence of lot streaming, the entrance time $\hat{e}_j$ and the departure time $\hat{d}_j$ of a job are the smallest and the largest entrance times of all its sublots, $\min_{\forall s|\gamma_{s,j}=0}\{e_{s,j}\}$ and $\max_{\forall s|\gamma_{s,j}=0}\{d_{s,j}\}$, respectively. The values of these variables are set by the constraints in Eqs. (4.32) and (4.33), and the objective function. The flowtime of a job, $\hat{f}_j$, is the difference $\hat{d}_j - \hat{e}_j$ as enforced by the constraint in Eq. (4.34). The constraint in Eq. (4.35) along the objective function enforce $\hat{f}_{max}$ to assume the maximum flowtime of all the jobs, $\max_{\forall j}\{\hat{f}_j\}$. The total job flowtime ($\hat{f}_{total}$) is evaluated by the constraint in *Eq.*(4.36). The values $\hat{f}_{max}$ and $\hat{f}_{total}$ correspond to the fourth and fifth terms, Z_4 and Z_5, of the objective function and their values are enforced by the constraints in Eqs. (4.37) and (4.38), respectively.

$$\hat{e}_j \leq e_{s,j} + \Omega(1 - \gamma_{s,j}); \quad \forall(s,j) \tag{4.32}$$

$$\hat{d}_j \geq d_{s,j} - \Omega(1 - \gamma_{s,j}); \quad \forall(s,j) \tag{4.33}$$

$$\hat{f}_j = \hat{d}_j - \hat{e}_j; \quad \forall j \tag{4.34}$$

$$\hat{f}_{max} \geq \hat{f}_j; \quad \forall j \tag{4.35}$$

$$\hat{f}_{total} = \sum_{j=1}^{J} \hat{f}_j \tag{4.36}$$

$$Z_4 = \hat{f}_{max} \tag{4.37}$$

$$Z_5 = \hat{f}_{total} \tag{4.38}$$

Maximum and Total Sublot finish-time Separation (Z_6 and Z_7)

In lot streaming, sublots are treated independently. As a result, one sublot of a job may be finished much sooner than the other sublot of the same job. This may increase work-in-process inventory as the entire job can not be made available for shipment or assembly within a reasonable time window. Hence, in this research, we introduce an objective function to minimize the gap between the earliest and the latest finish-times of sublots of the same job. In doing so, first we defined a variable $\hat{g}_j$ that assumes the earliest finish-time among all the sublots of a job, $\min_{\forall(s,j)|\gamma_{s,j}=1}\{d_{s,j}\}$, as enforced by the constraint in Eq. (4.39) and the objective function. The latest finish-time of the sublots of a job is its departure time $\hat{d}_j$ that was discussed previously. With these variables defined, the sublot finish separation time of a job, $\hat{h}_j$, is the difference $\hat{d}_j - \hat{g}_j$, enforced by the constraint in Eq. (4.40). The constraint in Eq. (4.41) and the objective function will enforce $\hat{h}_{max}$ to assume the value $\max_{\forall j}\{\hat{h}_j\}$. The total sublot finish time separation $\hat{h}_{total}$ is evaluated using the constraint in Eq. (4.42). The objective function terms Z_6 and Z_7 correspond to the values of $\hat{h}_{max}$ and $\hat{h}_{total}$, respectively, as enforced by the constraints in Eqs. (4.43) and (4.44).

$$\hat{g}_j \leq d_{s,j} + \Omega(1 - \gamma_{s,j}); \quad \forall(s,j) \tag{4.39}$$

$$\hat{h}_j = \hat{d}_j - \hat{g}_j; \quad \forall(j) \tag{4.40}$$

$$\hat{h}_{max} \geq \hat{h}_j; \quad \forall(j) \tag{4.41}$$

$$\hat{h}_{total} = \sum_{j=1}^{J} \hat{h}_j \tag{4.42}$$

$$Z_6 = \hat{h}_{max} \tag{4.43}$$

$$Z_7 = \hat{h}_{total} \tag{4.44}$$

Maximum Workload, Total Workload and Maximum Workload-Difference (Z_8, Z_9 and Z_{10})

In addition to makespan and flowtime, two other objectives commonly considered in FJSP scheduling are the minimization of maximum and total machine workload. They represent the intention of protecting machines from overuse (Chiang and Lin, 2013). Moreover, in this research, we noted that in the presence of alternative routing and sequence dependent setup time, these objectives could result in a substantially reduced overall system workload with a moderate increase in makespan. This can significantly free up machine operators for other activities such as quality improvement, development, and maintenance. The necessary variables and constraints to impose these categories of objective functions are discussed below.

The workload on machine m (i.e., $l_{m,o,s,j}$) because of an assigned operation o of sublot s of job j comprises the setup and the actual processing of the operation. The value of this variable is assigned by the constraints in Eqs. (4.45) to (4.48). The overall workload on machine m, ($\hat{l}_m$), comprises the workloads because of all the operations assigned to it from the current schedule and its release date D_m as shown in Eq. (4.49). The release date may represent the amount of work that spills into the current planning and scheduling period from the previous one. The objective function along with the constraints in Eqs. (4.50) and (4.51) set the values of $\hat{l}_{max} = \max_{\forall m}\{l_m\}$ and $\hat{l}_{min} = \min_{\forall m}\{l_m\}$, respectively. The workload difference between the maximally and the least loaded machines (maximum workload difference, $\hat{l}_{diff}$) is calculated using the constraint in Eq. (4.52). The total workload on the system $\hat{l}_{total}$ is evaluated by the constraint in Eq. (4.53). The objective function terms Z_8, Z_9, and Z_{10} represent the values of $\hat{l}_{max}$, $\hat{l}_{total}$, and $\hat{l}_{diff}$ as enforced by the constraints in Eqs. (4.54), (4.55) and (4.56), respectively.

$$l_{m,o,s,j} \geq S^*_{o,j,m} + b_{s,j} \cdot T_{o,j,m} - \Omega \cdot (1 - x_{1,m,o,s,j}); \quad \forall (m,o,s,j) \tag{4.45}$$

$$l_{m,o,s,j} \leq S^*_{o,j,m} + b_{s,j} \cdot T_{o,j,m} + \Omega \cdot (1 - x_{1,m,o,s,j}); \quad \forall (m,o,s,j) \tag{4.46}$$

$$l_{m,o,s,j} \geq S_{o,j,m,o',j'} + b_{s,j} \cdot T_{o,j,m} - 2\Omega \cdot (1 - x_{r,m,o,s,j} - y_{r-1,m,o',j'}); \quad \forall (r,m,o,s,j) | r > 1 \tag{4.47}$$

$$l_{m,o,s,j} \leq S_{o,j,m,o',j'} + b_{s,j} \cdot T_{o,j,m} + 2\Omega \cdot (1 - x_{r,m,o,s,j} - y_{r-1,m,o',j'}); \quad \forall(r,m,o,s,j)|r > 1 \tag{4.48}$$

$$\hat{l}_m = D_m + \sum_{j=1}^{J} \sum_{s=1}^{S_j} \sum_{o=1}^{O_j} l_{m,o,s,j}; \quad \forall(m) \tag{4.49}$$

$$\hat{l}_{max} \geq l_m; \quad \forall(m) \tag{4.50}$$

$$\hat{l}_{min} \leq l_m; \quad \forall(m) \tag{4.51}$$

$$\hat{l}_{diff} = l_{max} - l_{min}; \quad \forall(m) \tag{4.52}$$

$$\hat{l}_{total} = \sum_{m=1}^{M} l_m; \tag{4.53}$$

$$Z_8 = \hat{l}_{max} \tag{4.54}$$

$$Z_9 = \hat{l}_{total} \tag{4.55}$$

$$Z_{10} = \hat{l}_{diff} \tag{4.56}$$

4.3. Genetic Algorithm

4.3.1. Prototype Problem

To illustrate the solution representation and the various genetic operators, a prototype problem that consists of the processing of four jobs using five machines is considered. The complete data sets for this small problem are given in Tables 4.1 and 4.2. Data related to batch size (B_j), nature of setup being attached or detached ($A_{o,j}$), lag time ($L_{o,j}$) and alternative routing ($m, T_{o,j,m}$) for each operation are in Table 4.1. Sequence-dependent setup time data is provided in Table 4.2. This problem is also used in the numerical example to illustrate the various objective function terms of the proposed model.

Table 4.1: Data for Jobs for Problem-1

						(Eligible machine, Processing Time) = (m, $T_{o,j,m}$)			
j	B_j	S_j	o	$A_{o,j}$	$L_{o,j}$	i	ii	iii	iv
1	100	2	1	0	0	(1, 6.75)	(4, 6.50)	(5, 6.50)	
			2	1	120	(1, 3.00)	(2, 2.25)	(4, 2.75)	
			3	0	120	(1, 3.50)	(2, 3.25)	(4, 3.75)	(5, 3.50)
2	250	3	1	0	0	(1, 1.75)	(2, 2.00)	(5, 1.25)	
			2	1	0	(2, 5.00)	(3, 4.25)	(4, 5.00)	(5, 4.75)
			3	1	40	(1, 7.00)	(2, 7.00)	(3, 6.50)	(5, 6.50)
			4	0	40	(1, 2.50)	(2, 2.50)	(3, 2.75)	(4, 2.75)
3	200	3	1	0	0	(1, 5.25)	(5, 5.75)		
			2	1	0	(1, 4.50)	(3, 4.25)	(5, 4.25)	
			3	1	0	(1, 3.50)	(2, 3.50)		
4	100	2	1	0	0	(4, 6.00)	(5, 6.00)		
			2	0	0	(1, 4.25)	(3, 4.75)	(4, 4.75)	(5, 4.75)
			3	1	0	(2, 2.00)	(4, 1.25)	(5, 1.25)	

Machine release dates in minutes: $D_1 = 840$, $D_2 = D_3 = D_5 = 0$, $D_4 = 120$.

4.3.2. Solution Encoding

A solution encoding is a technique of transforming a problem statement into a searchable space of all feasible solutions, in which an algorithm can be applied to explore iteratively for optimal solutions. Hence, its design is the first most crucial step in solving a problem using a search-based algorithm. The solution encoding used in this work combines features from the solution representations in Defersha and Chen (2012a) for FJSP lot streaming and that in Defersha and Rooyani (2020) for dividing the genetic search into two stages. This solution encoding is depicted in Figure 4.1 for a typical solution of the prototype problem presented in the previous section. As shown in Figure 4.1-(a), the solution representation has two segments. The first segment (Segment-1), detailed in Figure 4.1-(b), encodes the numbers and sizes of sublots for all the jobs. The number of genes in this segment is equal to the sum of the maximum number of sublots of each job ($\sum_{j=1}^{J} S_j$), where there are S_j genes corresponding to each job. The gene $\alpha_{j,s}$ takes

a continuous value from the interval [0, 1]. The decoding procedure for the number and sizes of the sublots from Segment-1 is detailed in Section 4.3.3.

The second segment (Segment-2) of the solution encoding has two forms. The first form, detailed in Figure 4.1-(c), is applicable for the first stage of the search by the genetic algorithm. The number of genes in this segment is equal to the total number of operations in all the sublots, which can be computed as $\sum_{j=1}^{J} \sum_{o=1}^{O_j} S_j \times O_j$. Each gene is a 3-tuple $[j, s, o]$ composed of job, sublot and operation indices. For a particular $[j, s]$, there are O_j number of genes corresponding to each operation of the sublot, and a gene $[j, s, o]$ appears in the segment earlier than $[j, s, o']$ if $o < o'$. This segment provides the order (left to right) in which the operations are considered for assignment and sequencing. Whenever an operation of a sublot of a given job is to be assigned to a machine, the algorithm chooses the machine that completes the operation sooner after completing the operations previously assigned to this machine. In that case, the order in which the operations are assigned to machines represents their processing sequence.

The second form of Segment-2, detailed in Figure 4.1-(d), is for the second stage of the genetic search. This form of the segment explicitly encodes both the assignment and sequencing of the operations on the machines. Each gene, in this form, is 4-tuple $[j, s, o, m]$ where m encodes the machine assignment for operation $[j, s, o]$ and it is restricted to take the value such that $P_{j,o,m} = 1$. Moreover, for a given $[j, s]$, the gene $[j, s, o, m]$ appeared earlier the sequence than $[j, s, o', m']$ if $o < o'$. The sequence of the operation on a given machine m is dictated by the order in which the genes appeared on Segment-2. For instance, the assignment and the sequence of the operation on machine $m = 4$ is $(j1, s2, o1) \rightarrow (j2, s1, o2) \rightarrow (j4, s1, o2) \rightarrow (j1, s1, o2) \rightarrow (j2, s1, o4)$. The detail discussion of the decoding of Segment-2 under the first and the second stage of the genetic search is given in Section 4.3.3.

Segment-1	Segment-2
Encodes the numbers and sizes of the sublots	Encodes the assignment and sequencing of the operations

(a) General strucutre of the solution representation

Details of Segment-1 (Applicable both in Stage-1 and Stage-2 of the GA)									
Job-1		Job-2			Job-3			Job-4	
$\alpha_{1,1}$	$\alpha_{1,2}$	$\alpha_{2,1}$	$\alpha_{2,2}$	$\alpha_{2,3}$	$\alpha_{3,1}$	$\alpha_{3,2}$	$\alpha_{3,3}$	$\alpha_{4,1}$	$\alpha_{4,2}$

(b) Details for Segment-1, where a gene $\alpha_{j,s}$ takes a continous value from the interval [0, 1].

Details of Segment-2 (Applicable only in Stage-1 of the GA)																														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
3, 1, 1	4, 1, 1	1, 2, 1	1, 1, 1	3, 2, 1	2, 2, 1	2, 1, 1	3, 3, 1	4, 2, 1	2, 1, 2	2, 1, 3	3, 1, 2	4, 1, 2	3, 1, 3	2, 2, 2	2, 2, 3	2, 2, 4	2, 3, 1	4, 1, 3	3, 2, 2	4, 2, 2	3, 2, 3	4, 2, 3	1, 1, 2	1, 1, 3	2, 3, 2	3, 3, 2	3, 3, 3	1, 2, 2	2, 3, 3	1, 2, 3	2, 1, 4	2, 3, 4

(c) Details for Segment-2 in Stage-1, where a gene takes a value [j, s, o].

Details of Segment-2 (Applicable only in Stage-2 of the GA)																														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
3, 1, 1, 5	4, 1, 1, 5	1, 2, 1, 4	1, 1, 1, 1	3, 2, 1, 5	2, 2, 1, 1	2, 1, 1, 1	3, 3, 1, 1	4, 2, 1, 5	2, 1, 2, 4	2, 1, 3, 3	3, 1, 2, 3	4, 1, 2, 4	3, 1, 3, 1	2, 2, 2, 3	2, 2, 3, 1	2, 2, 4, 3	2, 3, 1, 1	4, 1, 3, 5	3, 2, 2, 5	4, 2, 2, 5	3, 2, 3, 2	4, 2, 3, 5	1, 1, 2, 4	1, 1, 3, 1	2, 3, 2, 5	3, 3, 2, 3	3, 3, 3, 2	1, 2, 2, 2	2, 3, 3, 1	1, 2, 3, 5	2, 1, 4, 4	2, 3, 4, 2

(d) Details for Segment-2 in Stage-2, where a gene takes a value [j, s, o, m].

Figure 4.1: Solution representation

4.3.3. Solution Decoding

Number and Size of Sublots

The decoding of the number and sizes of sublots from Segment-1 is similar to that discussed in Defersha and Chen (2012a). Given the values of the genes in Segment-1, the size of a sublot $b_{s,j}$ can be computed using Eq. (4.57). Once all the $b_{s,j}$'s are calculated, a sublot whose size is less than a minimum threshold value is set to zero, and the corresponding gene $\alpha_{s,j}$ is also set to zero. And then, the sizes of the other sublots are reevaluated using the same equation (Eq. (4.57)). In this decoding, the number of the sublots for a given job is equal to the number of sublots whose sizes are greater than zero.

$$b_{s,j} = \begin{cases} \frac{\alpha_{s,j}}{\sum_{s=1}^{S_j} \alpha_{s,j}} & \text{if } \sum_{s=1}^{S_j} \alpha_{s,j} > 0 \\ \\ B_j/S_j & \text{Otherwise} \end{cases} \tag{4.57}$$

Assignment, Sequencing and Completion Time

Once the sizes of all the sublots are known (see Section 4.3.3), the assignment, sequencing, and completion times of the operation of each non-zero sublot and other variables are determined using the information obtained from Segment-2 and two decoding procedures outlined in this section. The first decoding procedure is applicable for Stage-1 of the genetic search, while the second is for Stage-2.

Stage-1

In Stage-1 of the genetic search, the assignment and sequencing of the operations and the determination of their starting and finish times are obtained using a procedure that utilizes the information in the first form of Segment-2 of the solution representation (Figure 4.1-(c)). In describing this procedure, let us first define GeneS2F1[l] to denote the content of a gene $[j, s, o]$ in the first form of Segment-2 at location l, where l runs from 1 to the total number of genes in this segment. Moreover, let us define r_m as a run counter for machine m, which increases by one every time an operation is assigned to the machine. With this definition, the steps for the determination of the assignment and sequencing of operations in Stage-1 of the search are outlined in Figure 4.2 along with the procedure described in Figure 4.4 to evaluate the decision variables $c_{o,s,j,m}$, $l_{m,o,s,j}$, $e_{s,j}$, and $d_{s,j}$. In Step-1, the counters l and r_m are initialized to 1 and 0, respectively. The values of the indices j, s, and o are obtained from GeneS2F1[l] at Step-2. In Step-3, if $b_{s,j}$ is zero, the algorithm move to Step-9. Otherwise, it advances to Step-4. In these steps, the counter r_m is temporarily increased by 1 corresponding to all the eligible machines for operation o of job j. And then, using the procedure outlined in Figure 4.4, the variables $c_{o,s,j,m}$, $l_{m,o,s,j}$, $e_{s,j}$, and $d_{s,j}$ are evaluated corresponding to all

these eligible machines. In Step-5, the machine that finishes operation $[o, s, j]$ with the smallest $c_{o,s,j,m}$ is selected, and in Step-6, the operation is assigned to the r_m^{th} run of this machine. In Step-7, the values of the decision variable calculated corresponding to the selected machine are retained as final values. The values of the counter r_m, that was temporarily increased in Step-4, are reduced by 1 corresponding to those machines that are not selected to process operation $[o, s, j]$. In Step-9, the algorithm stops if all the operations are assigned, or otherwise, it increases the counter l by one and then returns to Step-2.

Step 1.	Set $l = 1$. Set $r_m = 0; \forall m$.
Step 2.	Set (j, s, o) = GeneS2F1[l] of the chromosome in Figure 4.1-(c).
Step 3.	If $b_{s,j} > 0$, go to Step 4; Otherwise, go to Step 9.
Step 4.	Temporarily set $r_m = r_m + 1$ for each eligible machine m of operation o of job j (for each m such that $p_{o,j,m} = 1$). Using the procedure described in Figure 4.4, calculate the completion time of operation o of sublot s of job j corresponding to each of the eligible machines m.
Step 5.	Using the results from Step 4, select the machine that can complete the operation sooner. Say this machine is machine m^*.
Step 6.	Assign operation o of sublot s of job j to the $(r_{m^*})^{th}$ run of machine m^*.
Step 7.	Retain the values of $c_{o,s,j,m}$, $l_{m,o,s,j}$, $e_{s,j}$, and $d_{s,j}$ calculated Step 4 corresponding to machine $m = m^*$ as the final values of these variables.
Step 8.	Set $r_m = r_m - 1$ corresponding to all the other machines considered in Step 4 but not selected to processes operation o of job j in Step 5.
Step 9.	If l is equal to the total number of operations, Stop; Otherwise set $l = l+1$ and go to Step 2.

Figure 4.2: A decoding procedure for the solution representation given in Figure 4.1-(c) for the first stage of the Two-Stage Genetic Algorithm.

Stage-2

In Stage-2 of the genetic search, the second form of Segment-2 of the solution representation (Figure 4.1-(d)) is used. This form of the segment explicitly encodes the assignment and sequencing of the operations as it was discussed in Section 4.3.2. Unlike the decoding procedure previously discussed for Stage-1, the decoding in Stage-2 does not follow a greedy approach in selecting a machine for an operation assignment as the assignment and sequencing are directly inferred from the solution representation. The decoding procedure is only for the determination of several continuous variables along with the start and finish times of the operations of all the sublots having non-zero sizes. This decoding procedure is outlined in Figure 4.3. In this decoding procedure, the natation GeneS2F2[l] denotes the content of the gene $[j, s, o, m]$ at location l of the second form of Segment-2. The notations l and r_m have the same meaning as they were used in the previous discussion.

Step 1.	Set $l = 1$. Set $r_m = 0; \forall m$.
Step 2.	Set (j, s, o, m) = GeneS2F2[l] of the chromosome in Figure 4.1-(d).
Step 3.	If $b_{s,j} > 0$, go to Step 4; Otherwise, go to Step 7.
Step 4.	Set $r_m = r_m + 1$.
Step 5.	Assign operation o of sublot s of job j to the $(r_m)^{th}$ run of machine m.
Step 6.	Calculate the values of $c_{o,s,j,m}$, $l_{m,o,s,j}$, $e_{s,j}$, and $d_{s,j}$ using the procedure described in Figure 4.4.
Step 7.	If l is equal to the total number of operations, Stop; Otherwise set $l = l+1$ and go to Step 2

Figure 4.3: A decoding procedure for the solution representation given in Figure 4.1-(d) for the second stage of the Two-Stage Genetic Algorithm

Calculating Objective Function Terms

In the decoding procedures presented in the previous section, the values of $c_{o,s,j}$, $e_{s,j}$, $d_{s,j}$, and $l_{m,o,s,j}$ were determined. Once the values of these variables are known for each

If operation o of sublot s sublot of job j is to be processed on r_m^{th} run of machine m, the values of the variables $c_{o,s,j}$, $e_{s,j}$, $d_{s,j}$, $\hat{e}_j$, $\hat{d}_j$ and $l_{m,o,s,j}$ are calculated based on one of the following four cases:

- **Case 1:** $[o = 1;\ r_m = 1]$
  (a) Operation o of sublot s of job j is the first operation to be assigned on machine m (i.e., $r_m = 1$), and
  (b) $o = 1$.

  $c_{o,s,j} = D_m + S^*_{o,j,m} + b_{s,j} \times T_{o,j,m}$
  $l_{m,o,s,j} = S^*_{o,j,m} + b_{s,j} \times T_{o,j,m}$
  $e_{s,j} = c_{o,s,j,m} - b_{s,j} \times T_{o,j,m} - A_{o,j} \times S^*_{o,j,m}$

- **Case 2:** $[o > 1;\ r_m = 1]$

  (a) Operation o of sublot s of job j is the first operation to be assigned on machine m (i.e., $r_m = 1$),
  (b) $o > 1$, and
  (c) Operation $o - 1$ of sublot s of job j was assigned on machine m'.

  $c_{o,s,j} = \max\{D_m + (1 - A_{o,j}) \times S^*_{o,j,m}\ \ ,\ \ c_{o-1,s,j,m'} + L_{o,j}\} + b_{s,j} \times T_{o,j,m} + A_{o,j} \times S^*_{o,j,m}$
  $l_{m,o,s,j} = S^*_{o,j,m} + b_{s,j} \times T_{o,j,m}$
  If $o = O_j$, then $d_{s,j} = c_{o,s,j,m}$

- **Case 3:** $[o = 1;\ r_m > 1]$

  (a) Operation o of sublot s of job j is not the first operation to be assigned on machine m (i.e., $r_m > 1$),
  (b) Operation o' of sublot s' of job j' is the operation to be processed immediately before operation o of sublot s of job j on machine m (i.e., Operation o' of sublot s' of job j' was assigned to run $r_m - 1$ of machine m), and
  (c) $o = 1$.

  $c_{o,s,j} = c_{o',s',j',m} + S_{o,j,m,o',j'} + b_{s,j} \times T_{o,j,m}$
  $l_{m,o,s,j} = S_{o,j,m,o',j'} + b_{s,j} \times T_{o,j,m}$
  $e_{s,j} = c_{o,s,j,m} - b_{s,j} \times T_{o,j,m} - A_{o,j} \times S_{o,j,m,o',j'}$

- **Case 4:** $[o > 1;\ r_m > 1]$

  (a) Operation o of sublot s of job j is not the first operation to be assigned on machine m (i.e., $r_m > 1$),
  (b) Operation o' of sublot s' of job j' is assigned immediately before operation o of sublot s of job j on machine m (i.e., Operation o' of sublot s' of job j' was assigned to run $r_m - 1$ of machine m),
  (c) $o > 1$, and
  (d) Operation $o - 1$ of sublot s of job j is assigned on machine m'.

  $c_{o,s,j} = \max\{c_{o',s',j',m} + (1 - A_{o,j}) \times S_{o,j,m,o',j'}\ \ ,\ \ c_{o-1,s,j,m'} + L_{o,j}\} + b_{s,j} \times T_{o,j,m} + A_{o,j} \times S_{o,j,m,o',j'}$
  $l_{m,o,s,j} = S_{o,j,m,o',j'} + b_{s,j} \times T_{o,j,m}$
  If $o = O_j$, then $d_{s,j} = c_{o,s,j,m}$.

Figure 4.4: Calculation of the decision variables $c_{o,s,j}$, $e_{s,j}$, $d_{s,j}$, and $l_{m,o,s,j}$

sublot having size greater than zero ($b_{s,j} > 0$), the various terms of the objective function can easily be calculated as shown in Table 4.3.

4.3.4. Handling Multi-Objectives

In the literature, there are many techniques in handling multi-objective optimization using evolutionary algorithms. However, because of its simplicity and computational efficiency, we choose a weighted sum approach in which multiple objectives are aggregated into a single objective using a weight vector. In the best scenario, the weight vector is assigned by decision-makers having knowledge about the relative importance of the objective functions. However, because of large differences in magnitudes of the objective functions, scaling the objectives is always desirable to obtain solutions consistent with the decision-makers preferences. Hence, in the aggregated objective, the k^{th} objective function has to be multiplied by the weights W_k, reflecting the decision makers' preference, and Ψ_k for scaling as shown in Eq. (4.58). In this research, we adopt a simple objective function scaling mechanism in such a way that, in the initial population of the genetic algorithm, the magnitude of the maximum values of objective function terms Z_2 through Z_{10} will have the same values as the maximum value of Z_1. This scaling procedure can be mathematically described as shown in Eq. (4.59) where $Z_k^{Ini-max}$ represents the maximum value of objective Z_k in the initial population. The decision-maker is free to choose any positive value of W_k. The problem may be solved multiple times with different sets of W_k's, and the resulting solutions can be presented to decision-makers for final decision. Nevertheless, scheduling is a day-to-day activity where the decision-makers may already have a preferred set of W_k's from previous experience.

$$Z = \sum_{k=1}^{10} W_k \cdot \Psi_k \cdot Z_k \tag{4.58}$$

$$\Psi_k = \frac{Z_1^{Ini-max}}{Z_k^{Ini-max}} \tag{4.59}$$

4.3.5. Genetic Operators

A genetic algorithm works on a population of solutions. The initial population is generated randomly, and the algorithm works iteratively to evolve this population towards promising solutions following the principles of natural evolution. The mechanisms used to achieve this artificial evolutionary process are collectively called genetic operators. These operators are broadly classified into selection, crossover, and mutation. The operators used in the proposed genetic algorithm are discussed below.

Selection Operators

The role of selection operator in a genetic algorithm is to mimic the principle of the survival of the fittest in natural evolution. This operator creates a mating pool of individuals for reproduction. Selection can be applied in a variety of ways. In this research, we considered the three most commonly used approaches in literature, namely (1) proportional, (2) linear ranking, and (3) tournament selections. The following notations are used to describe these selection operators.

N Number of individuals (solutions) in a population.

$U(t)$ Population of solution at generation t;

$U(i,t)$ The i^{th} individual in the population at generation t;

$M(t)$ Mating pool created via selection operator from the population $U(t)$ (the size of the mating pool is the same as that of the population);

$M(i,t)$ The i^{th} individual in the mating pool at generation t;

$Z(i,t)$ The weighted objective function value corresponding to the i^{th} individual in the population at generation t;

$Z_{min}(t)$ The minimum observed weighted objective function value in the population at generation t;

$Z_{max}(t)$ The maximum observed weighted objective function value in the population at generation t;

$F(i,t)$ The fitness value of the i^{th} individual in the population at generation t;

$R(i,t)$ The rank of the i^{th} individual in the population at generation t for linear ranking selection;

$P(i,t)$ Probability of selection of i^{th} individual in the population at generation t for proportional or linear ranking selection method;

T Tournament size for tournament selection.

Algorithm 1: Monte Carlo Simulation of Roulette Wheel Spinning for proportional or ranked selection

```
   Input  : P(i,t) for i = 1,2,...,N
   Output: Winner
 1 Set Sum = 0
 2 Set ρ = rand()
          /* Assign ρ a random number between 0 and 1 using random
   number generator function rand() */
 3 for i = 1 to N do
 4 |   Sum = Sum + P(i,t)
 5 |   if ρ ≤ Sum then
 6 |   |   Winner = i
 7 |   |   Break
   |   |                  /* Break the "for loop" and go to line 10 */
 8 |   end
 9 end
10 Return Winner
```

Proportional Selection

Proportional selection is a procedure in which individuals from a given generation are selected (with replacement) to move to the mating pool with a probability proportional to their fitness F that needs to be maximized. In a problem where the objective function Z is to be minimized, a fitness function F has to be devised so that a solution with

Algorithm 2: Monte Carlo Simulation of Tornament selection

```
   Input  : Z(i,t) for i = 1,2,...,N
   Output: Winner
1  for j = 1 to T do
2  |  Competitor[j] = RandIntBetween(1, N)
   |                          /* Select T competitors randomly */
3  end
4  Winner = Competitor[1]
                  /* Assign Winner the index of the first competitor */
5  for j = 2 to T do
6  |  w = Winner
7  |  i = Competitor[j]
8  |  if Z(i,t) < Z(w,t) then
9  |  |  Winner = i
10 |  end
11 end
12 Return Winner
```

Algorithm 3: Creating the mating pool $M(t)$

```
   Input  : U(t)
   Output: M(t)
1  for i = 1 to N do
2  |  j = Selection()
   |    /* The function Selection() returns the index of the individual
   |  selected from U(t). This function is implemented either using
   |  Algorithm-1 if roulette wheel selection is used or Algorithm-2 if
   |  tournament selection is used */
3  |  M(i,t) = U(j,t)
4  end
5  Return Winner
```

smaller Z will have higher fitness than a solution with larger Z. In such situations, a commonly used fineness function is the reciprocal of Z as shown in Eq. (4.60). We also considered other two transformations shown in Eqs. (4.61) and (4.62). Once the fitness values for all the individuals in the population are calculated, each individual is assigned a probability of selection defined by the equation in Eq. (4.63). As it can be seen from this equation, $P(i,t)$ is proportional to the fitness $F(i,t)$ and the sum $\sum_i^N P(i,t)$ is

equal to 1. This probability distribution can be sampled using Monte-Carlo simulation of a Roulette wheel, where each solution is assigned a slot proportional to its probability of selection ($P(i,t)$). Algorithm 1 depicts the Monte-Carlo simulation of a Roulette wheel, and every time this algorithm is called, it returns an integer number ($winner$) representing the index of the selected individual. The procedure of constituting the mating pool $M(t)$ from a given population $U(t)$ is depicted in Algorithm 3.

$$F(i,t) = \frac{1}{Z(i,t)} \tag{4.60}$$

$$F(i,t) = Z_{max}(t) + Z_{min}(t) - Z(i,t) \tag{4.61}$$

$$F(i,t) = Z_{max,t} - Z(i,t) \tag{4.62}$$

$$P(i,t) = \frac{F(i,t)}{\sum_i^N F(i,t)} \tag{4.63}$$

Linear Ranking Selection

In a linear ranking selection, the individuals in the population are assigned ranks based on a sorted sequence of their objective function values. The individuals with the worst objective function are assigned a rank of 1, and the next worse individuals are assigned a rank of 2, and so on. In this process, the best individuals are assigned the highest possible rank. Once each individual is a assigned a rank R(i,t), a selection probability $P(i,t)$ can be calculated using Eq. (4.64). This probability function can be sampled using Monte-Carlo simulation of a Roulette wheel (Algorithm 1) to constitute the mating pool using Algorithm 3.

$$P(i,t) = \frac{R(i,t)}{\sum_i^N R(i,t)} \tag{4.64}$$

Tournament Selection

Tournament selection is the most commonly used selection operator in literature. In this selection procedure, every time a selection is performed, T individuals are randomly

selected (with replacement) from the population, and the one with the smallest Z is selected as a winner. The process is repeated for N number of times to form a mating pool of N individuals from a given generation of population. The integer parameter T is referred to as tournament size, and it is usually equal to a small fraction of N where the smallest possible value is 2. A large value of T results in higher selection pressure and premature convergence, whereas a small value of T may result in slow convergence. The Monte-Carlo simulation of tournament selection is given in Algorithm 2, which can be used along Algorithm 3 to constitute the mating pool.

Crossover Operators

Crossover operators are responsible for creating offspring from parent chromosomes via the exchange of genetic materials, mimicking sexual reproduction in living organisms. Once $M(t)$ is formed using the selection operator, each individual is paired randomly to create a total of $N/2$ pairs. Then, a crossover operator is applied on each pair resulting from the creation of offsprings. The crossover operators used in this work are listed below. SSC1, SSC2, JLOSC, SLOSC and MAC are direct adaptations from Defersha and Chen (2012a). However, in this work, JLOSC, SLOSC, and MAC are applicable only in the second stage of the search. JLGSC and SLGSC share similarities with JLOSC and SLOSC, but they are applicable only in the first stage of the genetic search.

(a) Sublot-Size Crossover-1 (SSC1)

(b) Sublot-Size Crossover-2 (SSC2)

(c) Job Level Gene Sequence Crossover (JLGSC)

(d) Sublot Level Gene Sequence Crossover (SLGSC)

(e) Job Level Operation Sequence Crossover (JLOSC)

(f) Sublot Level Operation Sequence Crossover (SLOSC)

(g) Machine Assignment Crossover (MAC)

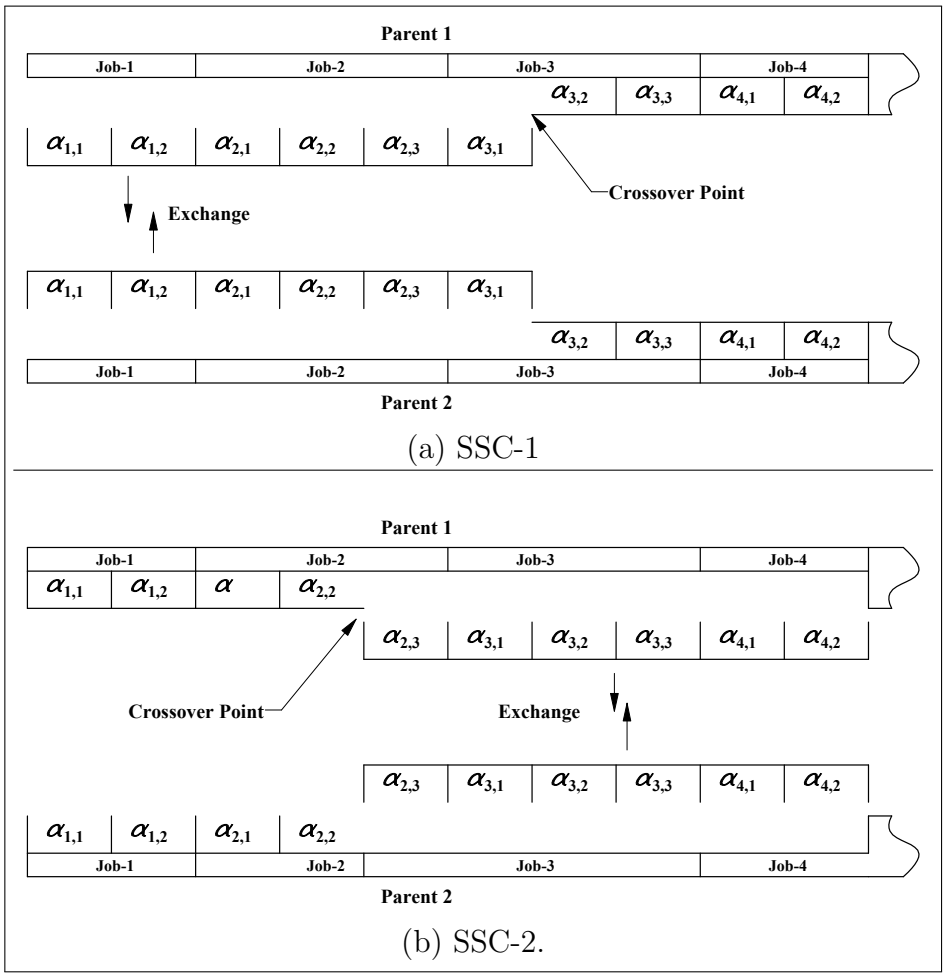


Figure 4.5: Illustration of the crossover operators SSC-1 and SSC-2

Figure 4.5 depicts the first two crossover operators (SSC1 and SSC2). When SSC1 (or SSC2) is applied, an arbitrary crossover point is selected on Segment-1, and the parts of this segment that lie to the left (or right) of the crossover point are exchanged. The step-by-step application of JLGSC is illustrated in Figure 4.6 where the creation of Child-1 is detailed. In Step-1, one gene is selected arbitrarily. This gene and all the other genes with the same job index, j, are copied from Parent-1 to Child-1. In Step-3, all the missing genes of Child-1 are copied from parent-2 in the order they appeared in this second parent. At the same time, Child-2 is also created by first copying genes from Parent-2 with the same job index as the arbitrarily selected gene. The missing genes of Child-2 will be obtained from Parent-1. SLGSC is applied in a similar manner as

JLGSC. However, the gene transfer in SLGSC is limited to the gens that have the same job and sublot index ($j,\ s$) as the arbitrarily selected gene in Step-1.

JLOSC is applied in four steps as shown in Figure 4.7. In Step-1, a crossover point (a gene) is selected arbitrarily. All the genes with the same job index as the arbitrarily selected gene are copied from Parent-1 to Child-1 in Step-2. In Step-3, the first three elements (j, s, o) of the missing genes of Child-1 are copied from Parent-2. In the last step, the machine assignments of the incomplete genes that were copied from Parent-2 are completed by the machine assignment obtained from Parent-1. The creation of Child-2 will be performed in a similar manner by starting from Parent-2. SLOSC is a reduced version of JLOSC where the first step is limited to the genes having the same job and sublot indices. Hence, if Figure 4.7 were for SLOSC, only the genes with job index $j = 3$ and sublot index $s = 2$ will be copied to Child-1 in Step-2. When either JLOSC or SLOSC is applied, Child-1 (Child-2) will have the same machine assigned as Parent-1 (Parent-2), but with a different operation sequence. Thus, JLOSC and SLOSC manipulate only the sequence of operations without altering machine assignments in stage-2 of the generic search.

The machine assignment crossover (MAC), shown in Figure 4.8, is responsible for exchanging machine assignment information between parent chromosomes during stage-2 of the genetic search. As it can be seen in the figure, this operator is applied in three steps to create offspring. In Step-1, several genes are arbitrarily selected (each one with 50% chance). In Step-2, the contents of Parent-1 are copied to Child-1 without the machine assignment information of the arbitrarily chosen genes. In the last step, the missing machine assignment information is copied from Parent-2. Child-2 is created in a similar manner by starting Step-1 from Parent-2 for the same locations of the arbitrarily selected genes in creating Child-1 (i.e., locations 6, 10, 14, 18, 20, 23, 25, 27, and 31). Here it is essential to mention that, though there are many crossover operators, whenever crossover is to happen between a pair of parent chromosomes, only one crossover operator will be arbitrarily selected and applied with a probability (p_{cros}) to create two offsprings. If the selected crossover operator is not applied (by chance), the parent chromosomes

will move to the next generation (with or without mutation operators applied, again by chance).

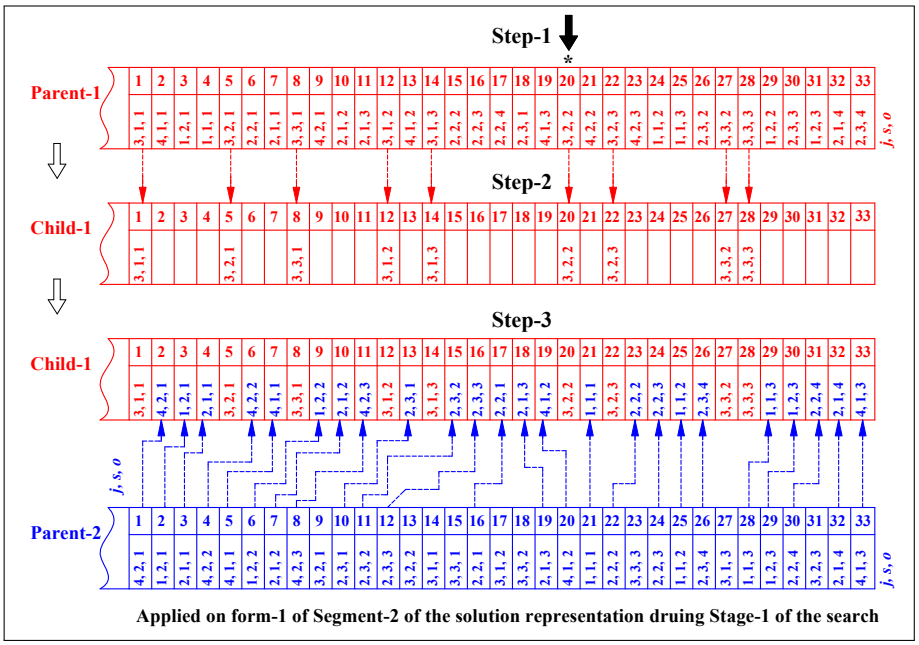


Figure 4.6: Illustration of JLGSC crossover operator

Mutation Operators

Mutation operators are applied with a small probability on a newly generated offspring to alter its genetic material. This category of operators used in this work are listed below. They are adapted from Defersha and Chen (2012a) while the need to divide the genetic search into two stages is taken into consideration. The first two mutation operators, SGVM and SGSM, are applicable both in Stage-1 and -2 of the GA search. Whereas OGSM is applicable only in Stage-1, and OSSM, ROAM, and IOAM are applicable only in Stage-2. Each one of the six mutation operators listed below is applied with a probability p_{mut} on a newly generated offspring as long as it is eligible for the stage of the search of the GA.

(a) Sublot Gene Value Mutation (SGVM)

(b) Sublot Gene Swap Mutation (SGSM)

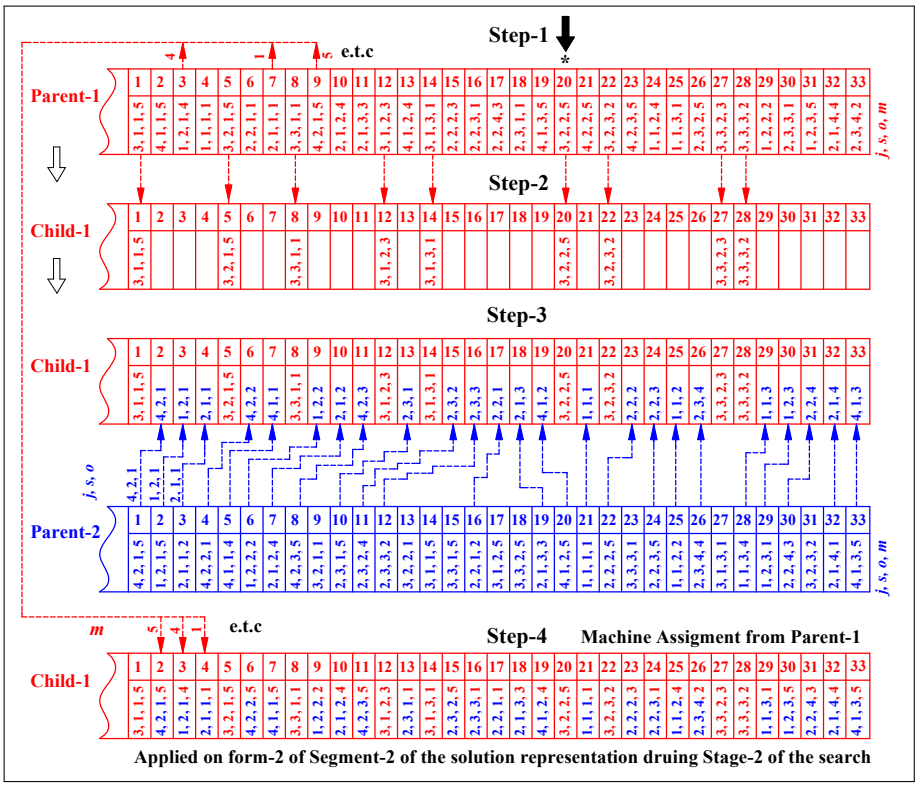


Figure 4.7: Illustration of JLOSC crossover operator

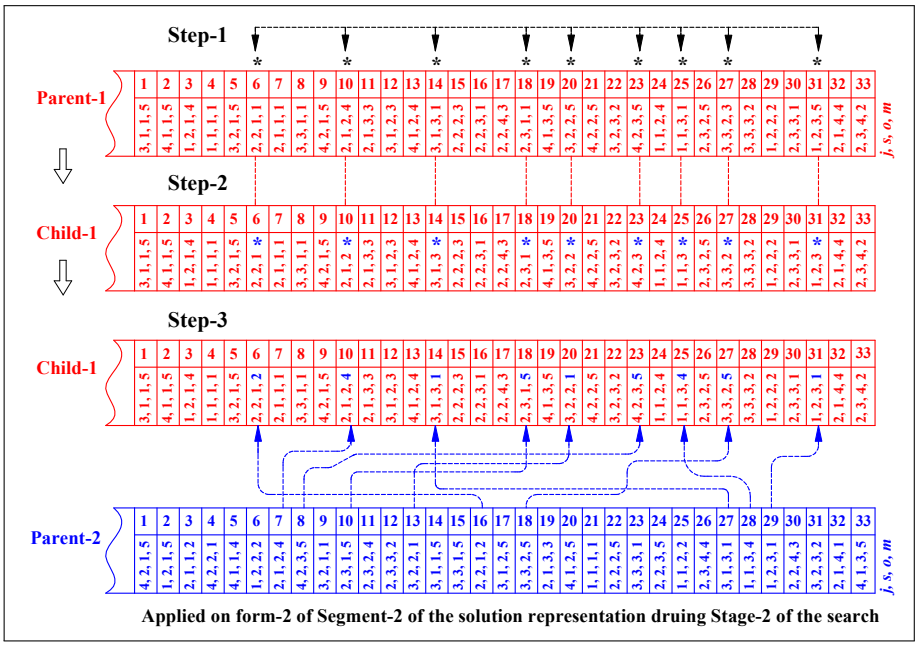


Figure 4.8: Illustration of MAC crossover operator

(c) Operation Gene Shift Mutation (OGSM)

(d) Operations Sequence Shift Mutation (OSSM)

(e) Random Operation Assignment Mutation (ROAM)

(f) Intelligent Operations Assignment Mutation (IOAM)

SGVM is used to alter the value of a gene $\alpha_{j,s}$ in Segment-1 of a newly born offspring chromosome. When this operator is applied, a single gene is arbitrarily selected, and its value is either increased or decreased (50% by chance) with a small quantity according to Eq. (4.65) or Eq. (4.66), respectively. In these two equations, rand() is a function that returns a random number in the interval [0, 1], and δ is the maximum increment or decrement quantity which can be regarded as the GA's parameter that needs to be set. In this research, we found that the value δ between 0.1 and 0.2 is preferable. The second mutation operator, SGSM, arbitrarily selects a job j in Segment-1 and swap the values of two arbitrarily selected genes $\alpha_{j,s}$ and $\alpha_{j,s'}$ corresponding to sublots s and s' ($s \neq s'$).

$$\alpha_{j,s} = \min\{1, \alpha_{j,s} + \text{rand}() \times \delta\} \tag{4.65}$$

$$\alpha_{j,s} = \max\{1, \alpha_{j,s} - \text{rand}() \times \delta\} \tag{4.66}$$

Operation gene shift mutation (OGSM) is applied on form-1 of Segment-2 of the solution representation (Figure 4.1-c) during Stage-1 of the GA search. When this operator is applied, it first arbitrarily selects a gene (j, s, o), and then relocates it to an arbitrary location after and before the locations of the genes $(j, s, o-1)$ and $(j, s, o+1)$, respectively. If $o = 0$, the selected gene (j, s, o) can be moved only forward to an arbitral location before the location of gene $(j, s, o + 1)$. Whereas, if $o = O_j$, the gene can be moved only backward to a location after the location of the gene $(j, s, o - 1)$. OGSM impacts both machine assignment and operation sequencing as it alters the sequence of the genes, which is used to determine operations assignment and sequencing in Stage-1 of the GA search by the greedy procedure described in Section 4.3.3. OSSM

is applied on form-2 of Segment-2 ((Figure 4.1-d)) to shift a location of an arbitrarily selected gene (j, s, o, m) during Stage-2 of the search. It is applied in a similar manner as OGSM was applied during Stage-1. However, in Stage-2, since the genes directly encode machine assignment, OSSM impacts only operation sequencing but not the machine assignment of the operations. ROAM is a mutation operator responsible for altering machine assignment in Stage-2 of the GA search. The operator arbitrarily selects a gene (j, s, o, m), and changes the value of m to a different eligible machine m' for operation o of job j (i.e., $P_{o,j,m'} = 1$). IOAM intelligently changes a machine assignment in an attempt to lower the workload on a heavily loaded machine. This operator first identifies the machine with the largest workload because of the solution under consideration for a mutation (let that machine be donated as m^*). Then it selects one of the operations assigned to m^* and relocates it to an eligible machine with the least load as long as the load transfer will not make the least loaded machine more loaded than m^* after the load transfer.

4.4. Numerical Studies

4.4.1. Model Analysis

Illustration of Objective Function Terms

This section attempts to illustrate the various objective function terms considered in this work. For this purpose, the prototype problem presented in Section 4.3.1 was solved using the proposed algorithm, where makespan minimization was the only objective function, and other objective function terms were merely evaluated. The details of a typical solution are given in Tables 4.4 to 4.8. Table 4.4 provides the sizes of the created sublots and operation-to-machine assignments and the run orders along with the Beginning and End times of Lag time, Setup, and Processing. The maximum Processing End time, 2603.8, is the makespan of the schedule.

The values of the objective function terms related to sublot flowtime can be extracted from Table 4.4. The entry time to the shop floor of a sublot ($e_{s,j}$) is the setup begin (SB) time of the first operation if the setup is attached or it is equal to the setup end (SE) time if the setup is detached. Here, it is important to note that if the setup of the first operation is detached, the setup can begin and be completed before raw material is dispatched to the shop floor. The sublot departure time ($d_{s,j}$) is the process end (PE) time of the last operation. From the first row of column ten of Table 4.4, $e_{1,1} = 100$, because the job is dispatched after its setup is completed as the first operation of this sublot has a detached setup time. The departure time $d_{1,1} = 2587.5$. Hence, the flowtime $f_{1,1} = 2587.5 - 100 = 2487.5$. The flowtimes for the other sublots can be determined similarly and are summarized in Table 4.5. At the bottom of this table, the performance measure f_{max} and f_{total} are indicated as 2487.5 and 16560.6, respectively.

Job flowtime and sublot-finish-separation objective function terms can be evaluated from Table 4.5. The entry time $\hat{e}_1$ and the departure time $\hat{d}_1$ of the first job are the same as $e_{1,1}$ and $d_{1,1}$, respectively, of the first sublot since this job has only one sublot in the final solution. Hence, its flowtime $\hat{f}_1 = f_{1,1} = 2487.5$. The second job has three sublots in the final solution. Thus, its entry time $\hat{e}_2$ is the minimum of $\{e_{1,2},\ e_{2,2},\ e_{3,2}\} = e_{3,2} = 120.0$, and its departure time $\hat{d}_2$ is the maximum of $\{d_{1,2},\ d_{2,2},\ d_{3,2}\} = d_{3,2} = 2599.8$. Therefore, the flowtime of job-2 is evaluated as $\hat{f}_2 = 2599.8 - 120.0 = 2479.5$. For job-3, $\hat{e}_3$ is the minimum of $\{e_{1,3},\ e_{2,3},\ e_{3,3}\} = e_{1,3} = 920.0$, and $\hat{d}_3$ is the maximum of $\{d_{1,3},\ d_{2,3},\ d_{3,3}\} = d_{2,3} = 2603.8$. Therefore, $\hat{f}_3 = 2603.8 - 920.0 = 1683.8$. Similarly, for the forth job, $\hat{e}_4 = 220.0$, $\hat{d}_4 = 2583.6$, and $\hat{f}_4 = 2363.6$. The sublot finish separation time for job-1 is zero since this job has only one sublot. Whereas, the sublot finish-time separation of job-2 is the difference between (1) the maximum of $\{d_{1,2},\ d_{2,2},\ d_{3,2}\} = d_{3,2} = \hat{d}_2$, and (2) minimum of $\{d_{1,2},\ d_{2,2},\ d_{3,2}\} = d_{1,2} = \hat{g}_2$, which is evaluated as $\hat{h}_2 = \hat{d}_2 - \hat{g}_2 = 2599.8 - 2257.4 = 342.4$. The sublot finish-time separations for the other jobs can be evaluated similarly. The result is summarized in Table 4.6. In the last row of this table, the objective function components $\hat{f}_{mas}$, $\hat{f}_{totoal}$, $\hat{h}_{mas}$ and $\hat{h}_{totoal}$ are indicated as 2487.5, 9014.7, 1006.1 and 1787,

respectively.

Table 4.7 provides the schedule for the prototype problem with respect to the machines. From this table, the workload because of each operation assignment can be evaluated by subtracting SB from PE. For instance, from the first row of this table, the load because of operation-1 of sublot-1 of job-3 is $l_{m,o,s,j} = l_{1,1,1,3} = 1342.0 - 840.0 = 502$. Similarly, the workload because of the other four operations on machine-1 can be evaluated as 441.8, 381.4, 291.3, and 147.3, bringing the total workload on this machine to 1763.8 + 840 = 2603.8, where 840 is the release date of the machine. The utilization (workload/makespan) of this machine is 100% as its workload is the same as the makespan. The workloads and the utilization of the other machines are also evaluated in a similar way as summarized in Table 4.8. The objective function components $\hat{l}_{max}$, $\hat{l}_{total}$ and $\hat{l}_{max} - \hat{l}_{min}$ are given in the last row of this table as 2603.8, 12488.4 and 427.7, respectively. The values for all of the objective function components are summarized in Table 4.9.

Table 4.2: Sequence Dependent Setup Time Data for Problem-1

			Setup Time $S^*_{o,j,m}$ and $S_{o,j,m,o',j'}$	
j	o	m	$(S^*_{o,j,m})$	$\cdots(j',\ o',\ S_{o,j,m,o',j'})\ \cdots$
1	1	1	(120)	(1,1,20)(1,2,100)(1,3,120)(2,1,210)(2,3,210)(2,4,240)(3,1,240)(3,2,210)(3,3,240)(4,2,210)
		4	(140)	(1,1,15)(1,2,80)(1,3,120)(2,2,180)(2,4,240)(4,1,210)(4,2,210)(4,3,240)
		5	(100)	(1,1,20)(1,3,80)(2,1,210)(2,2,180)(2,3,240)(3,1,180)(3,2,240)(4,1,180)(4,2,180)(4,3,180)
	2	1	(140)	(1,1,100)(1,2,20)(1,3,80)(2,1,240)(2,3,210)(2,4,180)(3,1,240)(3,2,210)(3,3,240)(4,2,210)
		2	(100)	(1,2,15)(1,3,100)(2,1,180)(2,2,210)(2,3,180)(2,4,180)(3,3,180)(4,3,210)
		4	(140)	(1,1,80)(1,2,10)(1,3,80)(2,2,240)(2,4,180)(4,1,240)(4,2,210)(4,3,240)
	3	1	(80)	(1,1,80)(1,2,120)(1,3,10)(2,1,180)(2,3,240)(2,4,180)(3,1,240)(3,2,180)(3,3,210)(4,2,180)
		2	(160)	(1,2,120)(1,3,20)(2,1,180)(2,2,210)(2,3,180)(2,4,180)(3,3,210)(4,3,210)
		4	(80)	(1,1,100)(1,2,100)(1,3,15)(2,2,240)(2,4,240)(4,1,240)(4,2,240)(4,3,210)
		5	(120)	(1,1,100)(1,3,20)(2,1,180)(2,2,180)(2,3,210)(3,1,180)(3,2,210)(4,1,180)(4,2,210)(4,3,240)
2	1	1	(80)	(1,1,240)(1,2,240)(1,3,240)(2,1,20)(2,3,120)(2,4,120)(3,1,210)(3,2,210)(3,3,180)(4,2,240)
		2	(120)	(1,2,180)(1,3,180)(2,1,10)(2,2,100)(2,3,120)(2,4,120)(3,3,180)(4,3,180)
		5	(160)	(1,1,240)(1,3,180)(2,1,15)(2,2,100)(2,3,100)(3,1,180)(3,2,210)(4,1,180)(4,2,180)(4,3,240)
	2	2	(120)	(1,2,180)(1,3,210)(2,1,80)(2,2,15)(2,3,120)(2,4,100)(3,3,240)(4,3,210)
		3	(100)	(2,2,20)(2,3,80)(2,4,120)(3,2,240)(4,2,210)
		4	(120)	(1,1,210)(1,2,180)(1,3,240)(2,2,10)(2,4,80)(4,1,180)(4,2,240)(4,3,240)
		5	(120)	(1,1,180)(1,3,240)(2,1,120)(2,2,20)(2,3,80)(3,1,210)(3,2,240)(4,1,240)(4,2,210)(4,3,240)
	3	1	(80)	(1,1,210)(1,2,240)(1,3,240)(2,1,80)(2,3,10)(2,4,80)(3,1,240)(3,2,240)(3,3,240)(4,2,210)
		2	(100)	(1,2,210)(1,3,180)(2,1,120)(2,2,80)(2,3,15)(2,4,80)(3,3,210)(4,3,180)
		3	(140)	(2,2,80)(2,3,10)(2,4,100)(3,2,240)(4,2,180)
		5	(160)	(1,1,240)(1,3,210)(2,1,80)(2,2,120)(2,3,10)(3,1,240)(3,2,180)(4,1,180)(4,2,240)(4,3,210)
	4	1	(160)	(1,1,180)(1,2,210)(1,3,210)(2,1,80)(2,3,100)(2,4,10)(3,1,240)(3,2,180)(3,3,180)(4,2,210)
		2	(140)	(1,2,180)(1,3,180)(2,1,120)(2,2,80)(2,3,120)(2,4,10)(3,3,180)(4,3,210)
		3	(160)	(2,2,100)(2,3,100)(2,4,10)(3,2,240)(4,2,210)
		4	(120)	(1,1,240)(1,2,240)(1,3,180)(2,2,120)(2,4,10)(4,1,210)(4,2,180)(4,3,210)
3	1	1	(80)	(1,1,240)(1,2,240)(1,3,240)(2,1,180)(2,3,180)(2,4,210)(3,1,15)(3,2,120)(3,3,100)(4,2,180)
		5	(80)	(1,1,210)(1,3,240)(2,1,240)(2,2,180)(2,3,240)(3,1,15)(3,2,100)(4,1,210)(4,2,180)(4,3,210)
	2	1	(120)	(1,1,240)(1,2,240)(1,3,180)(2,1,240)(2,3,240)(2,4,240)(3,1,80)(3,2,20)(3,3,100)(4,2,210)
		3	(140)	(2,2,240)(2,3,240)(2,4,240)(3,2,15)(4,2,180)
		5	(160)	(1,1,180)(1,3,210)(2,1,240)(2,2,180)(2,3,180)(3,1,100)(3,2,10)(4,1,240)(4,2,240)(4,3,240)
	3	1	(120)	(1,1,180)(1,2,240)(1,3,240)(2,1,210)(2,3,180)(2,4,180)(3,1,120)(3,2,100)(3,3,10)(4,2,210)
		2	(160)	(1,2,240)(1,3,210)(2,1,210)(2,2,180)(2,3,240)(2,4,180)(3,3,15)(4,3,210)
4	1	4	(100)	(1,1,210)(1,2,240)(1,3,210)(2,2,240)(2,4,180)(4,1,10)(4,2,100)(4,3,100)
		5	(100)	(1,1,180)(1,3,210)(2,1,180)(2,2,180)(2,3,210)(3,1,210)(3,2,210)(4,1,10)(4,2,120)(4,3,100)
	2	1	(100)	(1,1,240)(1,2,240)(1,3,210)(2,1,180)(2,3,180)(2,4,240)(3,1,180)(3,2,240)(3,3,240)(4,2,20)
		3	(80)	(2,2,240)(2,3,240)(2,4,240)(3,2,210)(4,2,20)
		4	(140)	(1,1,180)(1,2,210)(1,3,180)(2,2,240)(2,4,240)(4,1,100)(4,2,10)(4,3,100)
		5	(80)	(1,1,180)(1,3,180)(2,1,240)(2,2,210)(2,3,210)(3,1,240)(3,2,180)(4,1,80)(4,2,15)(4,3,80)
	3	2	(100)	(1,2,210)(1,3,210)(2,1,210)(2,2,210)(2,3,180)(2,4,210)(3,3,210)(4,3,20)
		4	(140)	(1,1,240)(1,2,210)(1,3,240)(2,2,180)(2,4,180)(4,1,120)(4,2,100)(4,3,15)
		5	(140)	(1,1,180)(1,3,240)(2,1,210)(2,2,180)(2,3,210)(3,1,240)(3,2,240)(4,1,100)(4,2,100)(4,3,10)

Table 4.3: Calculating the objective function terms once the values of $c_{o,s,j}$, $e_{s,j}$, $d_{s,j}$, and $l_{m,o,s,j}$ are evaluated corresponding to each sublot with non-zero size.

Intermediate Calculation	Obj. Function Term
$c_{max} = \max_{\forall(o,s,j)} c_{o,s,j}$	$Z_1 = c_{max}$
$f_{s,j} = d_{s,j} - e_{s,j}$ $f_{max} = \max_{\forall(s,j)\|b_{s,j}>0}\{f_{s,j}\}$	$Z_2 = f_{max}$
$f_{total} = \sum_{\forall(s,j)\|b_{s,j}>0}\{f_{s,j}\}$	$Z_3 = f_{total}$
$\hat{e}_j = \min_{\forall s\|b_{s,j}>0}\{e_{s,j}\}$ $\hat{d}_j = \max_{\forall s\|b_{s,j}>0}\{d_{s,j}\}$ $\hat{f}_j = \hat{d}_j - \hat{e}_j$ $\hat{f}_{max} = \max_{\forall j}\{\hat{f}_j\}$	$Z_4 = \hat{f}_{max}$
$\hat{f}_{total} = \sum_{\forall j}\{\hat{f}_j\}$	$Z_5 = \hat{f}_{total}$
$\hat{g}_j = \min_{\forall s\|b_{s,j}>0}\{d_{s,j}\}$ $\hat{h}_j = \hat{d}_j - \hat{g}_j$ $\hat{h}_{max} = \max_{\forall j}\{\hat{h}_j\}$	$Z_6 = \hat{h}_{max}$
$\hat{h}_{total} = \sum_{\forall j}\{\hat{h}_j\}$	$Z_7 = \hat{h}_{total}$
$\hat{l}_m = D_m + \sum_{\forall(m,o,s,j)\|b_{s,j}>0}\{l_{m,o,s,j}\}$ $\hat{l}_{max} = \max_{\forall m}\{l_m\}$	$Z_8 = \hat{l}_{max}$
$\hat{l}_{total} = \sum_{\forall m}\{l_m\}$	$Z_9 = \hat{l}_{total}$
$\hat{l}_{min} = \min_{\forall m}\{l_m\}$ $\hat{l}_{diff} = l_{max} - l_{min}$	$Z_{10} = \hat{l}_{diff}$

Table 4.4: Operation scheduling for Problem-1

j	s	$b_{s,j}$	o	m	r	LB	LE	SB	SE/PB	PE
1	1	100.0	1	5	1	0.0	0.0	0.0	100.0	750.0
			2	4	6	750.0	870.0	1577.5	1817.5	2092.5
			3	4	7	2092.5	2212.5	2112.5	2212.5	2587.5
2	1	90.8	1	2	2	0.0	0.0	303.0	313.0	494.6
			2	3	2	494.6	494.6	792.0	812.0	1197.8
			3	3	3	1197.8	1237.8	1237.8	1317.8	1907.8
			4	3	4	1907.8	1947.8	1907.8	2007.8	2257.4
	2	67.7	1	2	3	0.0	0.0	494.6	504.6	640.0
			2	2	4	640.0	640.0	640.0	720.0	1058.5
			3	2	5	1058.5	1098.5	1098.5	1178.5	1652.5
			4	2	7	1652.5	1692.5	2308.1	2428.1	2597.4
	3	91.5	1	2	1	0.0	0.0	0.0	120.0	303.0
			2	3	1	303.0	303.0	303.0	403.0	792.0
			3	2	6	792.0	832.0	1652.5	1667.5	2308.1
			4	3	5	2308.1	2348.1	2338.1	2348.1	2599.8
3	1	80.4	1	1	1	0.0	0.0	840.0	920.0	1342.0
			2	1	2	1342.0	1342.0	1342.0	1422.0	1783.8
			3	1	3	1783.8	1783.8	1783.8	1883.8	2165.2
	2	39.2	1	5	2	0.0	0.0	750.0	960.0	1185.5
			2	5	5	1185.5	1185.5	2104.4	2114.4	2281.1
			3	1	5	2281.1	2281.1	2456.5	2466.5	2603.8
	3	80.4	1	5	3	0.0	0.0	1185.5	1200.5	1662.8
			2	5	4	1662.8	1662.8	1662.8	1762.8	2104.4
			3	1	4	2104.4	2104.4	2165.2	2175.2	2456.5
4	1	50.0	1	4	2	0.0	0.0	520.0	530.0	830.0
			2	4	3	830.0	830.0	830.0	930.0	1167.5
			3	5	6	1167.5	1167.5	2281.1	2521.1	2583.6
	2	50.0	1	4	1	0.0	0.0	120.0	220.0	520.0
			2	4	4	520.0	520.0	1167.5	1177.5	1415.0
			3	4	5	1415.0	1415.0	1415.0	1515.0	1577.5

LB = Lag time Begins; LE = Lag time Ends; SB = Setup Begins; SE = Setup Ends; PB = Processing Begins; PE = Processing Ends.

Table 4.5: Sublot flowtime related performance measure (f_{max} and f_{total})

j	s	$e_{s,j}$	$d_{s,j}$	$f_{s,j}$
1	1	100.0	2587.5	2487.5
2	1	313.0	2257.4	1944.4
	2	504.6	2597.4	2092.8
	3	120.0	2599.8	2479.8
3	1	920.0	2165.2	1245.2
	2	960.0	2603.8	1643.8
	3	1200.5	2456.5	1256.0
4	1	530.0	2583.6	2053.6
	2	220.0	1577.5	1357.5
			Maximum	2487.5
			Total	16560.6

Table 4.6: Job flowtime and sublot finish-time separation performance measures ($\hat{f}_{mas}$, $\hat{f}_{totoal}$, $\hat{h}_{mas}$ and $\hat{h}_{totoal}$)

j	$\hat{e}_j$	$\hat{d}_j$	$\hat{f}_j$	$\hat{h}_j$
1	100.0	2587.5	2487.5	0.0
2	120.0	2599.8	2479.8	342.4
3	920.0	2603.8	1683.8	438.6
4	220.0	2583.6	2363.6	1006.1
		Maximum	2487.5	1006.1
		Total	9014.7	1787

Optimizing a Single Objective

When we optimize only one objective function term, unaccounted objective function terms can be adversely impacted. This phenomenon asserts the importance of multi-objective optimization. To illustrate this reality, we solve Problem-1 by considering one objective function term at a time. The result is depicted in Figure 4.9. Figure 4.9-(a) provides the values of the makespan (Z_1) when Z_1 through Z_{10} are optimized one at a time as a single objective function. As it should be the case, the smallest makespan

Table 4.7: Operation scheduling for Problem-1 with reference to the machines

m	r	j	s	o	SB	SE/PB	PE
1	1	3	1	1	840.0	920.0	1342.0
	2	3	1	2	1342.0	1422.0	1783.8
	3	3	1	3	1783.8	1883.8	2165.2
	4	3	3	3	2165.2	2175.2	2456.5
	5	3	2	3	2456.5	2466.5	2603.8
2	1	2	3	1	0.0	120.0	303.0
	2	2	1	1	303.0	313.0	494.6
	3	2	2	1	494.6	504.6	640.0
	4	2	2	2	640.0	720.0	1058.5
	5	2	2	3	1098.5	1178.5	1652.5
	6	2	3	3	1652.5	1667.5	2308.1
	7	2	2	4	2308.1	2428.1	2597.4
3	1	2	3	2	303.0	403.0	792.0
	2	2	1	2	792.0	812.0	1197.8
	3	2	1	3	1237.8	1317.8	1907.8
	4	2	1	4	1907.8	2007.8	2257.4
	5	2	3	4	2338.1	2348.1	2599.8
4	1	4	2	1	120.0	220.0	520.0
	2	4	1	1	520.0	530.0	830.0
	3	4	1	2	830.0	930.0	1167.5
	4	4	2	2	1167.5	1177.5	1415.0
	5	4	2	3	1415.0	1515.0	1577.5
	6	1	1	2	1577.5	1817.5	2092.5
	7	1	1	3	2112.5	2212.5	2587.5
5	1	1	1	1	0.0	100.0	750.0
	2	3	2	1	750.0	960.0	1185.5
	3	3	3	1	1185.5	1200.5	1662.8
	4	3	3	2	1662.8	1762.8	2104.4
	5	3	2	2	2104.4	2114.4	2281.1
	6	4	1	3	2281.1	2521.1	2583.6

SB = Setup Begins; SE = Setup Ends; PB = Processing Begins; PE = Processing Ends.

(about 2608) is achieved when Z_1 is considered as the only term in the objective function. However, when another term alone is optimized, makespan greatly deteriorates. For instance, when only Z_2 alone is optimized, the value of the makespan increases to 5067

Table 4.8: Machine workload related performance ($\hat{l}_{max}$ and $\hat{l}_{max} - \hat{l}_{min}$)

m	workload	utilization
1	2603.8	100.0
2	2557.4	98.2
3	2176.1	83.6
4	2567.5	98.6
5	2583.6	99.2

Maximum workload = 2603.8; Total workload = 12488.4;
Maximum workload difference = 427.7

Table 4.9: Values of the objective function components

Objective Term	Notation	Value
Makespan	Z_1	2603.8
Maximum Sublot Flowtime	Z_2	2487.5
Total Sublot Flowtime	Z_3	16560.6
Maximum Job flowtime	Z_4	2487.5
Total job flowtime	Z_5	9014.7
Maximum Sublot Finish-time Separation	Z_6	1006.1
Total Sublot finish-time Separation	Z_7	1787.1
Maximum Machine Load	Z_8	2603.8
Total Machine Load	Z_9	2603.8
Maximum Machine Load Difference	Z_{10}	427.7

(94% increase). Figures 4.9-(b) is a plot of the maximum sublot flowtime (Z_2) when Z_1 through Z_{10} are optimized one at a time. Its minimum value is 1468 when Z_2 alone is optimized. This value increases to 2431 when Z_1 alone is optimized. A single objective optimization of Z_6 through Z_{10} has significantly negative impacts on Z_2. A similar phenomenon is observed on all the other objective function terms, as can be seen from Figures 4.9-(c) to (j).

Here, it is important to note that the magnitude of the severity of a single objective optimization on the objective function terms that are not incorporated increases as the problem size increases. To exemplify this fact, we conducted a similar analysis on a relatively large problem (Problem-4), and the result is compiled in Table 4.10. For

instance, when the total machine load (Z_9) was the only objective function, its value is 573,164 minutes (see at row-Z_9 column-Z_9), which increases by 84,386 minutes when makespan (Z_1) is the only objective function optimized (see at row-Z_1 column-Z_9). This increment was only 721 minutes in a similar analysis in Problem-1. The total sublot flowtime in Problem-4 was equal to 2,115,095 when it was the only objective function (see row-Z_3 column-Z_3). This value increases to 4,004,945 when minimizing maximum machine load is the only objective (see row-Z_8 column Z_3). The last two rows of Table 4.10 show the best and the worst observed values of each objective function term. From these rows, we can see a considerably large gap between the best and the worst values of an objective function term. The best value of an objective function term is obtained when it is the only term optimized, and the worst is found when it is unaccounted for optimization. These significantly large deteriorations of unaccounted objective function terms in a large size problem greatly emphasize the need for multi-objective optimization in real industrial scheduling problems that are usually large in size.

Jointly Optimizing $Z_1, \ldots, Z_{10}$

In the previous section, the best and the worst values of the various objective function terms were determined when only one term was optimized at a time. In this section, we attempted to illustrate the ability of the proposed algorithm to jointly optimize all the terms and achieve values close to their best-known ones. In doing so, we first provide a plot of the values of the various objective function terms of Problem-4 in Figure 4.10-(a) when makespan is the only objective function optimized. In this figure, the values of the objective function terms are plotted on a scale between 0 and 1, corresponding to the best and worst values, respectively. Makespan achieves its minimum value since it is the only objective optimized. However, from this plot, one can see that several objective function terms, namely Z_6, Z_7, and Z_9, are not close enough to their respective best values obtained when each one of them was the only objective optimized. Next, we solve the same problem to jointly optimize all the objective function terms with equal weights set at one. The resulting values of the objective function terms are plotted in

Figure 4.10-(b). In this plot, except for Z_9, all the values of the objective function terms are close to their best values and far from their worst values. Finally, Problem-4 was optimized with increased weight for Z_9, and the values of terms are plotted in Figure 4.10-(c). From this final plot, one can see that all the terms of the objective function are much closer to their best values than to their worst values. This result demonstrates the ability of the proposed algorithm to jointly optimize all the objective function terms considered in the proposed model.

Further Empirical Study of Objective Functions

In this section, we conducted additional empirical investigations to illustrate the interaction of the objective function terms and their relevance in providing good quality solutions. A total of eleven cases were investigated. The cases differ by the values of the weights of the objective function terms. The settings for the weights for these eleven cases are given in Table 4.11. In each case, the genetic algorithm was executed ten times, and the average values of the objective function terms were collected. Table 4.12 provides these values.

Case-1 and Case-2 were considered to investigate flowtime performance measures. Case-1 attempts to minimize the maximum and total ***sublot flowtime*** (Z_2 and Z_3), whereas Case-2 attempts to minimize the maximum and total ***job flowtime*** (Z_4 and Z_5). The objective function terms Z_1, Z_8, Z_9 and Z_{10} are also optimized. In shifting from Case-1 to Case-2, the total job flowtime (Z_5) changes from 829235 to 823195 (less than 1% improvement). However, the total sublot flowtime (Z_3) changes from 1017100 to 1143060 (12% deterioration). Moreover, Case-2 increased the total workload (Z_9) by 10396 minutes (a change from 561306 to 571702 minutes). Hence, optimizing sublot flowtime is more desirable than optimizing job flowtime. However, as it can be seen from the values of Z_2, Z_3, Z_4, and Z_5 in Case-0, optimizing both sublot and job flowtime simultaneously can result in a favorable solution with respect to the overall flowtime performance.

In both Case-1 and Case-2, the maximum and total sublot finish-times separations

Table 4.10: Values of the objective function terms in Problem-4 when only one objective function term is optimized

Term optimized as a single objective function		Objective function term evaluated									
		Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	Z_8	Z_9	Z_{10}
Makespan	Z_1	26604	26511	2500084	26594	1002941	8580	96070	26593	660550	715
Maximum Sublot Flowtime	Z_2	28756	24014	2495091	28367	1016441	9426	110413	28175	665522	3063
Total Sublot Flowtime	Z_3	28822	27596	2115095	27977	951313	13437	173636	27964	667816	2747
Maximum Job flowtime	Z_4	28619	25156	2520010	25161	990089	8029	71921	28155	663042	2989
Total Job Flowtime	Z_5	27316	26770	2312532	26846	868391	6431	41727	27071	652393	2169
Max. Sublot Finish-time Separation	Z_6	37849	37323	3417121	37608	1337687	709	19222	30936	695833	6275
Total Sublot Finish-time Separation	Z_7	30966	30475	2857497	30561	1110654	1788	10813	29629	690055	4700
Maximum Machine Load	Z_8	44197	43469	4004945	43658	1586503	14786	158042	25793	643747	227
Total Machine Load	Z_9	43242	41933	3926277	42356	1494746	15611	76124	28123	573164	10040
Maximum Machine Load Difference	Z_{10}	36943	36238	3314121	36671	1348060	15639	161671	27729	693034	15
Minimum (Best Value)		26604	24014	2115095	25161	868391	709	10813	25793	573164	15
Maximum (Worst Value)		44197	43469	4004945	43658	1586503	15639	173636	30936	695833	10040

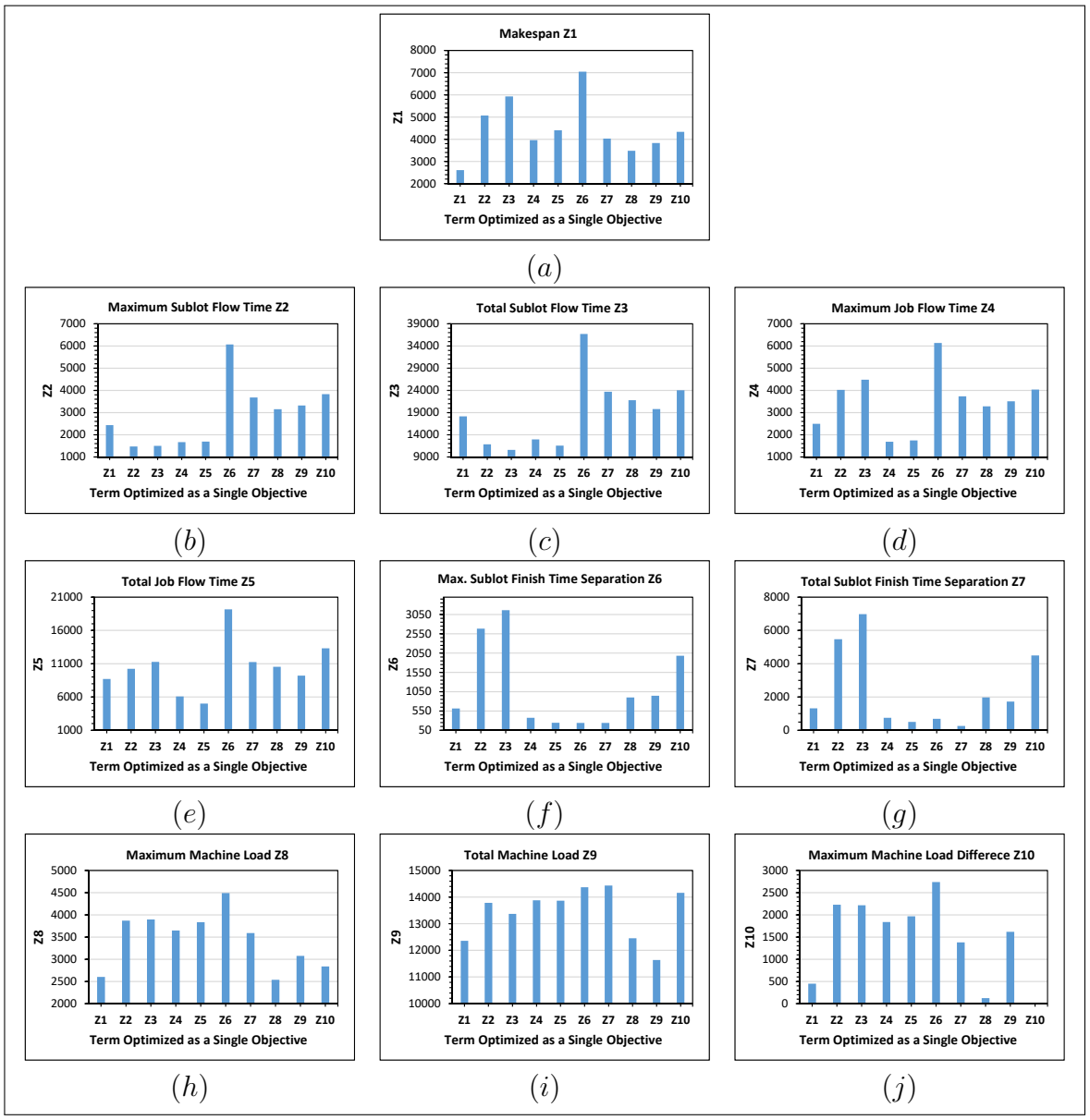
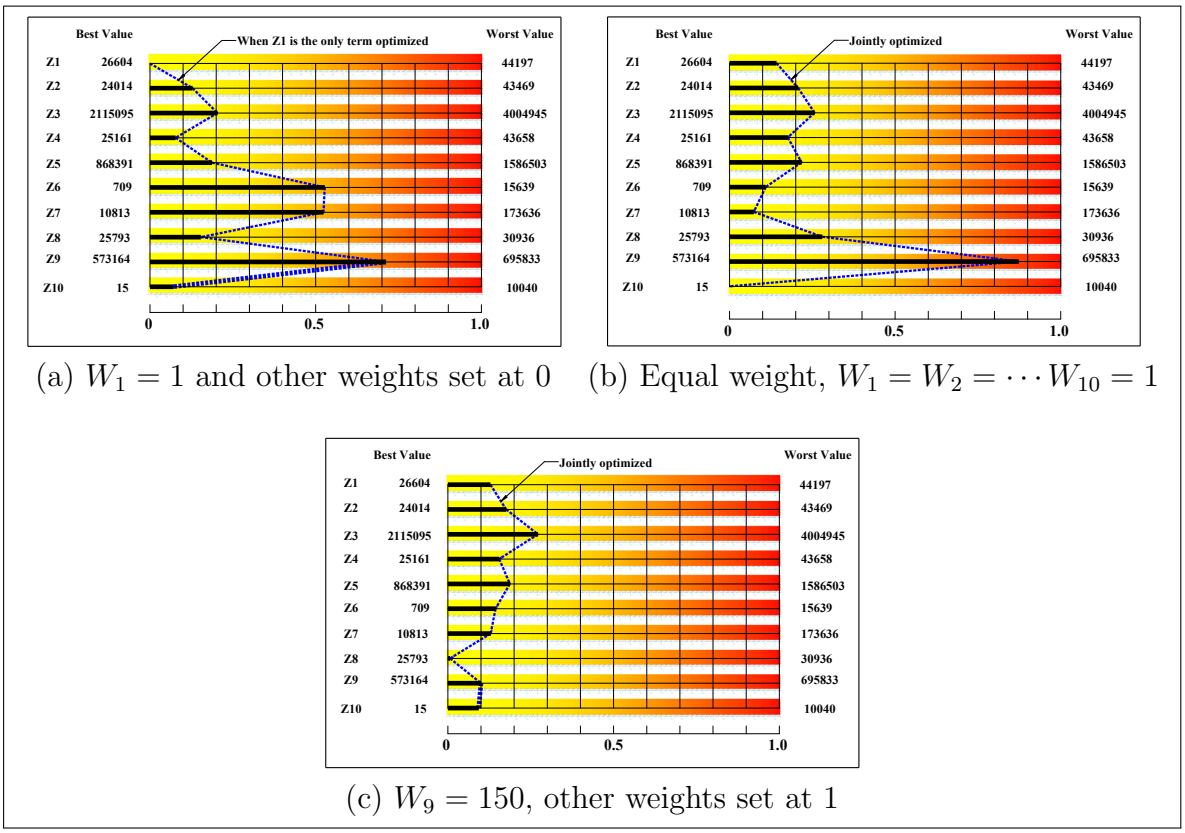


Figure 4.9: Values of the objective function terms in Problem-1 when only one objective function term is optimized

(Z_6 and Z_7, respectively) are significant compared to Case-3 and Case-4. Case-3 and Case-4 are similar to Case-1 and Case-2, respectively. However, in these two cases, Z_6 and Z_7 were also minimized. As it can be seen from the result, Z_6 and Z_7 were reduced substantially with minimal impacts on sublot and job flowtime perforce measures. The result confirms the importance of minimizing sublot finish-time separation along with sublot and job flowtime, which is initially reported in this work.



(a) $W_1 = 1$ and other weights set at 0 (b) Equal weight, $W_1 = W_2 = \cdots W_{10} = 1$

(c) $W_9 = 150$, other weights set at 1

Figure 4.10: Values of the objective function terms when (a) only makespan Z_1 is optimized, (b) all terms are jointedly optimized with equal weights set at one, and (c) all terms optimized with $W_9 = 150$ and other weights set at one.

Another observation from the empirical study in this section is the importance of jointly minimizing the maximum and the total of a performance measure. For instance, let us examine Case-1, Case-5, and Case-6. In Case-1, both maximum (Z_2) and total (Z_3) sublot flowtimes are minimized. Case-5 minimizes Z_2 but not Z_3, and Case-6 minimizes Z_3 but not Z_2. The values of (Z_2, Z_3) in Case-1, Case-5 and Case-6 are (23684, 1017100), (23470, 1195014), and (24578, 1013725), respectively. In shifting from Case-5 to Case-6, Z_2 deteriorates by 4.7%, and Z_3 improves by 15%. Thus, minimizing Z_2 alone results in an unfavorable value of Z_3 and vice versa. By adopting Case-1, Z_2 deteriorates only by 0.9% from its value in Case-5, and Z_3 deteriorates only by 0.33% form its value in Case-6. Thus, instead of minimizing the maximum or the total sublot flowtime alone, it is preferable to minimize both of them simultaneously. By examining Case-2, Case-7, and Case-8, we can also arrive at a similar conclusion regarding Z_4 and Z_5.

In literature, workload balancing in FJSP has been handled either by minimizing the workload of the most loaded machine (maximum workload Z_8) or minimizing the total workload (Z_9). Accordingly, Case-9 minimizes Z_8, and Case-10 minimizes Z_9. However, in both cases, we can see that the difference between the workloads of the most loaded and the least loaded machines (maximum workload difference, Z_{10}) is significant compared to all the cases from Case-1 to Case-8 where Z_{10} is also minimized along with other objective function terms. Thus, for a better workload balancing, it is desirable to minimize Z_{10} along with Z_8 and Z_9. The minimization of Z_{10} to improve workload balancing is reported for the first time in this work.

Table 4.11: Values of the weights of the objective function terms in Problem-4 in eleven different cases

	Objective Function Term Weight									
Case	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}
0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	1	1	1
2	1	0	0	1	1	0	0	1	1	1
3	1	1	1	0	0	1	1	1	1	1
4	1	0	0	1	1	1	1	1	1	1
5	1	1	0	0	0	0	0	1	1	1
6	1	0	1	0	0	0	0	1	1	1
7	1	0	0	1	0	0	0	1	1	1
8	1	0	0	0	1	0	0	1	1	1
9	1	1	1	1	1	1	1	1	0	0
10	1	1	1	1	1	1	1	0	1	0

4.4.2. Performance Evaluation of RGA and 2SGA

Initial Solution Quality

Bajer et al. (2016) and Rahnamayan et al. (2007) argued that the quality of the initial population is an important factor in determining the abilities of evolutionary algorithms to find acceptable solutions with minimal execution time. With this in mind, Defersha

Table 4.12: Average values of the objective function terms in Problem-4 from ten replications in each cases (Cases 0 to 10).

	Objective Function Terms									
Case	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	Z_8	Z_9	Z_{10}
0	24550	23710	1057020	23762	822152	425	2030	22909	565394	559
1	24524	23684	1017100	24206	829235	4642	15476	22657	561306	366
2	24640	23964	1143060	24021	823195	3222	14307	23047	571702	372
3	24891	24183	1053079	24286	847180	375	1337	22717	562619	396
4	24789	24322	1127425	24355	835824	579	2680	22959	569595	376
5	24503	23470	1195014	24297	877040	5370	23064	23073	573483	285
6	24737	24578	1013725	24618	831675	4550	14209	22639	561504	308
7	24406	23822	1176456	23839	874850	4198	17824	22977	571159	258
8	24546	24447	1088981	24476	815426	4218	15152	22872	568343	289
9	24842	23935	998845	23950	811569	84	376	22987	562417	2047
10	24756	23870	958741	23881	804054	94	319	23844	552727	4202

The setup load is the portion of the total workload Z_9 required to perform setup operations.

Table 4.13: Basic features of the problems considered for performance evaluation of the proposed algorithm

Problem	M	J	S_j (max)	O_j (min, max)	NAMPJ* (min, max)
4	25	40	4	(8 15)	(3, 6)
5	30	60	4	(8, 16)	(3, 6)
6	40	80	4	(10, 18)	(2, 8)
7	50	100	4	(10, 20)	(2, 8)

*NAMPJ = Number of Alternative Routing per Operation.

Table 4.14: Algorithm parameters

Parameters	Values
Population Size	2000
Tournament Size Factor α	0.005
Crossover Probability	0.85
Mutation Probability	0.15
Number of generation for the first sage in 2SGA	2500
Total nubmer of genration	10000
$W_1,\ W_2,\ \cdots,\ W_{10}$	1.0

Note: Tournamet size = $\alpha\times$ Population size

and Rooyani (2020) illustrated that one of the key factors for the success of their Two-Stage GA is its ability to find initial solutions with greatly improved makespan. In this research, we further illustrate the ability of Two-Stage Genetic Algorithm in finding improved initial population not only with respect to makespan but also with many other performance metrics of the multi-objective FJSP lot streaming presented in Section 4.2.2. Table 4.15 provides the means and the standard deviations of the objective functions Z_1 to Z_{10} in the initial population of 2000 individuals in Problem-1 and Problem-4. From this table, it can be clearly seen that the mean and the standard deviation of values of the various objective functions in the initial population are greatly improved as we move from RGA to 2SGA. For instance, the mean and standard deviation of maximum sublot flowtime (Z_2) improves by 41 % and 57 %, respectively, in Problem-1 and by 47 % and 79%, respectively, in Problem-4. The histogram for the weighted sum of all the objective function terms of the initial population is displayed in Figure 4.11. The histogram shows that 2SGA results in highly improved initial solution quality in solving the proposed multi-objective FJSP lot streaming problem.

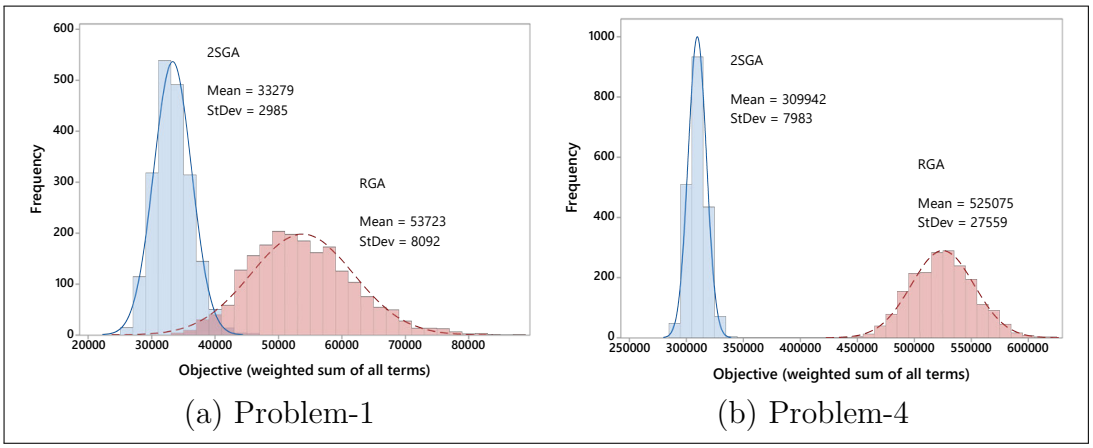
Table 4.15: Mean and standard deviation of the objective function terms in the initial population under RGA and 2SGA.

	Problem-1					Problem-4				
Objective	Mean		StDev		Percentage	Mean		StDev		Percentage
Term	RGA	2SGA	RGA	2SGA	Improvement*	RGA	2SGA	RGA	2SGA	Improvement*
Z_1	6317	3767	972	412	(40, 58)	58513	31277	3293	748	(47, 77)
Z_2	5477	3221	1045	447	(41, 57)	57435	30524	3336	690	(47, 79)
Z_3	30138	18243	6116	2422	(39, 60)	4574999	2476978	299643	89360	(46, 70)
Z_4	5895	3483	1003	438	(41, 56)	57957	30846	3312	705	(47, 79)
Z_5	18954	10829	3420	1035	(43, 70)	2081854	1122475	119664	17273	(46, 86)
Z_6	2896	1491	1171	581	(49, 50)	21250	10607	4788	2282	(50, 52)
Z_7	6297	3084	2772	1232	(51, 56)	248166	116497	40929	15422	(53, 62)
Z_8	5058	3336	894	291	(34, 67)	37728	29216	2621	538	(23, 79)
Z_9	14816	13835	596	550	(7, 8)	682856	667035	6709	6442	(2, 4)
Z_{10}	3789	1197	1228	532	(68, 57)	19471	5945	3411	1183	(69, 65)

*The percentage improvement in Mean and StDev (Mean, StDev) in the initial population achieved by 2SGA.

Convergence Behaviors

The previous section illustrated that 2SGA resulted in an improved initial population in all the objective function terms. In this section, we compare the convergence behavior



(a) Problem-1 (b) Problem-4

Figure 4.11: The distribution of the objective function of initial populations of 2SGA and RGA both in Problem-1 and Problem-4.

of 2SGA and RGA while solving large-size problems (Problems 4 to 7). The basic features of these problems are given in Table 4.13. The parameters of the GAs used in this numerical example are given in Table 4.14. Figure 4.12-(a) to (j) show the convergence along the objective function terms Z_1 to Z_{10}, respectively, of 2SGA and RGA in solving Problem-4 while all these terms are simultaneously optimized with equal weight ($W_1 = W_2 = \cdots = W_{10} = 1$). Each convergence curve is an average of 40 replications. From these convergence curves, we can see that 2SGA was able to converge more rapidly than RGA along Z_1 to Z_5, Z_8, and Z_9. In terms of these objective function terms, 2SGA was able to find better solutions just in a few hundred generations than those determined after more than 10,000 generations by RGA. In terms of Z_6, Z_7, and Z_{10}, RGA was able to converge more rapidly than 2SGA. However, 2SGA was able to catch up with RGA just in a few hundreds of generations right after it changed search stage, which occurred at 2500 generation. Figure 4.12-(k) is the convergence of 2SGA and RGA in terms of the weighted sum of all the objective function terms, which clearly shows the superiority of 2SGA over RGA. From the convergence graphs, the first stage of 2SGA was able to achieve convergence within the first few hundreds of generations. For instance, if 2SGA changed its search stage at 1000 generation, it could provide highly improved solutions in just 3000 generations that cannot be archived using RGA after many thousands of generations. 4.12-(l) depicts the histograms of

the objective function of the final solutions in 40 trials both in 2SGA and RGA. In these histograms, 2SGA achieves about 9.5% and 35.6% improvements in the mean and standard deviation, respectively. An improvement in the standard deviation by 2SGA represents its robustness in finding good solutions more consistently than RGA. Similar results were obtained while solving Problems-5, 6, and 7, as shown in Figure 4.13. The computational times required by 2SGA and RGA to complete the 10,000 generations using the parameters in Table 4.14 are approximately 120, 335, 840, and 1410 minutes in Problems 4, 5, 6, and 7, respectively.

Improvement through Parallelization

Parallelizing genetic algorithms using a high-performance parallel computing platform has been well recognized as a viable technique to enhance their abilities in solving many complex and large-size problems. Its application in solving shop scheduling problems has also been widely reported as reviewed in Luo and El Baz (2018). In this research, we adopted a randomly connected multi-population parallel GA (P-GA) proposed in Defersha and Chen (2008) to illustrate the performance improvement that can be achieved in both RGA and 2SGA. The P-GA consists of several subpopulations where each of them is assigned to a dedicated CPU. A subpopulation evolves independently and communicates periodically by sending and receiving selected solutions to and from other subpopulations. Whenever communication occurs, the CPU with rank 0 randomly generates a communication matrix and broadcasts it to all other CPUs. The migration of the copies of the selected solutions follows the route generated according to the communication matrix. An example communication matrix and the resulting migration route for a small instance of parallelization are depicted in Figure 4.14 where the CPUs are ranked from 0 to 6. The density of the communication matrix, the frequency of communication, and the strategy for the selection and replacement of migrants from the source and to the destination subpopulations are key parameters for this parallelization technique. An investigation of these parameters is not within the scope of this research.

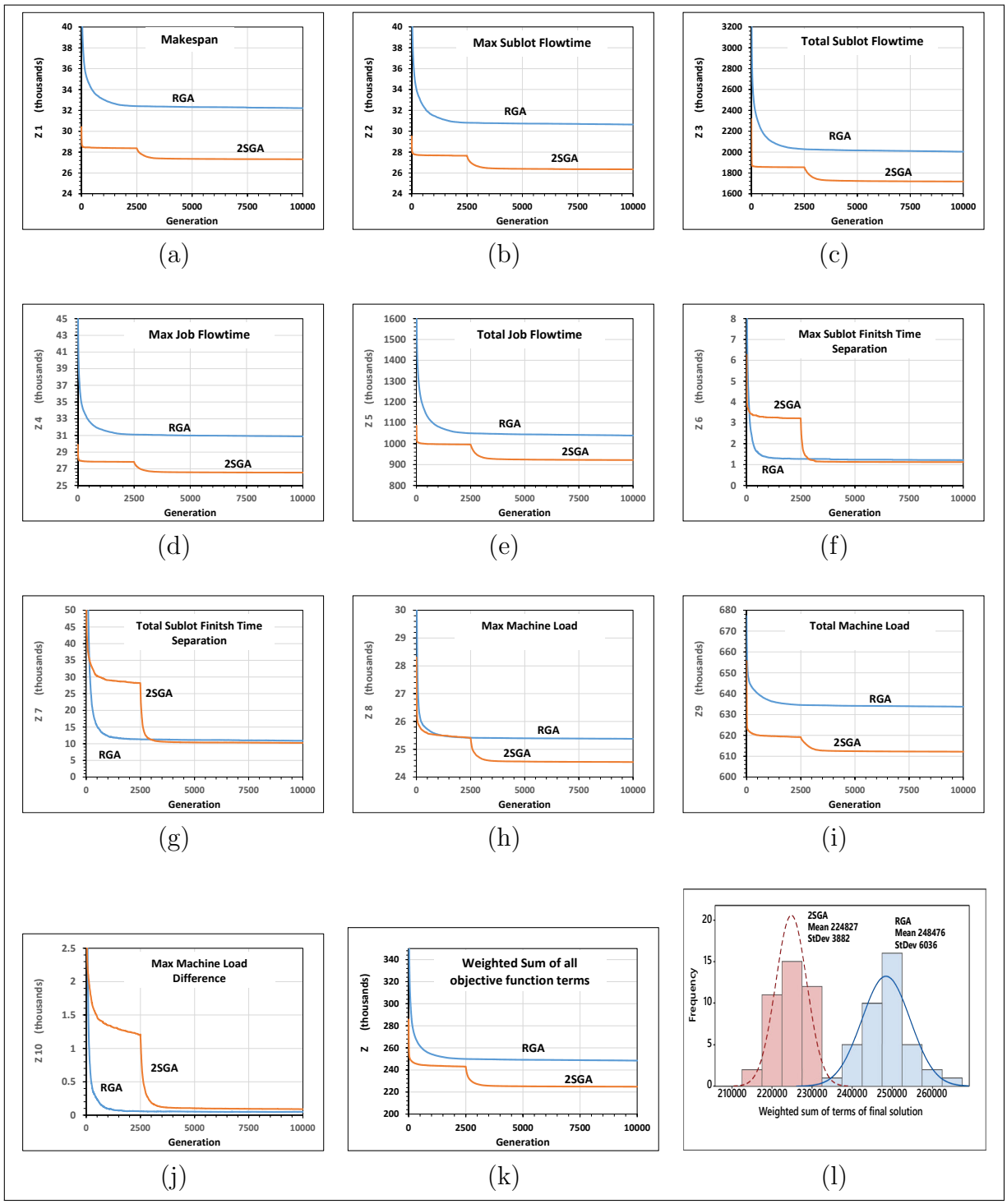
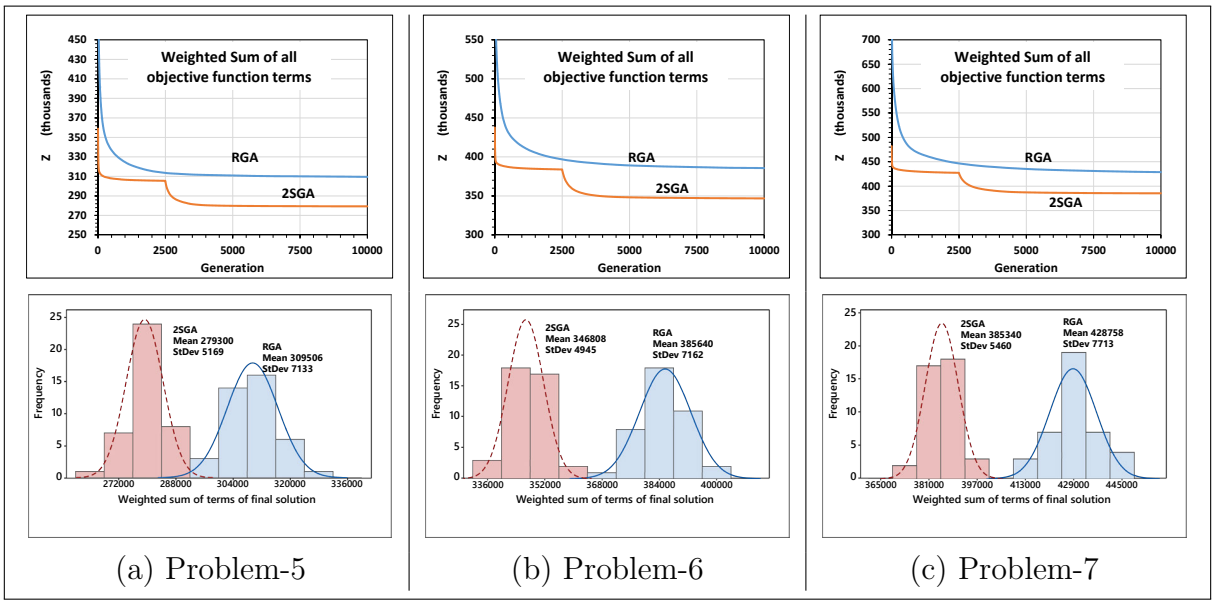


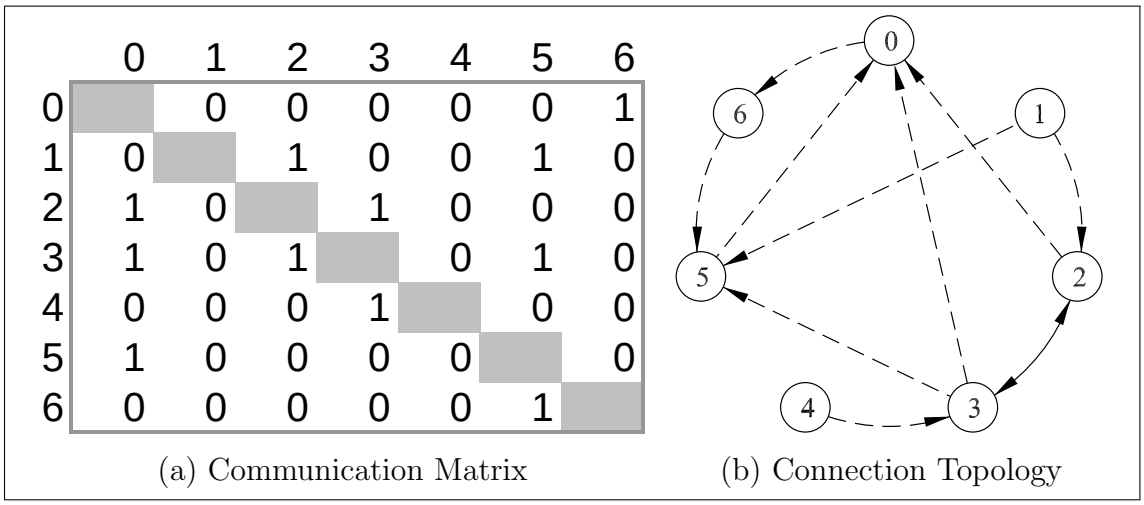
Figure 4.12: The convergence of 2SGA and RGA in solving Problem 4 while all the objective terms are optimized simultaneously. (Each convergence graph is an average of 40 trials, and (l) is the histogram of the final values of objective function in these 40 trials).



(a) Problem-5 (b) Problem-6 (c) Problem-7

Figure 4.13: The convergence of 2SGA and RGA in solving Problems 4, 5 and 6. (Each convergence graph is an average of 40 trials. The histograms are for the final values of objective function in 40 trials.)

In this study, we used a total of 80 concurrently available CPUs in high performance parallel computing platform to implement the Parallel RGA (P-RGA) and the Parallel 2SGA (P-2SGA). Problems 4, 5, 6, and 7 were solved using both the sequential and the parallel versions of these algorithms using a subpopulation size of 2000. The subpopulations were allowed to communicate every 30 generations. The change of stage for 2SGA occurs at 2500 generations. The computation was terminated after 10,000 generations. The resulting convergence graphs are given in Figure 4.15. From these graphs, one can see that parallelization brings performance improvement both in RGA and 2SGA. However, the very important finding in this investigation is that the sequential 2SGA using a single CPU outperforms the parallel RGA that uses 80 CPUs. This finding asserts the superiority of 2SGA over RGA in solving the proposed multi-objective FJSP lot streaming problem.



(a) Communication Matrix (b) Connection Topology

Figure 4.14: Randomly connected topologies for a given communication matrices (adopted from Defersha and Chen (2008)). Note: Communication matrix and topology is generated every time before solution migration occurs.

4.4.3. Empirical Analysis of Algorithm Parameters

Selection Operators

In this section, we present comparative empirical studies on the various selection operators presented in Section 4.3.5. The comparisons are presented in terms of the convergence behavior of 2SGA in solving Problem 4, whereas similar results were obtained in solving several other problems. The first of these imperial studies is aimed at comparing the three fitness transformation functions in Eqs. (4.60) to (4.62) used in the proportional selection method. Figure 4.16 provides the average convergence from ten test runs using these three different fitness transformation equations. As it can be seen from this figure, Eqs. (4.60) and (4.61) resulted in similar convergence behaviors of the algorithm. In contrast, the transformation function in Eq. (4.62) resulted in a much better convergence of 2SGA in using the proportional selection method.

The second empirical study investigates the impact of tournament size in tournament selection. Figure 4.17 depicts the results of this study. This figure shows that tournament selection with a smaller tournament size is preferred in solving the proposed mathematical model using 2SGA. Lastly, a comparison of proportional, linear

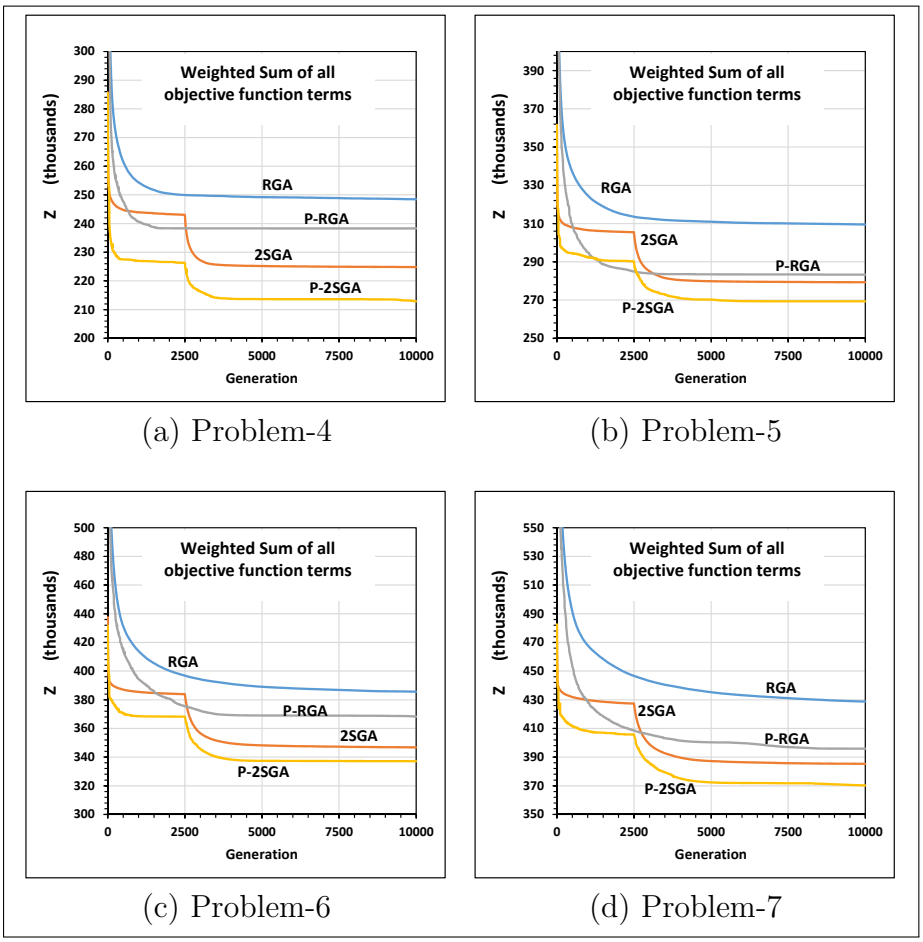


Figure 4.15: Improvements of convergence behaviours of RGA and 2SGA using parallelization technique.

ranking and tournament selection is conducted where the resulting convergence graphs are given in Figure 4.18. This figure shows that the tournament selection resulted in an improved convergence of the proposed 2SGA. Hence, tournament selection with a small tournament size is the preferred selection operator in the proposed algorithm.

Crossover and Mutation Probabilities

In the proposed algorithm, there are seven crossover and six mutation operators. Assigning probabilities individually for these thirteen operators and simultaneously tuning them can be a daunting task. Instead, in this work, we suggested the crossover and mutation operators be assigned one crossover and one mutation probability, respectively.

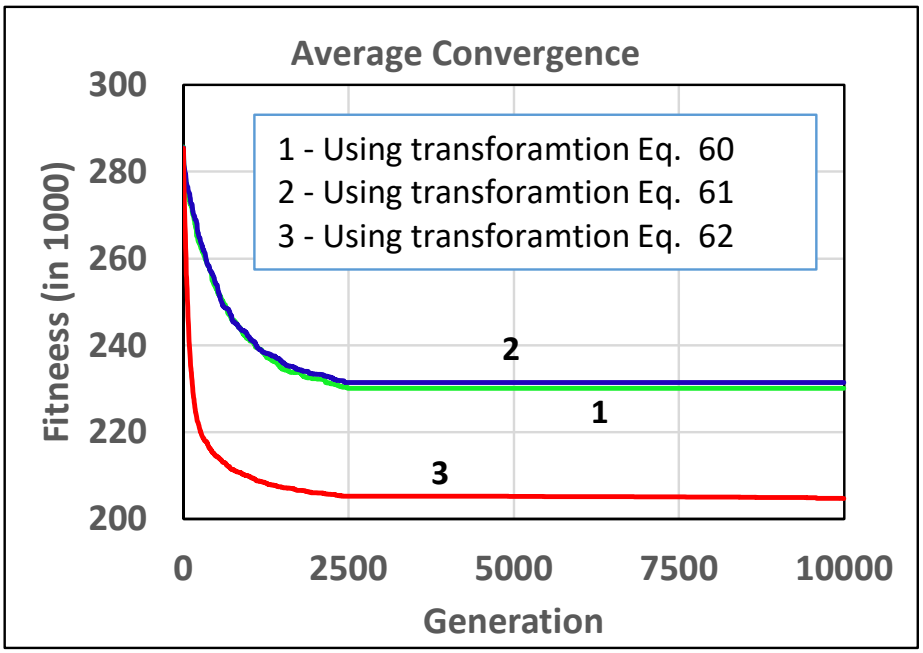


Figure 4.16: Average convergence from ten test runs of 2SGA using proportional selection under three different fitness transformation while solving Problem-4.

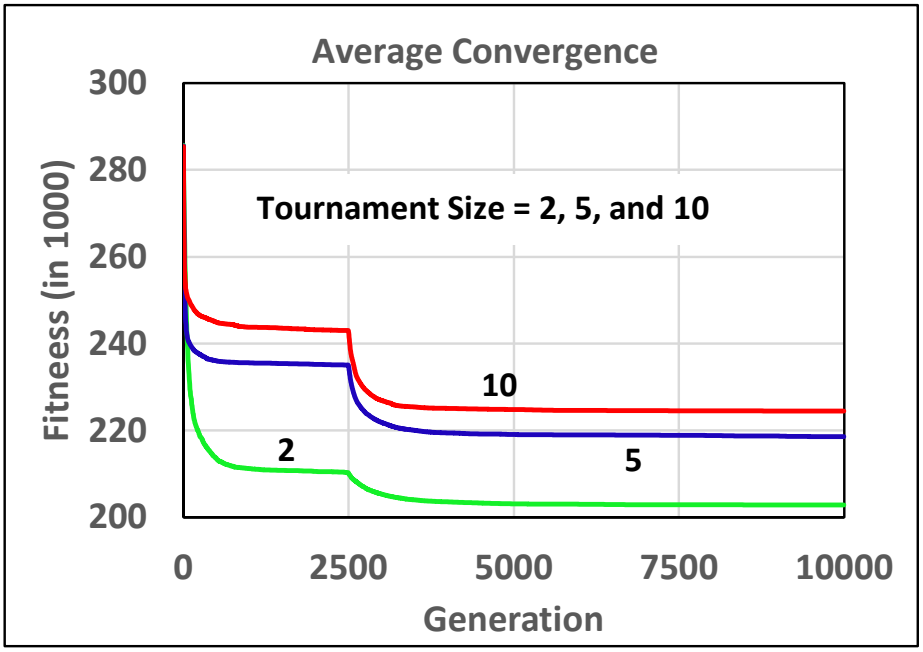
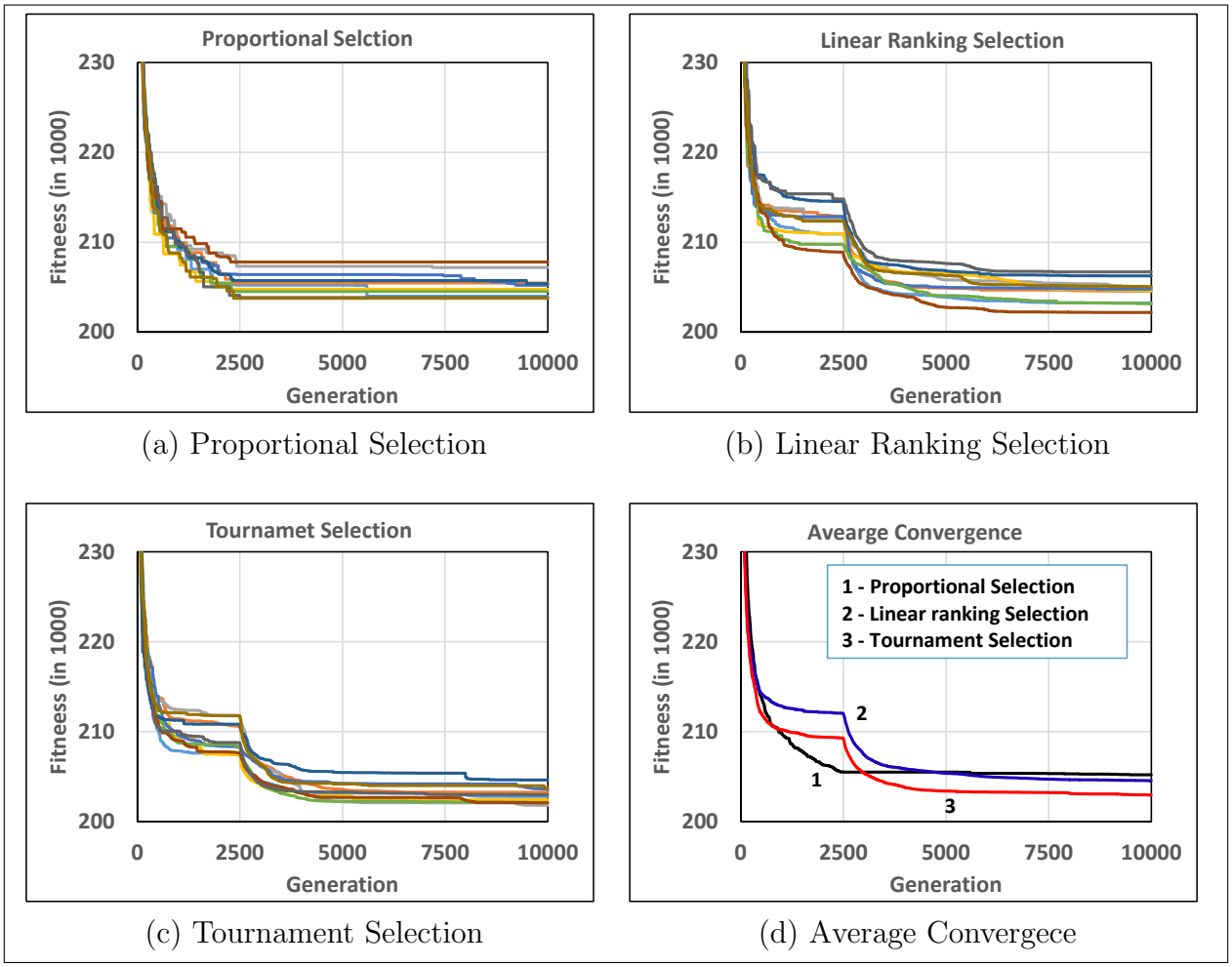


Figure 4.17: Average convergence from ten test runs of 2SGA using tournament selection under different tournament sizes while solving Problem-4.

With this scheme, we performed an Analysis of Variance (ANOVA), where mutation and crossover probabilities are the only two factors, and the objective function is the



(a) Proportional Selection (b) Linear Ranking Selection

(c) Tournament Selection (d) Average Convergece

Figure 4.18: Convergence of 2SGA under three different selection operators while solving Problem-4.

response. We chose six levels for each of these factors. The levels for the mutation and crossover probabilities are {0.05, 0.15, 0.25, 0.35, 0.45, 0.55} and {0.75, 0.80, 0.85, 0.90, 0.95, 1.00}, respectively. For each factor level combination, we conducted five replications. Hence, the experiment requires solving a problem 180 times. The genetic algorithm uses a different seed for its random number generator in each replication of the experiment.

The results of ANOVA for Problem-4 are presented in Table 4.16 and Figure 4.19. The P-values corresponding to the main effects of mutation and crossover probabilities are zero, implying that these two factors have statistically significant effects on the

final solution quality. On the other hand, the P-value for the interaction effect is very high (compared to a typical significance level $\alpha = 0.05$), which indicates the absence of interaction between these two factors. This lack of interaction simplifies parameter tuning allowing the user to optimize them independently. The plots of the main effects in Figure 4.19 show that the mutation probability needs to be set close to 0.35, and the crossover needs to be set at higher values between 0.90 and 1.00. The residual plots do not indicate unusual patterns, confirming the adequacy of the ANOVA. The Analysis also rendered very similar results on several other problems of a varying size considered in this research. Hence, the recommended values of the mutation and crossover probabilities can be used to solve different sets of problems using the proposed algorithm.

Table 4.16: Output of Analysis of Variance for Problem-4.

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Mutation Probability	5	1418299280	283659856	93.7	0.000
Crossover Probability	5	91944634	18388927	6.03	0.000
Mutation Probability*Crossover Probability	25	49082335	1963293	0.64	0.901
Error	144	438889368	3047843		
Total	179	1998215617			

DF = Degrees of Freedom; Adj SS = Adjusted sum of square; Adj MS = Adjusted mean square.

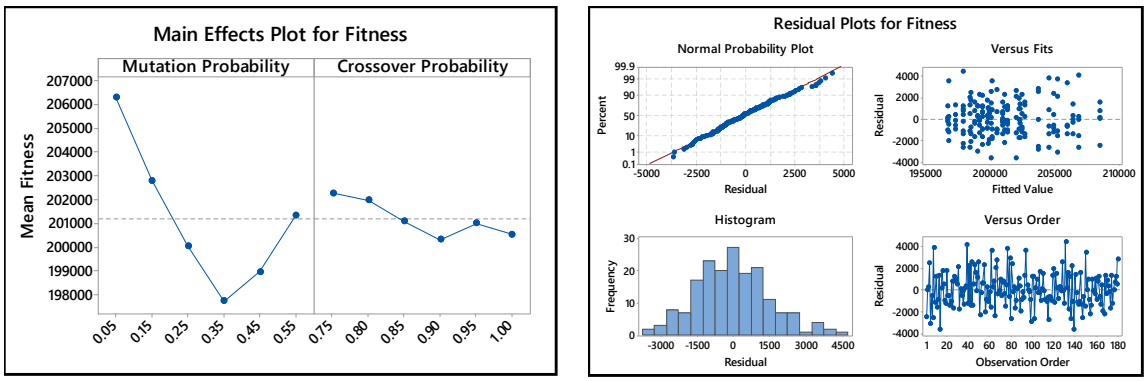


Figure 4.19: Main Effect and Residual Plots of the Analysis of Variance

4.5. Discussion and Conclusion

The recent trend in manufacturing scheduling is in developing efficient algorithms for very complex and comprehensive scheduling problems. Following this trend, we developed an efficient Two-Stage Genetic Algorithm and testing its effectiveness on classic and comprehensive FJSPs that is presented in Chapter 3 and published in Rooyani and Defersha (2019) and Defersha and Rooyani (2020). In this chapter, we extended the application of the Two-Stage Genetic Algorithm on a comprehensive FJSP that already incorporates (1) sequence-dependent setup time, (2) attached and detached nature of setups, (3) machine release date, and (4) lag time in two folds. We expand the comprehensive FJSP (1)to solve lot streaming problem while at the same time (2) to incorporate multiple objectives. The objective function terms included in this model are the minimization of (1) makespan, (2) maximum sublot flowtime, (3) total sublot flowtime, (4) maximum job flowtime, (5) total job flowtime, (6) maximum sublot finish-time separation, (7) total sublot finish-time separation, (8) maximum machine load, (9) total machine load, and (10) maximum machine load difference. We generated several numerical to illustrate the interaction of the various objective function terms and their relevance in providing better solution quality, and concluded the greater need for multi-objective optimization in larger problems. We also investigated the ability of the Two-Stage Genetic Algorithm to jointly optimize all the objective function terms and showed that the algorithm can generate initial solutions that are highly improved in all of the objective function terms. It also outperforms the regular genetic algorithm in convergence speed and final solution quality in solving the multi-objective FJSP lot streaming. We also demonstrate that high-performance parallel computation can further improve the performance of the Two-Stage Genetic Algorithm. Nevertheless, the sequential Two-Stage Genetic Algorithm with a single CPU outperforms the parallel regular genetic algorithm that uses many CPUs, asserting the superiority of the Two-Stage Genetic Algorithm in solving the proposed multi-objective FJSP lot streaming. The result of this research is published in Rooyani and Defersha (2022).

Following are the list of observations and conclusions:

1. The magnitude of the severity of a single objective optimization on the objective function terms that are not incorporated increases as the problem size increases. The result emphasizes the need for multi-objective optimization in real industrial scheduling problems that are usually large in size.

2. Optimizing sublot flowtime is more desirable than optimizing job flowtime. However, optimizing both terms simultaneously can also result in favorable solutions with respect to the overall flowtime performance.

3. In lot streaming, one sublot of a given job may be finished much sooner than the other sublot of the same job. This may increase work-in-process inventory. The newly proposed objective function terms (minimize the maximum sublot finish-time separation and total sublot finish-time separation) can alleviate this problem with minimal impact on sublot and job flowtime.

4. Instead of minimizing the maximum or the total sublot flowtime, it is advantageous to minimize both its maximum and total values simultaneously. The same is true with the other performance measures (job flowtime, sublot finish-time separation, machine workload).

5. Workload balancing in FJSP may not be fully achieved by minimizing the maximum or the total workload or both. A newly proposed objective function term (minimizing the maximum workload difference), can result in a better workload balance when considered along with the minimization of the maximum and/or the total workload.

6. The solution representation and the corresponding decoding of the first stage of the Two-Stage Genetic Algorithm can generate initial solutions that are highly improved in all the ten objective function terms.

7. The Two-Stage Genetic Algorithm can jointly optimize all the ten objective function terms of the multi-objective FJSP lot streaming considered in this research and greatly outperform the regular genetic algorithm.

8. Parallel computation can bring performance improvement in both the Two-Stage GA and the regular GA. However, the very important finding is that the sequential Two-Stage GA using a single CPU outperforms the parallel regular genetical algorithm that uses many CPUs in solving the proposed multi-objective FJSP lot streaming problem.

9. The performance of the proportional section method can be significantly improved by the appropriate choice of the fitness transformation function.

10. Proportional, linear ranking and tournament selection can result in comparable performance. However, tournament selection with smaller size of tournament slightly outperforms the other two.

11. Analysis of variance shows the lack of interaction between mutation and crossover probabilities. Thus, the two probabilities can be tuned independently.

The Two-Stage Genetic Algorithm may not be directly applicable in scheduling problems with the objective of minimizing earliness-tardiness. In particular, the greedy nature of the first stage is based on finding a schedule that finishes the jobs as early as possible, which is against minimizing earliness. For instance, finishing jobs too early may represent excess work-in-process in a JIT environment. Chapter 5 includes the development of a modified Two-Stage Genetic Algorithm to incorporate the minimization of earliness while including the outsourcing.

Chapter 5

Earliness and Tardiness Scheduling with Assembly and Outsourcing Extension

5.1. Introduction

In chapter 3, we have implemented the Two-Stage GA on the FJSP with the machine release date, lag time, and detached or attached sequence dependent setup time. Chapter 4 extended the application of Two-Stage GA to a multi-objective FJSP with 10 objectives varied from minimizing makespan and flowtime to machine load maximization. Several numerical examples proved the ability of 2SGA to jointly optimize all of those objectives. However, as we will describe in this chapter, the Two-Stage GA has a pragmatic issue when it comes to minimizing the earliness and tardiness of the jobs with the desired completion time or due date. It was expected since the most novel idea in developing the Two-Stage GA is adding a greedy stage to assign the operation to the machine that can complete it the soonest. So in this chapter, we will extend the work further to address an FJSP with jobs with due dates, earliness, and tardiness penalties, along with other jobs that should be completed as soon as possible. Additionally, these jobs have assembly relationships with some outsourced operations, unlike the classical FJSP

approach, where the jobs are independent, and each operation has at least one capable machine. The FJSP model in this chapter still has detached or attached sequence dependent setup time and is multi-objective with 3 minimization objectives of "On-time completion" for the jobs with due dates and "makespan" and "total completion time" for the jobs with no due dates.

5.1.1. Earliness and Tardiness (E/T) scheduling

In many manufacturing systems, especially Make To Order (MTO) types, jobs have due dates or planned completion dates, either set by the customer or the manufacturing company itself. However, it is clear that based on the production limitations, not every job finishes on time. The jobs completed early must be stored as finished-goods inventory, causing earliness costs, and the jobs completed after their due dates incur penalties and customer dissatisfaction. In order to avoid these unnecessary costs and keep the competitive edge in the current market conditions and fierce global competitiveness, many manufacturing companies have adopted the Just-In-Time (JIT) strategy that demands the scheduling system to complete the jobs as closely as possible to their due dates and to minimize the sum of earliness and tardiness costs.

The JIT inventory strategy for supply chain management has been applied in North American industries since the early 1970s. However, it was about a decade later that researchers like Baker and Scudder (1990) started to address scheduling problems to minimize earliness and tardiness costs (Kelbel and Hanzálek (2011)). Mousakhani (2013) reported that minimization of total tardiness is the most common objective function among due date-based criteria. Similarly, Fuchigami and Rangel (2018) reported tardiness measures as the second most popular objective function right after makespan among practical scheduling studies and case studies they have reviewed. Please refer to section 2.2.3 for a more detailed literature review on Earliness and Tardiness (E/T) scheduling.

These E/T scheduling systems aim to reduce inventory costs (also called waste),

including quantity, value, and shelving duration of the work-in-process (WIP) and finished good inventory. In addition to the direct inventory cost, jobs that are finished early cause cash flow issues, production congestion, and risk of damaged goods due to extra material handling. Similarly, late jobs in addition to customer dissatisfaction may also result in lateness penalties, contractual liquidated damages, customer loss, and idle times in subsequent production processes. So E/T scheduling, as one of the core technologies to support JIT production philosophy, became an important research branch in the production scheduling field.

Most classic flexible job shop scheduling problems use machines, jobs, and operations as decision variables. However, E/T scheduling as a modified type of FJSP that includes JIT objectives needs to consider other decision variables. Among different decision variables, the job release time is one of the most critical variables that must be controlled to minimize earliness and tardiness costs. Zambrano Rey et al. (2015) reports only a few studies in which release dates are considered in the algorithm's encoding scheme and then lists four types of methods that researchers considered the job release time as below:

1. Immediate release where jobs are released as soon as they arrive or become available. It is the most common type, either when all jobs are released at time zero or when they arrive at the fixed value or randomly,
2. Load-limited methods where jobs are released according to the current workload,
3. Release based on the flowtime and due date information,
4. Job release time considering both workload and job due dates.

As we will discuss in 5.2 and 5.3 in more detail, our model delays the start of the first operation of the jobs with due dates as long as is required to minimize E/T costs, even though they are ready to be released at time zero.

5.1.2. Subassembly Requirement

Assembly Job Shop (AJS) is an extension of job shop (JS) where some jobs have assembly relationships, unlike the classic JSP and FJSP approach, where the jobs are independent.

It is common in the real world that the final product is made of several components (purchased items) and/or subassemblies built in-house and should go through a sequence of operations before the assembly stage. Even these subassemblies can have their own components and subassemblies. This means any job with subassembly cannot start unless all the subassemblies (child) are finished. Similarly, the subassembly jobs are not considered complete even after all their operations are finished. They must be assembled with other subassemblies into the final product. So the assembly scheduling process concerns both resource availability and completion of all subassemblies on several levels. This "assembly requirement or constraint" is similar to the classical "sequence constraint" but between the different jobs instead of different operations of a single job. As discussed in chapter 1, FJSPs are among the most difficult optimization problems and are NP-Hard, so we should consider that assembly requirement increases the difficulty of FJSP even further. Although AJS has a long presence in industrial civilization history, the Assembly Scheduling Problem (ASP) only appeared in the research world for the first time in the 1960*s* to solve a multilevel assembly scheduling problem under a random environment (Li et al. (2022b)). We provided a more detailed literature review of the ASPs in section 2.2.4.

The hierarchy of the product tree is called the Bill of Material (BOM), usually presented in the hierarchical format and indicates all the components and subassemblies and their required quantities to build a single unit of the final product. In the hierarchical BOM, the highest level displays the final product, and the lowest level all the purchased components and raw materials. The subassemblies will appear in the middle levels (can be several levels) along with some other components being assembled into the next level product. All these components and subassemblies should be processed (and obviously scheduled) from the bottom up to build the final product. BOM is usually translated into a matrix named Precedence Constraint Matrix (PCM) in order to be used in scheduling problems and understood by computer programs. Row and column numbers of the precedence constraint matrix correspond to job numbers, and the elements indicate the assembly relationship of the jobs, usually in binary format. We also used PCM in both

the mathematical modeling of our problem (described in section 5.2) and the Two-Stage GA coding (described 5.3) to identify the subassembly relationship between jobs.

5.1.3. Outsourcing Extension

With the continuous promotion of manufacturing globalization and the increasing complexity of products, manufacturing companies have this opportunity to define their core business and become highly specialized in it while outsourcing the auxiliary processes. This is specifically more common in industrially advanced regions, such as southwestern Ontario, where various manufacturing enterprises are densely distributed and low-cost outsourcing services are readily available. The author also witnessed in his career working with several manufacturing companies that almost every manufacturer is outsourcing some processes. Outsourcing allows manufacturers to focus their resources to get a competitive advantage on what they are good at and outsource the necessary auxiliary processes instead of trying to share the limited capital fund to acquire necessary machines and professionals. Another situation that motivates manufacturing to outsourcing is when on-time delivery is critical and subcontracting is possible at a reasonable cost while the company is at full capacity. This is becoming more important when production lead time and on-time delivery affect customer satisfaction in this competitive market. Timely delivery can even be vital for the survival of some companies. For example, for suppliers of big automotive OEMs where any delay in delivery can shut down their production line and cost the supplier tremendously as a penalty or newspaper printing houses that cannot miss a very narrow delivery window. Outsourcing benefits manufacturing companies in different ways, including reducing production bottlenecks to improve lead time, decreasing direct and indirect manufacturing costs, or enabling them to produce more variety of products. Anyway, outsourcing is a common practice in today's manufacturing companies due to several reasons like existing manufacturing constraints and capacity limitations or because of the advantages of external vendors.

Unlike the popularity of outsourcing in the real world, most scheduling models did not consider outsourcing and assumed all the jobs and operations being done in-house.

This can be due to the fact that outsourcing introduces a whole level of complexity to scheduling problems. As we discussed in 2.2.5, not many studies have addressed job shop scheduling with outsourcing options. So in this chapter, we also incorporated this critical feature in our FJSP model. Outsourcing is either mandatory (no in-house capability) or optional (can be done internally too). Also, the company can outsource the whole job or only a few auxiliary operations of different jobs. Our model covers different scenarios, jobs can be completely outsourced or have several outsourced operations. It also can outsource a specific process of one job while doing it in-house for another job. However, every single operation is either outsourced or done in-house. These considerations come from the author's experience that most manufacturing companies rely on third-party vendors for different auxiliary operations (like special welding, coatings, and machining). Also, they outsource some processes or the whole job due to limited resources or on-time delivery considerations even though they can do it in-house. As described in both the mathematical model in section 5.2 and the Two-Stage GA coding in section 5.3, an outsourced operation in our model has a lead time that indicates the minimum time required between completion of the previous operation and becoming available for outsourcing till it comes back and is ready for the following operation which includes the transportation time.

5.2. Mathematical Modeling

In this section, we introduce the mathematical model of our proposed FJSP in the presence of outsourcing, subassembly, and sequence dependent setup that can be either detached or attached. The jobs in this model either have due dates with E/T cost or should be finished as early as possible to minimize makespan and total completion time. This MILP model can be programmed using any optimization packages like Lingo, Lindo, CPLEX, or GAMS to solve small-size problems. However, to solve larger size problems in a reasonable time, we have to use meta-heuristic techniques.

5.2.1. Problem description and notations

In Section 3.2, we introduced both classic and comprehensive FJSP that we used to test the developed Two-Stage GA, with Section 3.2.1 describing the full mathematical MILP model with and its notations and equations. All the model parameters and variables are also listed in the "List of Symbols". So in this section, we only introduce the newly added features and skip describing the FJSP with the machine release date, lag time, and detached or attached sequence dependent setup time concepts.

As we discussed, each job can have multiple subassemblies on multiple levels. To accommodate the presence of subassembly in our model, $K_{j,j'}$ has defined and equals to 1 if job j' is an immediate child (subassembly) of job j in Bill of Material (BOM) and zero otherwise. The parent job can only be started if all its subassemblies are completed. Also, in this model, each operation is either outsourced (if $U_{j,o}$ equals 1) with the outsourcing lead time (turnover time that includes the shipping and receiving time) of $R_{j,o}$, or it is processed in-house (if $U_{j,o}$ equals 0) with a processing time of $B_{m,j,o}$.

Regarding completion (or shipping) time, jobs have a due date (or promised shipping date) of D_j or should be shipped as soon as completed. E_j equals 1 if the job has promised ship date and equals 0 otherwise. For the jobs with a promised shipping date, minimizing the total earliness/tardiness (being either shipped early or late) has been defined as the objective function. While minimizing the total completion time and the makespan are the objective functions for the jobs with no promised shipping date. Minimizing total completion time is similar to makespan. However, it tries to shorten the completion time of every job, while makespan deals with only the longest completion time and does not concern with the rest.

Here are other parameters and variables that have been used in the proposed model:

Additional Parameters:

$B_{m,j,o}$ Process time of operation o on machine m for whole batch of job j (only if operation o is not outsourced and machine m is capable of processing).

$S^*_{m,j,o}$ Setup time of operation o of job j on on machine m if operation o is the first operation to be processed on machine m.

$S_{m,j,o,j',o'}$ Setup time of operation o of job j on on machine m where operation o' of job j' is operation processed immediately before.

$A_{j,o}$ A binary data equal to 1 if the setup of operation o of job j is attached, or 0 if this setup is detached.

$R_{j,o}$ Lead time of operation o of job j that is outsourced.

D_j Due date (promised shipping date) for job j.

R_m Maximum number of production runs of machine m where each production run is indexed by r or $u = 1, 2, ..., R_m$.

Ω Large positive number.

Variables:

Continuous Variables:

C_{max} Makespan of the schedule.

$\hat{c}_{m,r}$ Completion time of run r of machine m.

$c_{j,o}$ Completion time of operation o of job j.

Binary Variables:

$x_{m,r,j,o}$ Binary variable that takes the value 1 if the run r on machine m is assigned to operation o of job j, 0 otherwise.

$z_{m,r}$ Binary variable that is equal to 1 if run r of machine m has been assigned to any operation, 0 otherwise.

5.2.2. Mathematical formulation

Objective Functions:

This model has 3 objective functions to cover both types of jobs with and without due dates. The objective function (Eq. (5.4)) minimizes the makespan that is only for the jobs without due dates. Similarly, the second objective function (Z_2) is the total completion time for the jobs with no due dates. The Z_2 (Eq. (5.2)) assures the completion time of every job will be minimum, in other words, they all finish as soon as possible. It is a good addition to the makespan objective function that only minimizes the longest completion time, and the other jobs can still use the slack (float) time. The third objective function (Z_3) is minimizing the total earliness and tardiness (Eq. (5.3)), which obviously is for the jobs with due dates. This equation is not linear due to the presence of the absolute value function. Since optimization packages like Lingo, Lindo, CPLEX, or GAMS prefer MILP over MINLP (Mixed Integer Nonlinear Programming), we will transform it to linear form in equation Eq. (5.24).

There are many ways to aggregate the objective functions, but as explained in 4.3, we will use the scaled weighted aggregated objective function shown in Eq. (5.4). Each k^{th} objective function will be multiplied by the weights W_k and scaling of Ψ_k. Weights W_k represents decision makers' preference that can be any number, and Ψ_k is a simple scaling mechanism explained in Eq. (5.5) where $Z_k^{Ini-max}$ represents the maximum value of objective Z_k in the initial GA population. So the magnitude of the maximum values of objective function terms Z_2, and Z_3 will have the same values as the maximum value of Z_1.

$$Z_1 = c_{max}; \quad \forall(j)|\ E_j = 0 \tag{5.1}$$

$$Z_2 = \sum_{j=1}^{J} c_{j,o}; \quad \forall(j,o)|\ E_j = 0\ \&\ o = O_j; \tag{5.2}$$

$$Z_3 = \sum_{j=1}^{J} |c_{j,o} - D_j|; \quad \forall (j,o) |\ E_j = 1\ \&\ o = O_j \tag{5.3}$$

$$Z = \sum_{k=1}^{3} W_k \cdot \Psi_k \cdot Z_k \tag{5.4}$$

$$\Psi_k = \frac{Z_1^{Ini-max}}{Z_k^{Ini-max}} \tag{5.5}$$

Subject to:

$$c_{max} \geq c_{j,o}; \quad \forall (j) |\ E_j = 0\ \&\ o = O_j \tag{5.6}$$

$$\hat{c}_{m,r} \geq c_{j,o} + \Omega \cdot x_{m,r,j,o} - \Omega; \ \ \forall (m,r,j,o) \tag{5.7}$$

$$\hat{c}_{m,r} \leq c_{j,o} - \Omega \cdot x_{m,r,j,o} + \Omega; \ \ \forall (m,r,j,o) \tag{5.8}$$

$$\hat{c}_{m,1} - B_{m,j,o} - S^*_{m,j,o} - \Omega \cdot \ x_{m,1,j,o} + \Omega \geq 0; \ \ \forall (m,j,o) \tag{5.9}$$

$$\hat{c}_{m,r} - B_{m,j,o} - S_{m,j,o,j',o'} - \Omega \cdot (x_{m,r,j,o} + x_{m,r-1,j',o'}) + 2\Omega \geq \hat{c}_{m,r-1};$$
$$\forall (m,r,j,o,j',o') | (r > 1)\ \&\ ((j,o) \neq (j',o')) \tag{5.10}$$

$$\hat{c}_{m,1} - B_{m,j,o} - S^*_{m,j,o} \cdot A_{j,o} - \Omega \cdot (x_{m,1,j,o} + x_{m',r',j,o-1}) + 2\Omega \geq \hat{c}_{m',r'}; \ \ \forall (m,m,j,o',r') |\ o > 1 \tag{5.11}$$

$$\hat{c}_{m,r} - B_{m,j,o} - S_{m,j,o,j',o'} \cdot A_{j,o} - \Omega \cdot (x_{m,r,j,o} + x_{m',r',j,o-1} + x_{m,r-1,j',o'}) + 3\Omega \geq \hat{c}_{m',r'};$$

$$\forall(m, r, m', r', j, o, j', o')|(r > 1)\,\&\,(o > 1)\,\&\,((j, o) \neq (j', o')) \tag{5.12}$$

$$x_{m,r,j,o} \leq P_{m,j,o};\ \ \forall(m, r, j, o) \tag{5.13}$$

$$\sum_{m=1}^{M}\sum_{r=1}^{R_m} x_{m,r,j,o} + U_{j,o} = 1;\ \ \forall(j, o) \tag{5.14}$$

$$\sum_{j=1}^{J}\sum_{o=1}^{O_j} x_{m,r,j,o} = z_{m,r};\ \ \forall(m, r) \tag{5.15}$$

$$z_{m,r+1} \leq z_{m,r};\ \ \forall(m, r) \tag{5.16}$$

$$x_{m,r',j,o'} \leq 1 - x_{m,r,j,o};\ \ \forall(m, r, r', j, o, o')|(o' > o)\,\&\,(r' < r) \tag{5.17}$$

$$x_{m,r',j,o'} \leq 1 - x_{m,r,j,o};\ \ \forall(m, r, r', j, o, o')|(o' < o)\,\&\,(r' > r) \tag{5.18}$$

$$x_{m,r,j,o} \textit{ and } z_{m,r} \textit{ are binary} \tag{5.19}$$

Outsourcing Equations:

$$c_{j,1} \geq R_{j,1} + K_{j,j'} \cdot c_{j',o'};\ \ \forall(j, j', o') \mid U_{j,1} = 1\ \&\ o' = O_{j'} \tag{5.20}$$

$$c_{j,o} \geq c_{j,o-1} + R_{j,o};\ \ \forall(j, o) \mid U_{j,o} = 1\ \&\ o \geq 1 \tag{5.21}$$

Subassembly Equations:

$$\hat{c}_{m,1} - B_{m,j,1} - S^*_{m,j,1} \cdot A_{j,1} - \Omega \cdot x_{m,1,j,1} + \Omega \geq c_{j',o'};$$
$$\forall(m, j, o', j') \mid K_{j,j'} = 1\ \&\ U_{j,1} = 0\ \&\ o' = O_{j'} \tag{5.22}$$

$$\hat{c}_{m,r} - B_{m,j,1} - S_{m,j,1,j'',o''} \cdot A_{j,1} - \Omega \cdot (x_{m,r,j,1} + x_{m,r-1,j'',o''}) + 2\Omega$$

$$\geq c_{j',o'};$$

$$\forall (m, r, j, j', o', j'', o'') \mid K_{j,j'} = 1 \;\&\; U_{j,1} = 0 \;\&\; o' = O_{j'} \;\&\; (1, j) \neq (o'', j'') \;\&\; r > 1 \tag{5.23}$$

Many of the constraint functions of this model are similar to the MILP model of the original Two-Stage GA presented in section 3.2.1. However, in this new model, in addition to having new features and new objective functions, we have transformed the fundamental concept of completion time of each step of the production from $c_{j,o,m}$ to $c_{j,o}$ that is independent of the machine number. This significantly simplified the required outsourcing equations and reduced the dimension of the other equations, speeding up the lengthy solution time. Since we provided a description of the original equations in section 3.2.1, here we only describe the differences.

Equation (5.6) is similar to the original equation for calculating the makespan with the only difference of $c_{j,o}$ instead of $c_{j,o,m}$. Similarly, Eqs. (5.7) and (5.8) calculates the completion time of each step of the production process if it is in-house while Eqs. (5.20) and (5.21) are for outsourced processes. Eqs. (5.9) and (5.10) are almost the same as the original model, except in this model, we do not consider the machine release date and batch quantity for simplicity. It is important to note that in the calculation of the machine run time, based on its previous run, setups are considered regardless of their detached and attached natures since they both will take time from the machines. While Eqs. (5.11) and (5.12), which calculate the completion time of each run of the machine based on the previous operation of the job, only consider attached setups ($A_{j,o}$ =1). Similar to the original MILP model, Eqs. (5.13) and (5.14) assure operation o of job j is assigned to a single capable machine of m ($P_{m,j,o} = 1$) with the additional condition that only if it is not outsourced ($U_{j,o} = 0$). Equations (5.15) to (5.18) have not changed from the original model since they only assure the feasibility of the solution in terms of machine assignment and operation sequencing.

Eqs. (5.20) and (5.21) are handling the outsourcing processes, so they did not

exist in the original model. Equation (5.20) addresses the situations in which the parent job's first operation is outsourced. The outsourced operation can be released as early as time zero or as soon as the subassembly job is completed ($K_{j,j'} = 1$). Therefore, the completion time is greater or equal to the outsourcing lead time. It can be greater (instead of equal) to accommodate the intentional delay (late start) for minimizing earliness/tardiness objective. As per equation (5.21), the completion time of any not-first outsourced operation is equal to or greater than the completion time of the previous operation plus outsourcing lead time. We will show in the prototype problem (Figure 5.10) how this equation works even when there are two consecutive outsourced operations (i.e., forming a metal plate by one supplier and then powder coating it by another before the inhouse mechanical assembly starts). Eqs. (5.22) and (5.23) are subassembly equations that are also not in the original MILP model since no job can be started until all its immediate subassemblies have been completely processed. These equations,Eqs. (5.22) and (5.23), assure that the first operation of a parent job only starts after the completion of the last operation of its subassembly that can be either in-house or outsourced.

Linearizing the Model:

As we discussed, the proposed model is MINLP because of the "absolute value" function in the earliness/tardiness objective function (equation 5.3). In order to transform it into a linear equation, we define t_j as the absolute value of earliness/tardiness and re-write the equation (5.3) as equation (5.24) and add 3 more constraints (equations 5.25 to 5.27). Since t_j is a non-negative number and is under a minimization objective function, it will be equal to $|c_{O_j,j} - D_j|$.

$$Z_3 = \sum_{j=1}^{J} t_j; \quad \forall(j)|\ E_j = 1 \tag{5.24}$$

s.t:

$$c_{j,o} - D_j \leq t_j;\ \forall(j,o)|\ E_j = 1\ \&\ o = O_j \tag{5.25}$$

$$- c_{j,o} + D_j \leq t_j;\ \forall(j,o)|\ E_j = 1\ \&\ o = O_j \tag{5.26}$$

$$t_j \geq 0;\ \forall(j)|\ E_j = 1 \tag{5.27}$$

5.3. Genetic Algorithm

Chapter 1 introduced the Genetic Algorithm for the FJSP, and the previous studies in this field are reviewed in Chapter 2. Then in Chapter 3, we described the developed Two-Stage GA and proved that it outperforms the regular approach of GA for FJSP. So in this section, we only focus on describing the modified Two-Stage GA to handle E/T FJSP in the presence of subassembly and outsourcing without repeating the introduction or literature review.

5.3.1. Prototype Problem

To describe the modified Two-Stage GA, we have created a prototype FJSP with 6 jobs and 4 machines. Two of these jobs (job 1 and job 3) are final assemblies which half of them (job 1) have due dates and the other half (job 3) should finish ASAP. Each final assembly job has 1 to 3 sub-assemblies in 1 to 2 levels. Figure 5.1 shows the Bill of Material structure for this prototype problem. Each job has 2 to 4 operations that about 30% of them are outsourced and the remaining operations have 2 to 3 capable machines with different processing times. Each in-house operation has a sequence dependent setup that can be attached or detached. In this section, we use this prototype problem to explain the 2SGA model and its operators, and then we solve it in section 5.4.1 to explain how the model works.

Tables 5.1 and 5.2 provide all the data for this problem. Table 5.1 includes no. of operations of each job, outsourcing lead time for outsourced operations, the nature of the setup, and the processing time for in-house operations. Table 5.2 includes sequence dependent setup time, including the setup time if the operation is the first operation on

the machine ($S^*_{m,j,o}$) and $S_{m,j,o,j',o'}$ where j', o' are the previous operation on the machine m. We should note that Table 5.2 considers the precedence constraint and subassembly relationship and only shows the sequence dependent setup time for all feasible operation sequences. For example, there is no setup time for the cases where the future operation of a specific job is considered as the previous operation on the machine, like for $(j, o) = (2, 2)$ in the $S_{m,j,o,j',o'}$ we cannot see any $(2, 3)$ or $(2, 4)$ as (j', o'). Similarly, we cannot see any operations of job1 as (j', o') of job2 in the $S_{m,j,o,j',o'}$, since job2 is the child of job1 and needs to be processed completely before job1 can start.

Table 5.1: Data for Jobs for Prototype Problem

job	*Immediate*	*Operation*	*Outsourcing*	*Attached or*	(*Machine no.*, *Processing Time*)		
no.	*Child*	*no.*	*Lead Time*	*Detached Setup*	i	ii	iii
1	job2	1	–	1	(1,60)	(3,60)	(4,90)
		2	–	1	(1,13)	(3,6)	(4,5)
		3	63	–	–	–	
		4	–	0	(1,12)	(2,6)	(3,5)
2	none	1	42	–	–	–	
		2	–	1	(1,50)	(2,35)	(3,60)
		3	–	1	(3,5)	(4,7)	
		4	–	1	(1,30)	(4,50)	
3	jobs 4&6	1	–	0	(1,52)	(3,28)	(4,20)
		2	–	1	(2,33)	(3,42)	(4,30)
		3	–	1	(2,22)	(3,28)	
4	job5	1	195	–	–	–	
		2	–	1	(3,15)	(4,13)	
		3	–	1	(1,11)	(2,9)	(4,8)
5	none	1	–	1	(1,45)	(3,39)	
		2	108	–	–	–	
6	none	1	–	1	(1,44)	(3,20)	(4,20)
		2	126	–	–	–	
		3	42	–	–	–	

job 1 and *job* 3 are the final assemblies (shown in 5.1)
job 1 has due date = 800 but *job* 3 has no due date

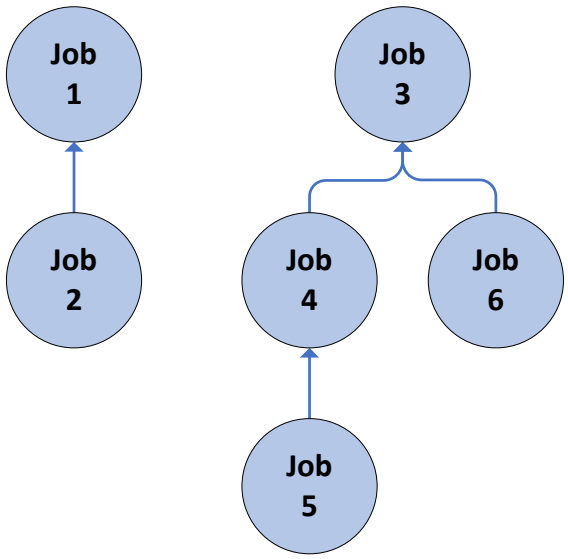


Figure 5.1: Bill of Material (BOM) for Prototype Problem

5.3.2. Solution Encoding and Decoding

The Two-Stage GA has a greedy nature and finds the machine that can finish the job fastest. So it can easily handle minimizing the makespan, total completion time, or many other objective functions we addressed in Section 4. However, as the numerical examples of section 5.4 prove, the Two-Stage GA is not able to minimize the earliness/lateness objective function for the jobs with due dates. To add this critical feature, the chromosome encoding of the Two-Stage GA had to change to make it possible for the algorithm to intentionally delay the release of the jobs.

The modified 2SGA model for E/T FJSP, described in Section 5.2, has two other new features of "Subassembly Requirement " and "Outsourcing Option". The presence of the subassembly feature requires the GA to recognize the relationship between multiple jobs in addition to the relationship between the operations of a single job. These features are embedded within the GA's evaluating and chromosome feasibility mechanisms, and hence there was no need to modify the 2SGA solution encoding. The outsourcing feature can be coded so that the outsourced operation is assigned to an imaginary machine with no capacity limitation that can handle several outsourced operations

concurrently. In this section, we describe how the original Two-Stage GA, explained in Chapter 3, is modified to accommodate the E/T FJSP in the presence of subassembly requirements and outsourcing options.

Section 3.3.1 reviewed the common solution encodings widely used in the literature to solve FJSPs, so here we only introduce the chromosome encoding of this GA model. Figure 5.2 shows the solution encoding using the best-found chromosome for Prototype Problem. As is shown, the chromosome encoding has two segments. The Left-Hand Side (LHS) chromosome handles the "Intentional Delay" problem. The Right-Hand Side (RHS) chromosome follows the original chromosome encoding of the Two-Stage GA and addresses sequencing and assignment problems. In order to lessen the WIP level, we allow the GA to only add the intentional delay to the very first operation of each job instead of holding them in between the processes. So the number of genes in the LHS could be reduced to the number of jobs (instead of the total number of operations for RHS). Each LHS gene is composed of two parts of "Is-Delayed" (or α) and "Delay-Time" (or β). "Is-Delayed" is the first element of the gene and is a boolean parameter, and the second part, "Delay-Time", indicates the delay duration. If "Is-Delayed" takes a 0 value, the corresponding job will not be delayed and will be released to the job floor as soon as possible. However, if it takes the value of 1, the corresponding job will be delayed, equal to the "Delay-Time". So the total delay of each job can be calculated by multiplying the two elements ($\alpha \times \beta$) of the corresponding LHS gene. The RHS is the original solution encoding of the 2SGA described in Section 3.3.1 in which the number of genes equals the total number of operations. In the first stage, the GA only determines the sequence of operation for machine assignment but without any machine assignment in the solution encoding, as shown in Figure 5.2-b. However, in the second stage of the 2SGA, both sequencing and assignment problems are addressed by GA search, hence the gene is a 3-tuple form of $[j, o, m]$.

General Structure of Solution Encoding	
Left-Hand Side Segment	**Right-Hand Side Segment**
Encodes Intentional Delay of the Jobs	Encodes the Assignment and Sequencing of the Jobs

(a) General structure of the solution representation

Solution Encoding for the First Stage of the Search

LHS-Segment (Is Delayed/Delay Time)

1		2		3		4		5		6	
Job_1		Job_2		Job_3		Job_4		Job_5		Job_6	
1	410	1	1	0	85	0	73	0	2	1	123

RHS-Segment (Operation Sequencing)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
j, o																		
5,1	2,1	5,2	6,1	6,2	4,1	4,2	4,3	2,2	6,3	2,3	2,4	1,1	1,2	3,1	3,2	1,3	1,4	3,3

(b) Solution Encoding for the First Stage of the Search which RHS Dictates the Sequecting of the Operations

Solution Encoding for the Second Stage of the Search

LHS-Segment (Is Delayed/Delay Time)

1		2		3		4		5		6	
Job_1		Job_2		Job_3		Job_4		Job_5		Job_6	
1	410	1	1	0	85	0	73	0	2	1	123

RHS-Segment (Operation Sequencing / Machine Assignment)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
j, o, m																		
5,1,3	2,1,5	5,2,5	6,1,4	6,2,5	4,1,5	4,2,4	4,3,2	2,2,1	6,3,5	2,3,3	2,4,1	1,1,1	1,2,4	3,1,3	3,2,2	1,3,5	1,4,3	3,3,2

(c) Solution Encoding for the Second Stage of the Search which RHS Determines both Sequecing and Assignment

Figure 5.2: Solution representation

5.3.3. Genetic Operators

Genetic Algorithms use "Genetic Operators" to evolve generations to improve the fitness of chromosomes. The main groups of operators are "selection operators", "crossover operators", and "mutation operators". Section 4.3.5 provides a detailed description of different types of operators. So this section only focuses on genetic operators specifically designed to handle "Intentional Delay", "Subassembly" and "Outsourcing" for the Two-Stage GA model. For the selection operator, we follow the findings of section 4.4.3 that suggest tournament selection performs better than other common types for the Two-Stage GA model. The tournament selection randomly picks a number of chromosomes (with replacement) and places the fittest one in the mating pool. It repeats this process to fill the mating pool to equal the population size. In tournament selection, even the chromosomes with not great fitness value have the chance to be selected. Therefore the

mating pool has a better diversity that reduces the chance of the GA falling in the local optimum.

After forming the mating pool, "crossover operators" will be applied to create the offspring. Similar to the nature, there are two parents who create two offspring that inherit their genes from both parents. Although, in the GA, the parents are selected and paired randomly from a mating pool. As explained in section 3.3.2, this Two-Stage GA model has two crossover operators, which are the modified versions of the Single Point Crossover Operator (SCO) and Assignment Crossover Operator (ACO). These two crossover operators are modified to accommodate the chromosome's LHS segment that handles the model's intentional delay.

As shown in Figure 5.3, the modified Single Point Crossover Operator (modified SCO) arbitrarily selects a point in the parents' chromosome. Then it copies all the genes on the left side of the crossover point from one parent, completes the remaining genes in the same sequence as the other parent, and creates two offspring. As seen in Figure 5.3, this operator is applied at both stages with a probability of p_1. On the other hand, the modified Assignment Crossover Operator (modified ACO) has different forms of implementations in stage-1 and stage-2 of the Two-Stage GA. In the first stage, it only modifies the LHS since there is no machine assignment in the RHS of chromosome encoding. As it is shown in Figure 5.4, each child takes the "is-delayed" (or α) element of each gene from one parent and the "delay-time" (or β) from the other parent. However, the RHS of the child comes from the original parent intact. In stage-2, in addition to the LHS changes of stage-1, parents exchange the machine assignment to create two offspring. The ACO is applied with a probability of p_2, too.

After the creation of the new generation of offspring, they randomly go through mutation where "mutation operators" alter their genes. The mutation operators are applied with a small probability and help improve the GA population's diversity. Due to the complexity of the chromosome encoding of Two-Stage GA for FJSP in the presence of subassembly, outsourcing, and intentional delay, five mutation operators have been designed. Two mutation operators are applied on the RHS of the chromosome, and the

other three operators alter LHS genes.

The Assignment Altering Mutation operator is applied by a small probability of p_3. It arbitrarily selects one gene from the RHS and changes its machine assignment only if it is not outsourced. Figure 5.5 shows how this operator modifies some genes. Since this operator alters the machine assignment, it is only applied in stage-2 of the Two-Stage GA. The other RHS mutation operator is Operation Swapping Mutation (OSM), shown in Figure 5.6. OSM selects a chromosome by the probability of p_4 and then switches two adjacent genes of the RHS only if they are not 1) from the same job and 2) from the jobs with a parent-child relationship. As is shown in Figure 5.6, the gene $(6, 1, 4)$ is selected. However, the OSM operator does not switch its position since the adjacent gene, $(6, 2, 5)$, is the subsequent operation of the same job, and switching their position will make the chromosome infeasible due to precedence constraints. Similarly, gene $(2, 4, 1)$ that is adjacent to $(1, 1, 1)$ cannot switch their positions either since job 2 is a subassembly of job 1. Nevertheless, OSM switches two other selected genes $(4, 3, 2)$ and $(3, 2, 2)$ whose adjacent genes are not from the same job or have a subassembly relationship.

The other three mutation operators are applied on the chromosome's LHS or intentional delay segment. The Delay Change Mutation operator changes a radome gene's delay-time (or β) from 50% to 150% according to equation Eq. (5.28). In this equation, rand$(0.5, 1.5)$ is a function that returns a random number in the interval [0.5,1.5] following a uniform distribution. This mutation operator is being applied by a small probability of p_5, shown in Figure 5.7.

$$\text{delay-time} = \text{delay-time} \times \quad \text{rand}(0.5, 1.5) \tag{5.28}$$

The other mutation operator is the Delayed Flip Mutation operator, which inverts the "is-delayed" (or α) binary element of a randomly selected gene as illustrated in Figure 5.8. Inverting "is-delayed" (or α) makes a delayed job (is-delayed equal to 1) to be released immediately (is-delayed equal to 0) and vice versa. The last type of LHS mutation operator is the Delay Swap Mutation, shown in Figure 5.9, which works similarly to OSM and switches a random gene of the LHS segment with its adjacent gene. As illustrated in Figure 5.2, each LHS gene simply refers to a job number corresponding

to the location of the gene in the LHS chromosome. So the Delay Swap Mutation operator, unlike OSM, can be applied to any gene regardless of its relationship with its adjacent gene. Delayed Flip Mutation and Delay Swap Mutation operators are applied by small probabilities of p_6 and p_7, respectively. It should be mentioned that Figures 5.5 to 5.9 try to fully illustrate how mutation operators work by demonstrating altering several genes, while in the actual Two-Stage GA model, mutation operators alter only one arbitrary gene of each randomly selected chromosome.

To summarize, below is the list of the GA operators for the Two-Stage GA for E/T FJSP with the presence of subassembly and outsourcing:

(a) Modified Single Point Crossover Operator (modified SCO),

(b) Modified Assignment Crossover Operator (modified ACO) for stage 1,

(c) Modified Assignment Crossover Operator (modified ACO) for stage 2,

(d) Assignment Altering Mutation,

(e) Operations Swapping Mutation (OSM),

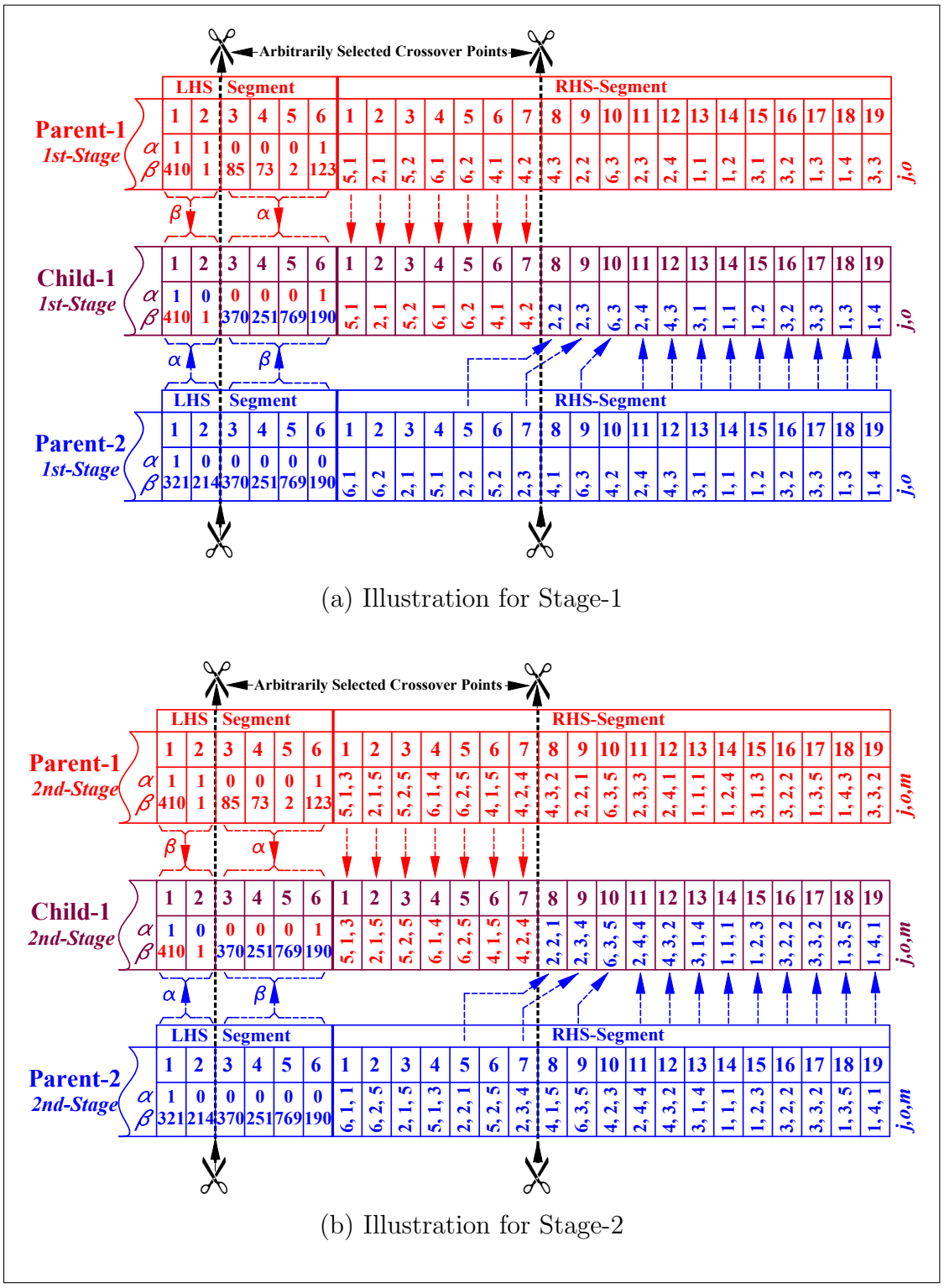
(f) Delay Change Mutation operator,

(g) Delayed Flip Mutation,

(h) Delay Swap Mutation.

Table 5.2: Sequence Dependent Setup Time Data for Prototype Problem

			Setup Time $S^*_{m,j,o}$ and $S_{m,j,o,j',o'}$	
j	o	m	$(S^*_{m,j,o})$	$\cdots(j', o', S_{m,j,o,j',o'})$ $\cdots$
1	1	1	(10)	(2,2,20)(2,4,36)(3,1,20)(4,3,24)(5,1,32)(6,1,40)
		3	(15)	(2,2,40)(2,3,40)(3,1,20)(3,2,20)(3,3,28)(4,2,24)(5,1,24)(6,1,32)
		4	(20)	(2,3,36)(2,4,36)(3,1,40)(3,2,32)(4,2,20)(4,3,32)(6,1,40)
	2	1	(15)	(1,4,24)(2,2,20)(2,3,32)(2,4,24)(3,1,36)(4,3,40)(5,1,36)(6,1,36)
		3	(10)	(1,4,27)(2,2,36)(2,3,28)(2,4,20)(3,1,28)(3,2,24)(3,3,20)(4,2,32)(5,1,32)(6,1,20)
		4	(10)	(3,1,24)(3,2,20)(4,2,20)(4,3,28)(6,1,40)
	3	Outsourced		
	4	1	(10)	(2,2,40)(2,4,20)(3,1,32)(4,3,28)(5,1,20)(6,1,40)
		2	(10)	(2,2,36)(3,2,28)(3,3,24)(4,3,32)
		3	(10)	(2,2,28)(2,3,32)(3,1,24)(3,2,40)(3,3,28)(4,2,28)(5,1,36)(6,1,20)
2	1	Outsourced		
	2	1	(10)	(3,1,20)(4,3,28)(5,1,24)(6,1,32)
		2	(15)	(3,2,40)(3,3,36)(4,3,24)
		3	(20)	(3,1,20)(3,2,32)(3,3,24)(4,2,28)(5,1,24)(6,1,40)
	3	3	(10)	(2,2,18)(3,1,32)(3,2,28)(3,3,20)(4,2,36)(5,1,36)(6,1,28)
		4	(15)	(3,1,36)(3,2,20)(4,2,20)(4,3,36)(6,1,40)
	4	1	(15)	(2,2,27)(3,1,24)(4,3,28)(5,1,32)(6,1,36)
		4	(10)	(2,3,18)(3,1,36)(3,2,20)(4,2,24)(4,3,28)(6,1,32
3	1	1	(15)	(1,1,32)(1,2,36)(1,4,24)(2,2,24(2,4,24(4,3,20)(5,1,28)(6,1,20
		3	(15)	(1,1,28)(1,2,20)(1,4,20)(2,2,40)(2,3,36)(4,2,40)(5,1,40)(6,1,36)
		4	(15)	(1,1,28)(1,2,32)(2,3,24)(2,4,36)(4,2,28)(4,3,36) (6,1,24)
	2	2	(20)	(1,4,28)(2,2,36)(4,3,20)
		3	(15)	(1,1,36)(1,2,32)(1,4,24)(2,2,24)(2,3,24)(3,1,15)(4,2,24)(5,1,28)(6,1,32)
		4	(10)	(1,1,28)(1,2,40)(2,3,36)(2,4,32)(3,1,24)(4,2,32)(4,3,32)(6,1,40)
	3	2	(10)	(1,4,40)(2,2,28)(3,2,21)(4,3,28)
		3	(15)	(1,1,40)(1,2,40)(1,4,40)(2,2,28)(2,3,20)(3,1,27)(3,2,15)(4,2,20)(5,1,40)(6,1,20)
4	1	Outsourced		
	2	3	(10)	(1,1,20)(1,2,40)(1,4,36)(2,2,24)(2,3,36)(5,1,28)(6,1,24)
		4	(20)	(1,1,20)(1,2,36)(2,3,36)(2,4,40)(6,1,20)
	3	1	(10)	(1,1,20)(1,2,20)(1,4,20)(2,2,36)(2,4,24)(5,1,40)(6,1,20)
		2	(10)	(1,4,24)(2,2,40)
		4	(20)	(1,1,20)(1,2,36)(2,3,28)(2,4,36)(4,2,18)(6,1,32)
5	1	1	(20)	(1,1,20)(1,2,40)(1,4,24)(2,2,28)(2,4,20)(6,1,20)
		3	(15)	(1,1,32)(1,2,24)(1,4,20)(2,2,24)(2,3,32)(6,1,24)
	2	Outsourced		
6	1	1	(15)	(1,1,40)(1,2,24)(1,4,24)(2,2,32)(2,4,28)(4,3,32)(5,1,24)
		3	(10)	(1,1,24)(1,2,40)(1,4,36)(2,2,32)(2,3,28)(4,2,40)(5,1,24)
		4	(10)	(1,1,40)(1,2,32)(2,3,28)(2,4,28)(4,2,36)(4,3,20)
	2	Outsourced		
	3	Outsourced		



(a) Illustration for Stage-1

(b) Illustration for Stage-2

Figure 5.3: Illustration of Modified Single Point Crossover Operator for the Two-Stage GA for E/T FJSP

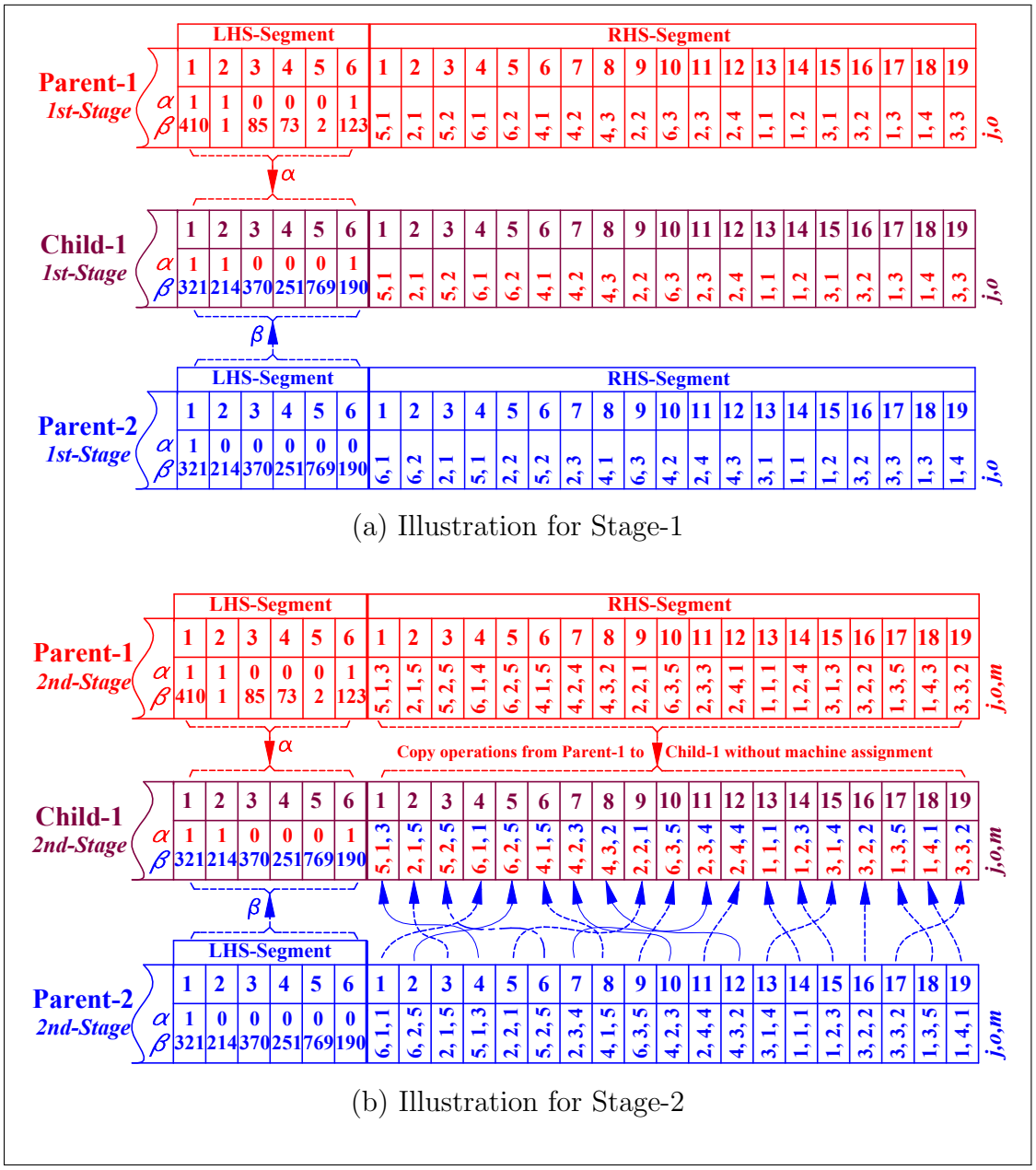


Figure 5.4: Illustration of Modified Assignment Crossover Operator for the Two-Stage GA for E/T FJSP

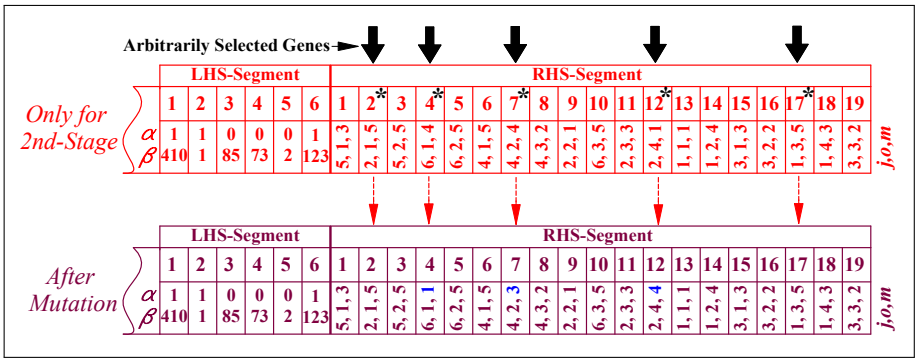


Figure 5.5: Illustration of Assignment Altering Mutation Operator for the Two-Stage GA for E/T FJSP

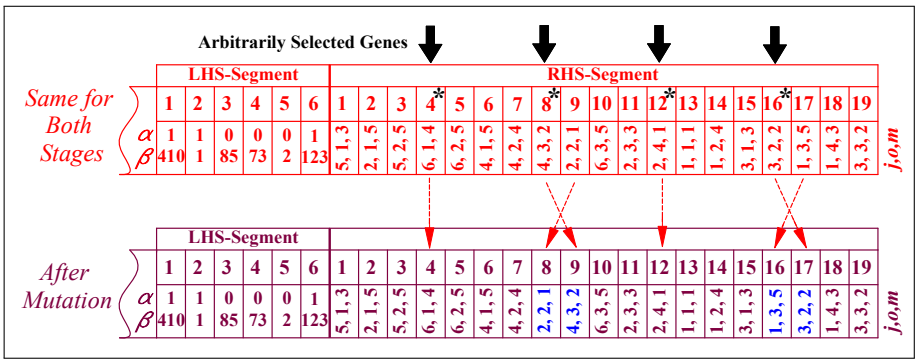


Figure 5.6: Illustration of Operation Swapping Mutation Operator for the Two-Stage GA for E/T FJSP

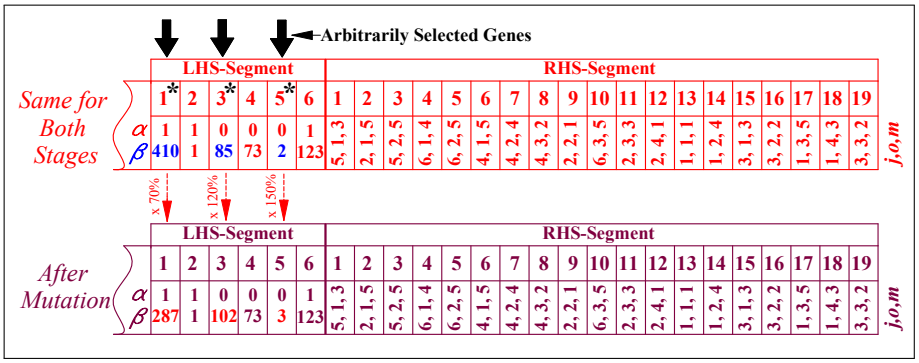


Figure 5.7: Illustration of Delay Change Mutation Operator for the Two-Stage GA for E/T FJSP

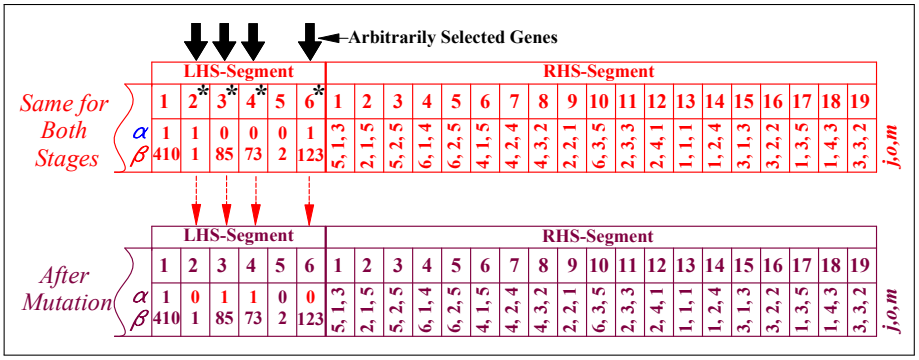


Figure 5.8: Illustration of Delayed Flip Mutation Operator for the Two-Stage GA for E/T FJSP

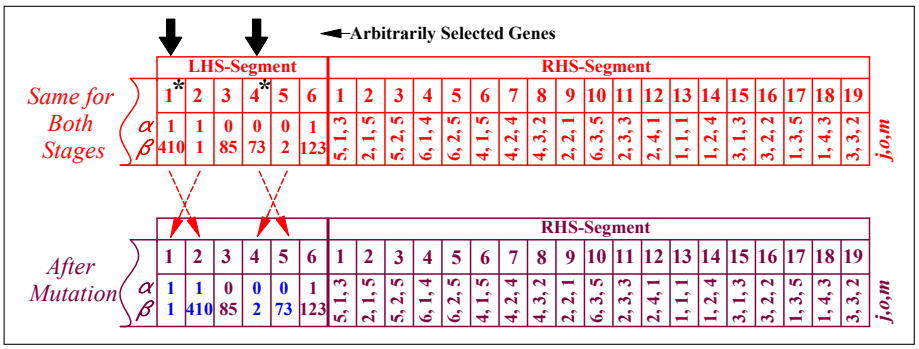


Figure 5.9: Illustration of Delay Swap Mutation Operator for the Two-Stage GA for E/T FJSP

5.4. Numerical Studies

In this section, we illustrate the performance of the modified Two-Stage GA for E/T FJSP in the presence of sub-assembly requirements and outsourcing options. First, we solve the 6 *job by* 4 *machine* prototype problem that was described in Section 5.3.1 and its data was presented in Tables 5.1 and 5.2. This example shows how the model considers all the constraints and conditions of the example, including the machine capability, detached or attached setup time, outsourcing, sub-assembly relationship, intentional delay to meet the due dates, etc.

As reviewed in Section 2.2.3, E/T scheduling objectives of meeting the planned completion dates (or due dates) to minimize the earliness and tardiness costs are very popular in both literature and the real world. However, we suspect the original Two-Stage GA that was introduced in Chapter 3 is not able to minimize earliness and tardiness due to its greedy nature that is very efficient in finding machines that can complete the jobs as soon as possible to minimize makespan and the total completion time. In this chapter, we will test this hypothesis and show that the original model, in fact, cannot minimize earliness and tardiness by solving a set of example problems. Then we illustrate that adding the intentional delay to the model will solve this issue and enable the model to minimize earliness and tardiness objectives effectively.

Another feature of this model is eliminating the unrealistic assumption of the classic FJSP that all the operations are being processed in-house. By solving the prototype problem, we illustrated how the model handles the outsourcing. We also wanted to demonstrate how the model can be used as a decision-making tool to check different outsourcing strategies and what-if scenarios. For example, if the company decides to outsource more operations to free up internal resources considering that each machine in FJSP can handle more than one operation. Also, what if the company negotiates with its vendors to expedite and reduce the lead time. In this section, we also show how the model can handle the scheduling of more complex products with multilevel BOM by solving a 100j x 50m problem that has 10 final assemblies that each has sub-assemblies

up to 10 levels.

5.4.1. Prototype Problem Solution

To illustrate how the proposed model works in solving E/T FJSP, the prototype problem introduced in section 5.3.1 is solved. As it can be seen in tables 5.1 and 5.2, this problem has 4 machines and 6 jobs with a total of 19 operations. Six of these operations are outsourced, and the remaining have either 2 or 3 alternative machines. The in-house operations have sequence dependent setup in which two are detached and the rest are attached. As shown in figure 5.1, job 1 and job 3 are final assemblies with 1 and 3 subassemblies, respectively. Only job 1 has a due date of 800.

This problem was solved with the developed Two-Stage GA, and the best-found solution encoding is shown in figure 5.2-c. The Gantt chart of this solution encoding is demonstrated in figure 5.10. Also, table 5.3 reports the scheduling results sorted by jobs, and table 5.4 lists the same result but sorted by machines. The most important observation is that the model achieved a good and feasible solution that follows all the constraints of the prototype FJSP. As per solution encoding shown in figure 5.2-c, jobs 5, 2, and 6 are the first jobs that are being processed concurrently. Job 5 starts at time 0 and on machine no.3 and then goes right away for outsourcing. Job 4 starts at 162 right after completion of the job 5, which is its immediate subassembly.

Job 3 has 2 subassemblies which are job 4 (finishing at 409) and job 6 (finishing at 321), so it can start only as early as 409 when both of its subassemblies are completed. It is interesting to see that the setup of operation no.1 of job 3 is detached and starts exactly enough ahead of the arrival of the job and is completed right at 409 when job 3 can start. Job 3 has no due date and needs to be completed ASAP, and as we can see in its longest subassembly path ($5 \rightarrow 4 \rightarrow 3$), no operation is delayed. However, in the other path ($6 \rightarrow 3$), which has some slack (float), we can see that job 6 has an intentional delay of 123 time units, and it still finishes before job 4. Job 6 has another interesting feature of two consequent outsourcing processes that the model scheduled them right after one another as it should.

On the other hand, job 1 has a due date of 800, and we can easily conclude that it is a pretty relaxed due date by only looking at the solution table. So the schedule should dictate some intentional delay to avoid earliness issues. As per the LHS-Segment of the solution encoding shown in figure 5.2-c, both jobs 1 and 2 have intentional delays, although not equal. Job 2 starts at 1 with an outsourced operation, and job 1 starts after completion of job 2 with a delay of 410 to finish right on time at 800. Operation 3 of job 1 is outsourced and completed at 795. However, the subsequent operation, operation 4, has a detached setup scheduled at 771 to finish at 795 when the job arrives at the machine. This shows that the model has no issue scheduling the detached setup even after outsourced operations.

Table 5.4 shows the same schedule but sorted per machine, which proves the solution also respects the machine-related constraints. Each operation has been assigned to only one round of any machine, respecting the precedence constraints. Also, we can see how the two detached setups $(j3, o1)$ and $(j1, o4)$, which both are scheduled on machine 3, were able to start before the arrival of the job since the machine was idle at that time. The most important observation of this table is how the model schedules several outsourced operations concurrently. Unlike in-house machines, the outsourcing process has been set to have no limited capacity, but it still respects the precedence constraints. The heuristic and metaheuristic algorithms (including 2SGA) cannot guarantee reaching the optimum solution. However, the table 5.5 shows the 2SGA has reached the best possible objective function of 0 for Total Earliness & Tardiness. Also, the Gantt chart 5.10 shows a pretty fast-tracked schedule for Job 3 that is subjected to minimization of the Makespan and Total Completion Time objective functions.

5.4.2. E/T Minimization Performance

As mentioned before, the main idea of developing the modified Two-Stage GA was solving its pragmatic issue of not being able to handle minimizing the earliness/tardiness by adding the intentional delay. In this section, we like to test the below hypothesizes:

(a) The original 2SGA can minimize makespan and total completion time,

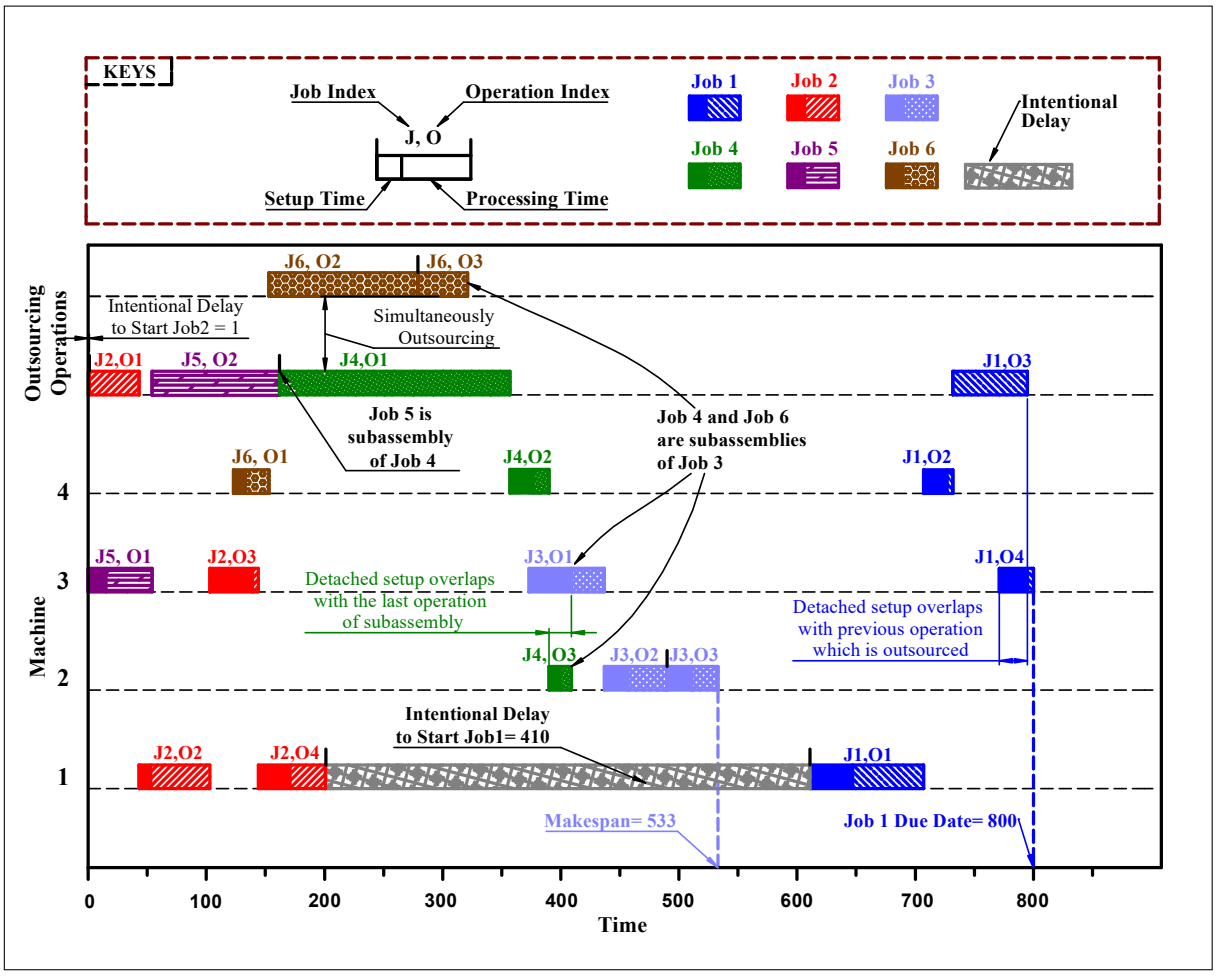


Figure 5.10: Gantt Chart of the schedule corresponding to the best found solution for the prototype problem

(b) The original 2SGA is not able to minimize earliness and tardiness,

(c) Adding intentional delay keeps the ability to minimize the makespan and total completion time,

(d) Adding intentional delay enables the 2SGA model to minimize earliness and tardiness.

In order to do so, we developed a set of 7 examples and solved them under two conditions of with and without intentional delay. As shown in table 5.6, the problem sizes vary from $20J\ x\ 10M$ to $50J\ x\ 30M$, and each has 8 to 15 final assembly jobs. As described in the mathematical model in Section 5.2, the jobs without due dates have

Table 5.3: Operation scheduling for Prototype Problem reported by jobs

j	o	m	r	ID	SB	SE/PB/OS	PE/OE
1	1	1	3	410	611	647	707
	2	4	3		707	727	732
	3	O/S				732	795
	4	3	4		771	795	800
2	1	O/S		1		1	43
	2	1	1		43	53	103
	3	3	2		103	139	144
	4	1	2		144	171	201
3	1	3	3	0	373	409	437
	2	2	2		437	457	490
	3	2	3		490	511	533
4	1	O/S		0		162	357
	2	4	2		357	377	390
	3	2	1		390	400	409
5	1	3	1	0	0	15	54
	2	O/S				54	162
6	1	4	1	123	123	133	153
	2	O/S				153	279
	3	O/S				279	321

ID = Intentional Delay; SB = Setup Begins; SE = Setup Ends; PB = Processing Begins; OS = Outsourcing starts; PE = Processing Ends; OE = Outsourcing Ends; O/S = Outsourcing;

two objective functions: makespan and the total completion time. In contrast, the other jobs with due dates are subjected to the minimization of total earliness and tardiness. To have a better comparison, we separated the earliness and tardiness values. As it was expected and table 5.6 shows, the Two-Stage GA cannot minimize the earliness without intentional delay. After solving the 7 sample problems with no intentional delay, the final solutions had an average of 603 time units as the total earliness value. In contrast, the Two-Stage GA with intentional delay ended with 0 earliness for every sample problem except one with earliness of only 1. This can be easily explained if we note that the

Table 5.4: Operation scheduling for Prototype Problem reported by machines

m	r	j	o	SB	SE/PB	PE
1	1	2	2	43	53	103
	2	2	4	144	171	201
	3	1	1	611	647	707
2	1	4	3	390	400	409
	2	3	2	437	457	490
	3	3	3	490	511	533
3	1	5	1	0	15	54
	2	2	3	103	139	144
	3	3	1	373	409	437
	4	1	4	771	795	800
4	1	6	1	123	133	153
	2	4	2	357	377	390
	3	1	2	707	727	732
Outsourcing		j	o	DP	LT	RE
		2	1	1	42	43
		5	2	54	108	162
		6	2	153	126	279
		4	1	162	195	357
		6	3	279	42	321
		1	3	732	63	795

SB = Setup Begins; SE = Setup Ends; PB = Processing Begins; PE = Processing Ends.
DP = Dispatching for outsourcing; LT = Lead Time; RE = Return from Outsourcing.

Table 5.5: Values of the objective function components for Prototype Problem

Objective Term	Calculation	Value
Makespan	$Z_1 = c_{max};\ \forall(j)\mid E_j = 0$	533
Total Completion Time	$Z_2 = \sum_{j=1}^{J} c_{j,O_j};\ \forall(j,o)\mid E_j = 0$	533
Total Earliness & Tardiness	$Z_3 = \sum_{j=1}^{J} \lvert c_{j,O_j} - D_j\rvert;\ \forall(j,o)\mid E_j = 1$	0
Weighted Aggregated Obj. Function	$Z = \sum_{k=1}^{3} W_k \cdot \Psi_k \cdot Z_k$ $W_k = 1/3 \& \Psi_k = \frac{Z_1^{Ini-max}}{Z_k^{Ini-max}}$	357.1

earliness value means there are some jobs with achievable or relaxed due dates, and the only thing the scheduling model needs to do is delay them to be released late enough to be finished on time. This is exactly how intentional delay works, as shown in Figure 5.2.

It is interesting to see that even total tardiness improves significantly with adding intentional delay. As we can see on average, total tardiness for the Two-Stage GA without intentional delay is 192 time units while it reduces to 99 time units by adding intentional delay, which stands for 48% improvement. Tardiness means there are jobs with unachievable due dates, so unlike earliness, simply delaying the jobs cannot improve tardiness. However, we should consider delaying some jobs that are not rushed (earliness issue) will free up resources for other jobs to finish earlier. For example, the tardiness of sample problem no. 5, which has the highest total earliness of 1307 among all sample problems, improved by 86% when the intentional delay was added. However, delaying no-rushed jobs cannot guarantee that rushed jobs will finish on time. For example, the tardiness of sample problem no.7, which has the second highest total earliness value, only improved 23% when the intentional delay was added. So it can be concluded that other factors, like how tight the deadlines are or how many resources these jobs share, impact the tardiness improvement results.

Another example is sample problem no. 6, which has the least earliness and is worsened by adding the intentional delay. This means delaying some jobs adds even more constraints on resources. In conclusion, we can say that intentional delay can generally improve total tardiness by prioritizing the jobs with a tight deadline over the jobs that can be delayed. It is obvious the total value of earliness and tardiness will be improved too. In these sample problems, adding intentional delay, on average, improved the total earliness and tardiness by 88%.

As was expected, the objective functions of the jobs with no due date mainly stayed the same. Adding the intentional delay improves makespan for some sample problems and worsens it for some, but on average, the changes are insignificant (almost 0%). Similar results are recorded for the total completion time, which is the total

completion time of all the jobs without due dates, vs. makespan, which is the longest completion time. As can be seen, the total completion time for most of the sample problems worsened slightly when the intentional delay was added, however, it is still an insignificant value of -1% on average. We should not be surprised that the overall fitness value (aggregated objective function) was improved in every sample problem due to a significant improvement in total earliness and tardiness. The aggregated objective function is improved between 6% to 33% for an average of 21%.

5.4.3. Outsourcing Analysis

As discussed in 2.2.5 and 5.1.3, outsourcing is a common practice in both literature and real-world job shop manufacturing that we added to this model. We showed how the modified Two-Stage GA model could schedule outsourced operations by solving the prototype problem shown in the Gantt chart 5.10. This section will demonstrate how the model can be used for decision-making or what-if scenario analysis. So we created 7 sample problems with the below characteristics:

(a) Number of machines vary from 5 to 25 for each problem,

(b) Number of jobs vary from 10 to 50 for each problem,

(c) Number of final assembly jobs vary from 3 to 10 for each problem,

(d) Half of the final assemblies of each problem have due dates, and the other half do not,

(e) Each job has 2 to 4 operations,

(f) Outsourcing processes is 30% of the total operations,

(g) the lead time of outsourced operations is on average 3 times longer than in-house processing time (outsourcing lead time random generator equation is as same as the processing time random generator equation only times three),

(h) Every operation has a sequence dependent setup that can be attached or detached.

Table 5.6: Performance comparison of Two-Stage GA with or without the Intentional Delay

Sample Problem	Problem Size	no. of Final Assemblies		Total Earliness		Total Tardiness		
no.	JxM	$W.DD^1$	$N.DD^2$	$N.ID^3$	$W.ID^4$	$N.ID^3$	$W.ID^4$	$Imp\%^5$
1	20x10	5	3	425	0	118	115	3%
2	30x15	3	5	781	0	1	1	0%
3	30x15	5	3	235	0	330	29	91%
4	40x20	6	4	185	0	67	1	99%
5	40x20	6	4	1307	0	160	23	86%
6	50x20	5	7	44	0	4	14	-250%
7	50x30	7	8	1245	1	661	508	23%
Average				603	0	192	99	**48%**

Sample Problem	Problem Size	Total Earliness/Tardiness			Makespan		
no.	JxM	$N.ID^3$	$W.ID^4$	$Imp\%^5$	$N.ID^3$	$W.ID^4$	$Imp\%^5$
1	20x10	543	115	79%	688	686	0%
2	30x15	782	1	100%	792	792	0%
3	30x15	565	29	95%	489	489	0%
4	40x20	252	1	100%	938	938	0%
5	40x20	1467	23	98%	967	1035	-7%
6	50x20	48	14	71%	1098	980	11%
7	50x30	1917	509	73%	714	759	-6%
Average		796	99	**88%**	812	811	**0%**

Sample Problem	Problem Size	Total Completion Time			Aggregated Obj. Function		
no.	JxM	$N.ID^3$	$W.ID^4$	$Imp\%^5$	$N.ID^3$	$W.ID^4$	$Imp\%^5$
1	20x10	1283	1346	-5%	1410	1171	17%
2	30x15	2817	2885	-2%	2596	1786	31%
3	30x15	1253	1255	0%	1000	897	10%
4	40x20	2979	3017	-1%	1980	1870	6%
5	40x20	2963	3103	-5%	2793	1873	33%
6	50x20	5674	5497	3%	4009	3702	8%
7	50x30	4220	4399	-4%	4539	3216	29%
Average		3027	3072	**-1%**	2618	2074	**21%**

1. W.DD = Final assembly jobs *With Due Date*;
2. N.DD = Final assembly jobs with *No Due Date*;
3. N.ID = Two-Stage GA with *No Intentional Delay*;
4. W.ID = Two-Stage GA *With Intentional Delay*;
5. Imp% = Performance *Improvement Percentage* ;

Each of these 7 sample problems was tested under 3 different what-if scenarios that are listed below:

1. The number of outsourced processes is doubled (another 30% of in-house operations are outsourced for a total of 60%) with a random lead time of about 3 longer than in-house processing time,

2. The lead time of the original outsourced processes (30% of total) is expedited to half of the original lead time,

3. Combination of Scenario1 and Scenario2 that will be 60% outsourced operations with half lead time (on average 1.5 times longer than in-house operations).

Table 5.7 shows the summary of this experiment, including the size of sample problems in terms of the number of jobs and machines (JxM) and three objective function values along with the aggregated objective function for the original problem and the three what-if scenarios. It also shows how much each objective function has improved by implementing scenario no.3 compared to the original problem. As it can be seen, scenario no.1 that is outsourcing more processes which will take much longer than in-house operations, did not improve any objective functions for any sample problem except for slight improvement of makespan and total completion time for $15x5$ that resulted in improving the aggregated objective function value for this problem. The worst drop is for the "Total E/T" of the $50x25$ problem that changes from 2 to 406.

The other 2 scenarios, scenario 2 and scenario 3, have improved all objective functions for every sample problem with no exception. Scenario no.2 is just expediting the currently outsourced processes, and scenario no.3 doubles the number of outsourcing and expedites them all. Although based on the average value reported in the table 5.7, scenario2 has a better performance than scenario3, it is interesting to see these two scenarios more or less improved the objective functions equally. To be more accurate, scenario3 had slightly better performance in some cases and scenario2 in some. Only with this amount of information and without going into details of solutions we can conclude that this job shop will get the best results from negotiating the lead time with

its vendor and does not need to outsource any more processes. However, the decision maker may want to investigate more into the results of the scenario3 and see whether any specific process should be outsourced. This model can also help run those kinds (like the second round) of what-if analysis.

We should note that the whole idea of this experiment is to demonstrate the ability of the model to perform outsourcing what-if scenarios. This is very common in real-world job shops to outsource the processes that can be done in-house to meet the tight due date, free up resources for other jobs, or both. For situations with tight due dates, the job shops may negotiate the expediting option with their vendors, which usually involves more cost. This model can help the company to forecast the improvement results and to perform the cost-benefit analysis. We can provide many examples of the job shops that outsource to free up their resources or for both objectives of meeting the tight deadline and freeing up the resources. For example, a fabrication shop does not have enough welders and decides to outsource fabricating of some subassemblies and then complete them in-house. They also want to negotiate the lead time with the vendor since it will exceed the expected completion date. This model can help the shop run the what-if scenario and see the result. The numerical result of different scenarios of the table 5.7 only can be considered as proof of the model's ability to perform such a what-if analysis.

5.4.4. Subassembly Consideration

This section aims to illustrate how the model handles the subassembly requirement for more complex products. In order to do so, a sample problem was generated with the below conditions:

(a) There are 100 jobs and 30 machines,

(b) There are 10 final assembly jobs,

(c) The remaining of 90 jobs are subassemblies distributed in minimum of 5 levels,

(d) 5 of final assemblies have due date and other 5 do not.

Table 5.7: Model performance for analyzing what-if outsourcing scenarios

Problem Size	Total Earliness/Tardiness					Makespan				
(JxM)	*orig.*	$Scn.^1$	$Scn.^2$	$Scn.^3$	*Imp%*	*orig.*	$Scn.^1$	$Scn.^2$	$Scn.^3$	*Imp%*
10 x 5	295	396	149	170	42%	529	622	394	369	30%
15 x 5	517	669	299	250	52%	395	384	323	275	30%
15 x 10	412	450	238	207	50%	441	537	269	268	39%
25 x 10	627	999	350	365	42%	583	815	406	460	21%
25 x 15	189	319	115	167	12%	564	857	458	506	10%
50 x 15	1190	2090	426	592	50%	825	1276	646	676	18%
50 x 25	2	406	2	1	75%	856	1183	584	665	22%
Average	**462**	**761**	**226**	**250**	**46%**	**599**	**811**	**440**	**460**	**25%**

Problem Size	Total Completion time					Aggregated Obj. Function				
(JxM)	*orig.*	$Scn.^1$	$Scn.^2$	$Scn.^3$	*Imp%*	*orig.*	$Scn.^1$	$Scn.^2$	$Scn.^3$	*Imp%*
10 x 5	529	622	394	369	30%	502	605	362	329	34%
15 x 5	395	384	323	275	30%	362	322	247	213	41%
15 x 10	866	1291	648	701	19%	691	1147	492	511	26%
25 x 10	1840	2677	1681	1728	6%	1866	2750	1459	1683	10%
25 x 15	1168	1608	890	983	16%	850	1397	692	795	6%
50 x 15	2252	3221	1614	1798	20%	1597	2532	1191	1284	20%
50 x 25	5131	6676	3699	3963	23%	2872	4859	2317	2605	9%
Average	**1740**	**2354**	**1321**	**1402**	**21%**	**1249**	**1945**	**966**	**1060**	**21%**

Orig.= Original sample problem with outsourcing probability of 30% and outsourcing leadtime of 3 times longer than in-house processing time;
$Scn.^1$ = Scenario no.1 of doubling the number of outsourced processes;
$Scn.^2$ = Scenario no.2 of expediting the outsourcing leadtime to half of original;
$Scn.^3$ = Combination of $Scn.^1$ and $Scn.^2$, doubling the outsourcing with half leadtime;
Imp% = Improvement percentage $Scn.^3$ compare to the original;

(e) Each job has 2 to 4 operations (293 operations in total),

(f) Chance of outsourcing is 30% (101 outsourced operations in total),

(g) Each in-house operation has 5 to 7 alternative machines with different process time,

(h) Every operation has a sequence dependent setup that can be attached or detached.

Figures 5.11, 5.12, and 5.13 demonstrate the BOM for this sample problem. In these figures, each node represents one job with the job number shown as underlined. "S" stands for the Start of the first operation, which is either starting the outsourcing process if the first operation is outsourced or starting its setup (attached or detached) process if it is an in-house operation. At each node, the finish time of the job is also identified by "F", which shows the end of processing of the last operation. We should note that the setup of the first operations of some jobs is detached, so if the machines are also available, the setup starts before the job arrives at the machine. So in that case, "S" or the start of the job will be before the finish time or "F" of the subassembly job. For instance, in figure 5.11 under job no.6, there are 3 jobs (job no. 48, 55 and 98) with "S" less than "F" of their sub assemblies. For example, job no. 48 starts at the time of 94 while its subassembly, job no. 39, finishes at 104. In these cases, we also show "B", which stands for the beginning of the processing of the first operation after the completion of the setup. This number could never be less than its subassemblies' greatest "F". For the rest of this section, we will refer to this as "the basic rule of subassembly relationship".

We checked the solution, and all 100 jobs follow this basic rule of subassembly relationship. For example, we can look at job numbers 6, 73, and 7 shown in figure 5.11. In these 3 jobs, each job has only one subassembly, and the "S" (or "B") of each job is exactly the same as the "F" time of its only subassembly. These jobs (job numbers 6, 73) demonstrate how the model schedules the jobs with no due date. For instance, subassemblies of job numbers 6 and 73, which are subject to the minimization of makespan and total completion time, are being completed one after another with no delay. Also, it is interesting to see job no. 7 with a tight due date is scheduled the same

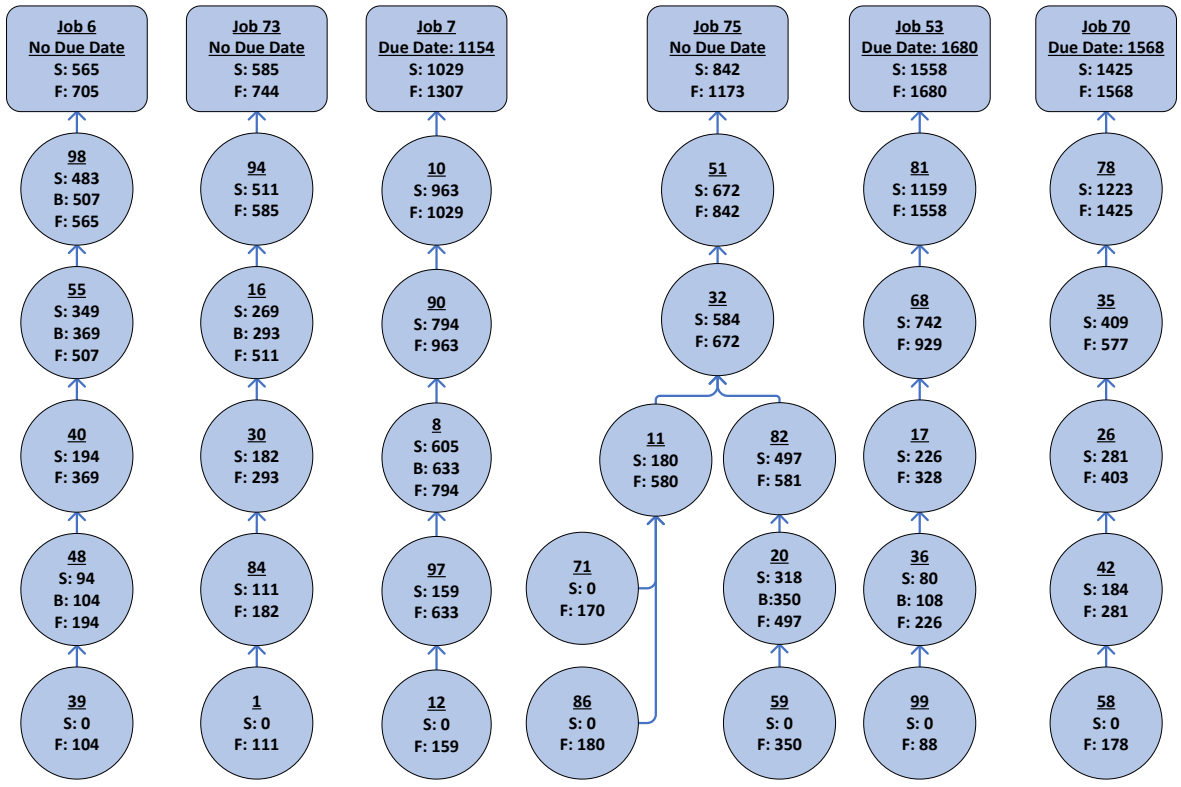
way that each job starts right after its subassembly to finish by the due date. However, as shown, it still finished later than its due date (1307 vs. 1154).

In the same figure, figure 5.11, job no.75 is also interesting to describe. It shows the application of the basic rule of subassembly relationship for the jobs with two subassemblies. As we can see, job no. 11 has two subassemblies, $J71$ with a finish time of 170 and $J86$ with a finish time of 180. So job no.11 can only start as early as 180, and it did. Job no.11 itself is a subassembly of job 32 along with job no.82. Job no.11 finishes at 580 and job no.82 at 581. Since their final assembly, job no.75, has no due date, job no.32 is expected to start at 581, but it is delayed for 3 time units (to 584). After checking the solution details, it appeared that it was due to the availability of the machine or, as it is called, the resource conflict. In this case, the machine no.22 is busy with operation no.2 of job no.54 from 568 to 584 time unit and only becomes available at 584 for accepting operation no.1 of job no.32.

To understand how the model schedules the jobs with relaxed due dates, we can look at job no.$J53$ and $J70$ shown in figure 5.11. Checking the start and finish times of each of their subassemblies shows that the model intentionally delays each subassembly unevenly to finish the job on time. In these two examples, the immediate child of the final assembly has been delayed much longer than other subassemblies (e.g., for the job no. 78, it is more than 640 units). Some other subassemblies are delayed too, but not that much. This is different in job no.91 and job no.57 shown in figure 5.12 that all the required intentional delay is applied on one subassembly, job no.46 and job no.66 respectively, and no other subassembly. These two jobs also have several examples of applying the basic rule of subassembly relationship in jobs with more than one immediate subassembly.

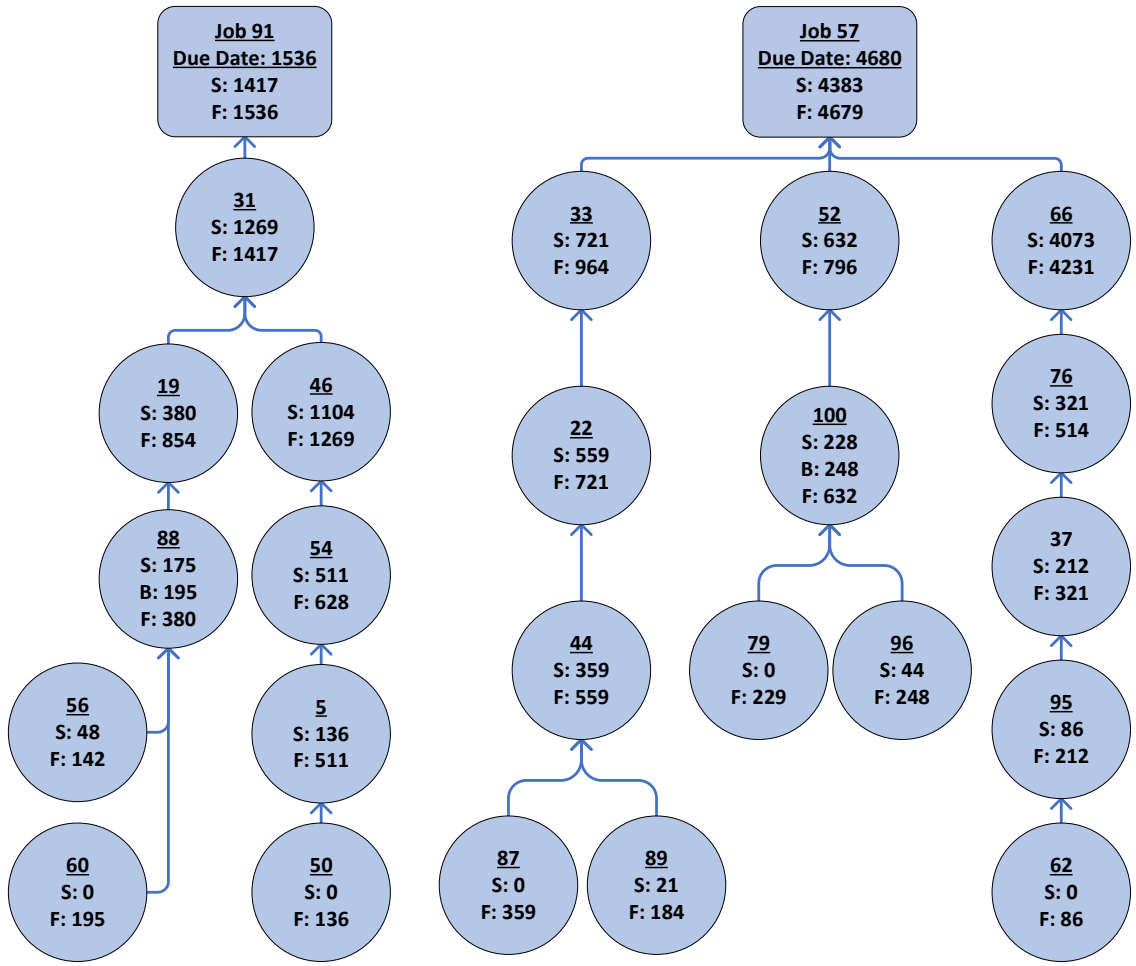
The scheduling of job no. 61 shown in figure 5.13 also demonstrates an interesting aspect of this model. Job no. 61 has no due date, so it should finish as soon as possible. However, in the schedule of two of its three subassemblies, there are several jobs with a delayed start. For example, job no.34 starts at 264 while its subassemblies are finished by 167. Similarly, job no. 18 and 24 themselves are not starting as soon as their immediate

subassemblies finish. To explain the logic, we should note that the longest production path among immediate children of job no.61 is job no. 29 that has no delay between any of its subassemblies. This shows how the model follows the critical path concept and uses the slack (float) of jobs. Another example of using the job slack can be seen in $J9$ in the same figure, figure 5.13, where job no.27 is delayed since the next job, job no.38, is waiting for its other subassembly that is the job no. 64.



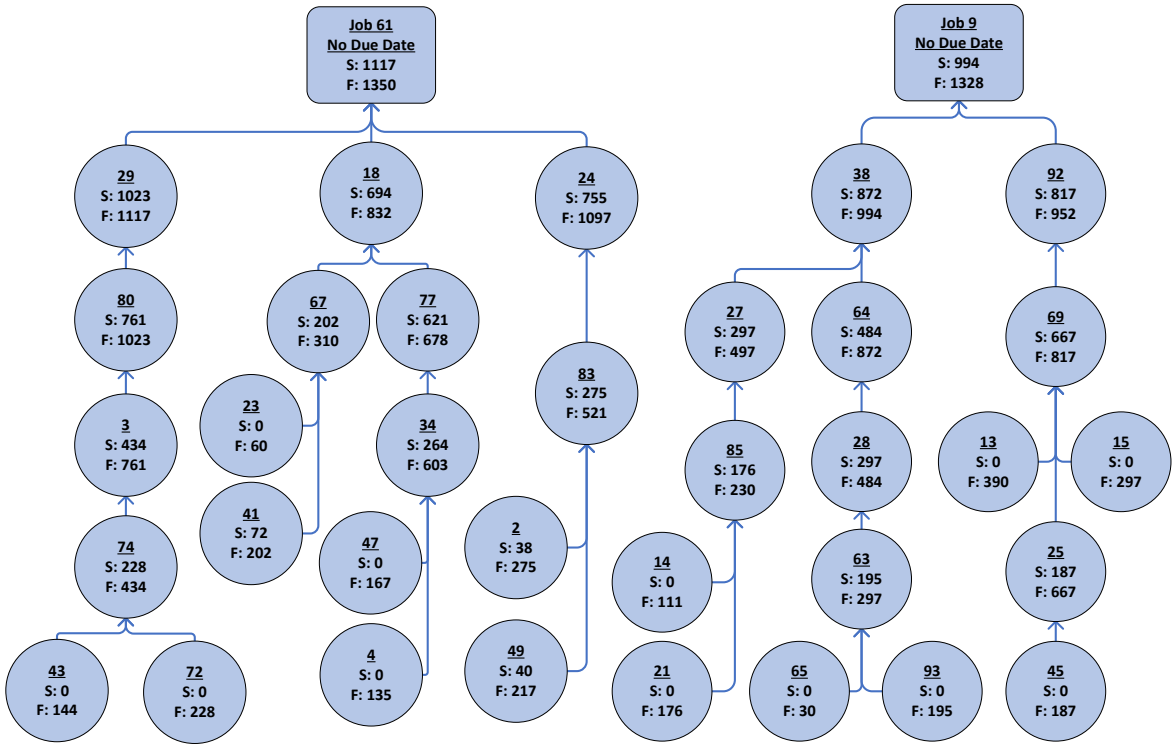
Note: Each node shows the job number,"S" or start time and "F" or finish time of the job,
if setup of the first operation is detached and starts before the job arrival,
the node also includes "B" or beginning of the first operation.

Figure 5.11: Subassembly consideration by the model-(a)



Note: Each node shows the job number,"S" or start time and "F" or finish time of the job, if setup of the first operation is detached and starts before the job arrival, the node also includes "B" or beginning of the first operation.

Figure 5.12: Subassembly consideration by the model-(b)



Note: Each node shows the job number, "S" or start time and "F" or finish time of the job, if setup of the first operation is detached and starts before the job arrival, the node also includes "B" or beginning of the first operation.

Figure 5.13: Subassembly consideration by the model-(c)

5.5. Discussion and Conclusion

Real-world job shops often receive orders with the requested completion time. This can be due to the increasing competitiveness level of the manufacturing world that forces industries to cut down all unnecessary costs, including the cost of inventory, which has led to the Just In Time concept. It is evident that scheduling plays the primary role in achieving the JIT goal. Hence, scheduling models with the objective of on-time completion or its equivalent of minimizing earliness and tardiness are very common. Per some reviews, minimizing tardiness is the second most common objective function after makespan. In Chapter 4, we illustrated that the Two Stage GA for FJSP is very efficient in optimizing a wide variety of objectives. However, in this chapter, we showed that the Two-Stage GA is not capable of handling earliness and tardiness objectives, and adding the intentional delay will solve this issue. The proposed genetic algorithm in this chapter has an additional LHS segment that determines the intentional delay of each job. A number of GA operators have also been modified or developed to accommodate these intentional delay gens in LHS with α and β.

In addition to the intentional delay, we added subassembly requirements and outsourcing considerations to the modified Two-Stage GA. These are two other realistic features that the regular GA for FJSPs are not considering. Outsourcing is based on the fact that many job shops rely on outsourcing processes since they either have no in-house capability or do not have enough capacity. In this model, each job can have multiple outsourced operations anywhere in the routing sequence. Another feature is subassembly consideration which defines the relationship between jobs. Any job can have multiple subassemblies in different levels that must be completed before the higher level job in the BOM can start. This adds another critical pragmatic capability missing from regular GAs for FJSP, which assumes no relationship between jobs exists. Adding all these features aims to make a more practical scheduling algorithm for real-world FJSPs.

In this chapter, we first developed the mathematical (MILP) model of the E/T

FJSP in the presence of outsourcing and subassembly requirements. Then we described the modified Two-Stage GA model and the GA operators to address this problem. Finally, to demonstrate how the model works and to prove its capacities, we created and solved several numerical problems as listed below:

(a) A prototype problem with 6 jobs and 4 machines,

(b) 7 examples with 20 to 50 jobs (8 to 15 final assemblies) and 10 to 30 machines to test E/T minimization performance,

(c) 7 examples with 10 to 50 jobs (3 to 10 final assemblies) and 5 to 25 machines for outsourcing analysis,

(d) A bigger size problem with 100 jobs (10 final assembly and 90 subassemblies) and 30 machines for testing subassembly consideration.

All the above sample problems are FJSPs with outsourcing, sequence-dependent setup, and subassembly requirement, which some of the final assemblies have due dates.

Below is the list of observations and conclusions that were made by solving the numerical examples:

- The original 2SGA is not able to minimize earliness and tardiness efficiently,
- Adding intentional delay enables the 2SGA model to minimize the earliness and tardiness while maintaining the ability to minimize makespan and total completion time,
- The model generates feasible solutions with respecting FJSP regular constraints, outsourcing and subassembly requirements,
- The model reaches good solutions (in most cases, earliness is zero),
- Jobs can have multi outsourced processes (consequently or separate) and can start or end with outsourced processes and still the model schedules them properly,
- The outsourced processes can be scheduled concurrently due to no limitation of the outsourcing,

- The model respects sequence-dependent setup and its nature (detached or attached) even if they fall before or after outsourced processes,
- The model can be used for analyzing what-if scenarios like the impact of different outsourcing strategies on objective functions,
- We concluded that the model can easily handle multilevel subassembly problems,
- The model uses the slack (float) of the jobs not on the subassembly critical path.

Chapter 6

Conclusion and Future Research

6.1. Summary and Conclusion

The systematic approach toward scheduling has been around for more than half a century, but it was only in 1990 that the basic FJSP was introduced as an extension to JSP. As we reviewed in chapters 1 and 2, researchers utilized several heuristic and meta-heuristic algorithms to solve JSP and FJSP due to their level of complexity and the fact they are classified as NP-hard. Among all these heuristic and meta-heuristic algorithms, the Genetic Algorithm is by far the most common solution approach used to address FJSP. Although GA and other mata heuristic algorithms made it possible to solve complex FJSPs, still the existing FJSP scheduling studies have assumed several simplifying conditions to FJPS in order to be modeled and solved. These assumptions are mainly impractical deviations from the real-world job shop manufacturing condition that makes these studies (as complex and advanced as they are) far from real-world schedulings.

As described in Section 1.4, this gap was also the primary motivating factor for the Ph.D. student that witnessed during his almost 20 years of experience. We noticed that job shop principles are the fundamental concept of many manufacturing companies, and lacking a good systematic approach for scheduling is one common issue even though many of these companies have implemented advanced ERP systems. So in this thesis, first, we tried to improve the performance of the GA for solving FJSP by introducing an

efficient Two-Stage GA. Then we removed several impractical assumptions in order to make the model closer to the real-world FJSPs. To lessen the gap between the theory and reality even further, we also developed multi-objective scheduling models for FJSP that cover a wide variety of different common objectives. The result of this research is published in one conference proceeding (Rooyani and Defersha (2019)) and in two academic journals (Defersha and Rooyani (2020), and Rooyani and Defersha (2022)). We also have another paper at the final drafting stage about Two-Stage GA for E/T scheduling at subassembly and outsourcing presence.

6.1.1. Contribution - Two-Stage GA

Chapter 3 describes the novel Two-Stage GA for FJSP, with the first stage being different from the regular GA approach. The first stage has a unique solution encoding that only determines the operation sequence for assignment and then uses a greedy approach to find the machine with the quickest completion time for each operation. This stage provides a very high-quality solution population that will be used as an initial population for the second stage. The second stage follows the common GA approach for FJSP which the GA dictates both sequencing and machine assignment. This stage explores the areas of the solution space that might have been missed due to the greedy nature of the first stage to reduce the chance of the GA falling in the local optimum.

In Chapter 3, we initially applied the Two-Stage GA on the basic FJSP and presented its result in 9^{th} IFAC Conference on Manufacturing Modelling, Management and Control in 2019 at Berlin, Germany that, later published at Rooyani and Defersha (2019). In that paper, we introduced the concept and solved several benchmark problems along with generated sample problems. We illustrated that the Two-Stage GA not only outperforms the Regular GA but also outperforms several other solution methodologies.

Since the main strength of the GA (and subsequently 2SGA) is solving the larger size and more complex problems, we applied the Two-Stage GA on a more comprehensive FJSP with sequence dependent attached/detached setup, machine release date, and operation lag time. The result is published at Defersha and Rooyani (2020). Similar

to the other paper, we tested the proposed algorithm by solving many benchmarks and randomly generated large-size problems. In this research, we demonstrated that the superiority of the Two-Stage GA is better recognized with more complex and comprehensive FJSPs. Hence, we concluded that the proposed algorithm is a viable choice for solving practical and real-world problems. We also applied parallel computation on Two-Stage GA, which improved its performance even further. However, the more interesting result we found was that the sequential version of the proposed algorithm (using a single CPU) outperformed a parallel implementation of the regular GA that uses 48 CPUs. This work has gained the research's attention, and Defersha and Rooyani (2020) paper has already been cited by about 40 studies, in addition to about 15 citations for Rooyani and Defersha (2019). Even Lei et al. (2022) referred to the Two-Stage GA as a "*state-of-the-art meta-heuristic algorithm*" and compared their algorithm performance with ours. They concluded that the Two-Stage GA surpasses their algorithm in terms of the quality of the solutions, but theirs is faster.

In addition to what we discussed in sections 3.1 and 3.5, the following is the summary of the contribution and conclusion that were made based on the results of these two studies:

(a) The Two-Stage GA is a very novel idea and a promising concept we developed through this research which has already attracted the attention of several scholars,

(b) The Two-Stage GA is based on a systematically designed solution representation and a greedy decoding mechanism of the first stage,

(c) The first stage creates a high-quality initial population for the second stage. The quality of this initial population is much better than those created by some other specialized initialization techniques from the literature,

(d) The Two-Stage GA is able to solve a wide range of FJSPs, from benchmark problems to fairly large-size sample problems,

(e) The Two-Stage GA is outperforming regular GA, specifically when the size and

complexity of the FJSP increases,

(f) Parallelization will improve the performance of Two-Stage GA, but even a sequential Two-Stage GA outperforms parallel regular GA.

6.1.2. Contribution - Multi-Objective Lot Streaming

As we discussed, the Two-Stage GA proved to be a good and effective approach for solving FJSPs, specifically with increasing the size and complexity of the problem. So in Chapter 4, we applied the Two-Stage GA to solve FJSPs that, in addition to the sequence dependent attached or detached setup, machine release date, and operation lag time, incorporates lot streaming and multiple objective functions. Lot streaming splits a production lot of a job into several independent sublots. Hence it enables the scheduling systems to 1) move the sublots from one machine to the next without waiting for the other sublots and 2) simultaneously process sublots of a given job on multiple machines. Therefore, lot streaming significantly reduces the completion time of the job, and as reviewed in Chapter 2, it is a very popular strategy for time-based competition in today's global market as well as in the literature. For instance, it has been adopted by Lean Manufacturing which usually refers to it as "one (or single) piece flow" in contrast with "batch production" as a tool to improve efficiency. Hence we wanted to show how the Two-Stage GA can handle the FJSP with lot streaming.

Another contribution of this chapter is incorporating 10 objective functions in FJSP. It is very rare to impossible to find a scheduling system with a single objective in the real world. However, the majority of scheduling studies in the literature considered only a single objective function, mainly makespan. With the ultimate goal of lessening the gap between the theory and practice of the FJSP schedule in mind, in this study, we wanted to incorporate multi-objectives FJSP within Two-Stage GA as well. The 10 objective functions we have considered are the minimization of (1) makespan, (2) maximum sublot flowtime, (3) total sublot flowtime, (4) maximum job flowtime, (5) total job

flowtime, (6) maximum sublot finish-time separation, (7) total sublot finish-time separation, (8) maximum machine load, (9) total machine load, and (10) maximum machine load difference. Also, this study provides several numerical studies that demonstrate the importance of multi-objective optimization, specifically in large-size problems. The result of this research is published in Rooyani and Defersha (2022).

Here is the summary of the conclusion and contributions of this study that has been already described in Sections 4.1 and 4.5:

(a) The Two-Stage GA is very capable of handling more complex FJSPs that, in addition to sequence dependent attached or detached setup, machine release date, and operation lag time, include lot streaming,

(b) The Two-Stage GA can jointly optimize multi-objective FJSPs,

(c) The first stage generates solutions that are highly improved in all of the ten objective function terms,

(d) The numerical examples prove the greater need for multi-objective optimization in larger problems due to their interactions and tradeoff. This emphasizes the fact that the single objective scheduling algorithms are not suitable for real industrial systems that need to optimize several objectives at the same time,

(e) There is a need for some newly proposed objective functions. For example, workload balancing in FJSP may not be fully achieved by minimizing the maximum or the total workload or both, and the newly proposed objective of minimizing the maximum workload difference can result in a better workload balance when considered along with the minimization of the maximum and/or the total workload,

(f) Similarly, minimizing the maximum sublot finish-time separation and total sublot finish-time separation can decrease work-in-process inventory with minimal impact on sublot and job flowtime,

(g) The Two-Stage GA is outperforming regular GA in both convergence speed and final solution quality in solving the multi-objective FJSP lot streaming,

(h) Parallel computation improves the performance of the Two-Stage Genetic Algorithm further,

(i) A sequential Two-Stage GA is outperforming parallel regular GA,

(j) Among different selection GA operators, tournament selection with the smaller size of tournament outperforms proportional and linear ranking selection operators,

(k) Analysis of variance shows there is no interaction between mutation and crossover probabilities, and they can be defined separately.

6.1.3. Contribution - E/T Schedule with Assembly and Outsourcing

Chapter 4 illustrated that the Two-Stage GA could efficiently solve comprehensive FJSPs with a wide range of objectives from makespan and flowtime to machine load maximization. However, in Chapter 5, we demonstrated that the Two-Stage GA is not able to address Earliness/Tardiness objectives. It was not a surprise since the core idea of the Two-Stage GA is adding a greedy mechanism that assigns the machine with the shortest completion time to each operation. This is a pragmatic issue for 2SGA since usually some of the orders that job shops receive have requested completion time or due dates with some sort of penalties or costs (either internally like inventory cost or storage fee or externally like late charges or liquidated damages). In order to address this issue, we modified the Two-Stage GA solution encoding (added an LHS segment to the encoding chromosome) and GA operators to accommodate the intentional delay. The intentional delay enables the algorithm to release the jobs only during a specific time window and makes it possible to finish the job on time and not early. As a result, the model can have several jobs with due dates, while the rest of the orders should be finished ASAP.

The outsourcing capability and assembly requirement are the two other contributions of the FJSP model of Chapter 5, in addition to modifying the original Two-Stage GA to accommodate intentional delay and minimization of Earliness/Tardiness. As discussed in Chapter 2, in order to simplify the modeling process, the classic FJSPs assume

all operations are done in-house, and there is no relationship between the jobs. However, these conditions are the opposite of real-world job shop situations. Hence in this model, the jobs can have some outsourced operations at any step of their routing while their other operations have several capable machines. Also, the model allows the jobs to have several sub-assemblies on multiple levels. This chapter provides several randomly generated problems of different sizes with outsourcing, subassemblies, and due dates to demonstrate the performance of the modified Two-Stage GA. We are in the final stage of drafting the paper to publish the results of this chapter.

In addition to sections 5.1 and 5.5 describing the contribution and conclusion of this model in detail, we also provide a summary here:

(a) The original Two-Stage GA is not able to minimize earliness and tardiness objective functions, but adding the intentional delay solves this issue,
(b) The modified Two-Stage GA generates feasible and good solutions with respecting FJSP regular constraints, outsourcing and subassembly requirements, and in most cases zero earliness,
(c) The model allows the jobs to have several outsourced processes (consequently or separate) at any stage of routing (including the first and the last operation). Furthermore, at any point, we can have several operations under outsourcing processing (no capacity limit),
(d) The jobs also can have several subassemblies in multi-levels,
(e) The model can be used for analyzing what-if scenarios like the impact of different outsourcing strategies on objective functions.

6.2. Future Research

Research in the Flexible job shop scheduling field, and generally in the scheduling field, have many opportunities to be more practical. In this section, we recommend Dynamic FJSP scheduling and Multi-Resource Scheduling as immediate future research fields and Meta Scheduling and Industry 4.0 as a more long-term expansion for applying Two-Stage

GA.

6.2.1. Dynamic Scheduling

This study, similar to the majority of scheduling studies, has assumed the scheduling takes place in a static production environment, where 1) the shop floor information is known in advance, and 2) the deterministic scheduling scheme will not be changed during the entire planning horizon. The fact that most scheduling algorithms have only assumed static conditions is a major limitation of scheduling literature. Since manufacturing systems operate in dynamic environments where frequent real-time events occur and make a previously feasible schedule obsolete. Shen et al. (2014) categorizes all the scheduling disruptions that can outdate near-optimal schedules generated by classic (static) approaches into data uncertainties and real-time disruptive events. Baykasoğlu and Karaslan (2017) have listed five types of real-time disruptive events for a job shop scheduling as "new order arrivals", "machine breakdowns", "arrival of urgent orders", "changes of the due dates", and "order cancellation". Hence there is a need for algorithms that reflect these real-life scheduling situations.

This newer type of scheduling algorithm is called dynamic scheduling and takes into consideration some of those data uncertainties and real-time disruptive events. We can see an increasing number of scholars paying attention to Dynamic Flexible Job Shop Scheduling (DFJSP) that can respond to the unexpected events of the flexible job shop in real-time. Shen et al. (2014) listed "shop efficiency", "schedule robustness", and "system stability" as the typical objectives of the existing dynamic scheduling studies. Hence, extending Two-Stage GA's application to address comprehensive FJSP in a dynamic situation can be an immediate future research field.

6.2.2. Multi-Resource Scheduling

In this research, similar to most FJSP studies, we assumed that machines were the only limited resources needed to be considered in the scheduling system. In other words, the classic FJSP assumes that the next assigned operation can start as soon as the machine is

available, which is not a good reflection of real-world job shop conditions. The operators are the most essential resource type missing in the classic FJSP since, usually, in real job shops, there is not one operator per machine. The majority of job shops have highly skilled operators who can run several machines, so the scheduling system needs to assure the operator who can run the machine is also available to start the next operation. Operators are not the only type of missing resources in the FJSP scheduling, as we can name setup technicians, jig and fixtures, special tools, or material handling devices as some other examples of shared resources in the job shop environment that are needed to be available before an operation can start. Another interesting example of shared resources is studied by Singh and Weiskircher (2011) in some coal mines in a remote off-grid region that rely on one power plant that can produce limited electricity. Resources can be categorized into two types of renewable resources like labour and nonrenewable resources like fuel or perishable tools (Liu et al. (2018)). We believe applying the Two-Stage GA to solve multi-resource scheduling, at minimum dual resources of machines and operators, can be a fruitful future research field that makes the model more practical.

6.2.3. Industry 4.0

Industry 4.0, or the Fourth Industrial Revolution, also called the "smart factory" conceptualizes the integration of advanced technologies such as the Internet of Things, big data, and Artificial Intelligence (AI) with enterprise resource planning, process control management, and production technologies. Scheduling, as the core of enterprise resource planning and manufacturing management systems, has a crucial role in improving productivity and resource utilization in a manufacturing factory in the era of Industry 4.0 (Chang et al. (2022)). We believe GA, and consequently, Two-Stage GA can provide a good and practical solution methodology for solving more complicated scheduling problems in a complex manufacturing system with integrated advanced technology.

6.2.4. Meta-Scheduling

This study proposes an effective solution for the comprehensive multi-objective flexible job shop scheduling problem. The" job shop" is a specific and defined production framework that the FJSP tries to address. Similarly, most scheduling algorithms in the literature are frameworks specific such as a single machine shop, parallel machines shop, flexible flow shop, flexible job shop, and cellular manufacturing. This fact that most algorithms are framework specific such as job-shop or flow-shop, which rarely match real manufacturing systems, is a major problem in translating scheduling algorithms into practical solutions for industries (Mejía et al. (2012)). Since most manufacturing companies utilize a combination of different manufacturing systems, it is rare to find a manufacturing plant that follows a single platform perfectly.

Although many researchers have recognized the benefit of system-independent scheduling tools, only a few published attempts exist. For example, Davis and Fox (1994) and McIlhagga (2005) made initial attempts to develop system-independent scheduling frameworks, but both fell short in terms of the generality and flexibility of the frameworks they propose. Smith and Becker (1997) and Metaxiotis et al. (2001) created unified scheduling ontologies, but the ontologies are not well suited for straightforward problem representation and solution encoding. A scheduling methodology that can be used across domains (job shop, traveling salesman, vehicle routing) was patented and presented in Montana (2002). However, the approach lacks depth in each domain and cannot address the challenges of today's complex manufacturing systems. Framinan et al. (2014) provided guidelines and architecture for unified scheduling tools. However, the authors provided broad-brush approaches to the problem that lacks specific algorithm development.

So it is crucial to develop a generalized framework and algorithm, called meta-scheduling, that can schedule a wide range of manufacturing systems without altering its internal code. Such a generalized meta-scheduling algorithm can be more usable in practical settings as it will enable the manufacturing system scheduler to use the same scheduling tool in a variety of scenarios. Such a system will allow ERP developers

to incorporate detailed scheduling functionalities in their software. Hence it enables industries to acquire automated scheduling tools without the need to develop expensive customized algorithms.

We believe the core concept of Two-Stage GA which is adding a greedy approach to find better solutions in the first stage of the GA to create a high-quality initial population for the second stage can be utilized to handle a wide range of discrete manufacturing systems with greater operational complexity.

Bibliography

Tamer F. Abdelmaguid. A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times. *Applied Mathematics and Computation*, 260:188–203, 2015. ISSN 00963003. doi: 10.1016/j.amc.2015.03.059.

Nabil R. Adam, J.Will M. Bertrand, Diane C. Morehead, and Julius Surkis. Due date assignment procedures with dynamically updated coefficients for multi-level assembly job shops. *European Journal of Operational Research*, 68(2):212–227, jul 1993. ISSN 03772217. doi: 10.1016/0377-2217(93)90304-6. URL https://linkinghub.elsevier.com/retrieve/pii/0377221793903046.

Joseph Adams, Egon Balas, and Daniel Zawack. Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3):391–401, 1988. ISSN 00251909. doi: 10.1287/mnsc.34.3.391. URL https://www.jstor.org/stable/2632051.

Mohammad Mahdi Ahmadian and Amir Salehipour. The just-in-time job-shop scheduling problem with distinct due-dates for operations. *Journal of Heuristics*, 27(1-2):175–204, apr 2021. ISSN 1381-1231. doi: 10.1007/s10732-020-09458-6. URL http://link.springer.com/10.1007/s10732-020-09458-6.

Umbarkar A.J. and Sheth P.D. Crossover operators in Genetic Algorithms: A review. *ICTACT Journal on Soft Computing*, 06(01):1083–1092, oct 2015. ISSN 09766561. doi: 10.21917/ijsc.2015.0150. URL http://ictactjournals.in/ArticleDetails.aspx?id=2109.

Onur Serkan Akgündüz and Semra Tunalı. An adaptive genetic algorithm approach for the mixed-model assembly line sequencing problem. *International Journal of Production Research*, 48(17):5157–5179, sep 2010. ISSN 0020-7543. doi: 10.1080/00207540903117857. URL https://www.tandfonline.com/doi/full/10.1080/00207540903117857.

Onur Serkan Akgündüz and Semra Tunalı. A review of the current applications of genetic algorithms in mixed-model assembly line sequencing. *International Journal of Production Research*, 49(15):4483–4503, aug 2011. ISSN 0020-7543. doi: 10.1080/00207543.2010.495085. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2010.495085.

Nasr Al-Hinai and T. Y. Elmekkawy. An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem. *Flexible Services and Manufacturing Journal*, 23(1):64–85, 2011. ISSN 19366582. doi: 10.1007/s10696-010-9067-y.

Nasr Al-Hinai and T.Y. ElMekkawy. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics*, 132(2):279–291, aug 2011. ISSN 09255273. doi: 10.1016/j.ijpe.2011.04.020. URL https://linkinghub.elsevier.com/retrieve/pii/S0925527311001952.

Arianna Alfieri, Shuyu Zhou, Rosario Scatamacchia, and Steef L. van de Velde. Dynamic programming algorithms and Lagrangian lower bounds for a discrete lot streaming problem in a two-machine flow shop. *4OR*, 19(2):265–288, jun 2021. ISSN 1619-4500. doi: 10.1007/s10288-020-00449-8. URL https://link.springer.com/10.1007/s10288-020-00449-8.

Ali Allahverdi, C. T. Ng, T. C.E. Cheng, and Mikhail Y. Kovalyov. A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032, 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.06.060.

M. Amiri, M. Zandieh, M. Yazdani, and A. Bagheri. A variable neighbourhood search algorithm for the flexible job-shop scheduling problem. *International Journal of Production Research*, 48(19):5671–5689, oct 2010. ISSN 0020-7543. doi: 10.1080/00207540903055743. URL https://www.tandfonline.com/doi/full/10.1080/00207540903055743.

Muhammad Kamal Amjad, Shahid Ikramullah Butt, Rubeena Kousar, Riaz Ahmad, Mujtaba Hassan Agha, Zhang Faping, Naveed Anjum, and Umer Asgher. Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems. *Mathematical Problems in Engineering*, 2018:1–32, 2018. ISSN 15635147. doi: 10.1155/2018/9270802.

A. Bagheri and M. Zandieh. Bi-criteria flexible job-shop scheduling with sequence-dependent setup times - Variable neighborhood search approach. *Journal of Manufacturing Systems*, 30(1):8–15, 2011. ISSN 02786125. doi: 10.1016/j.jmsy.2011.02.004.

A. Bagheri, M. Zandieh, Iraj Mahdavi, and M. Yazdani. An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems*, 26(4):533–541, 2010. ISSN 0167739X. doi: 10.1016/j.future.2009.10.004.

Dražen Bajer, Goran Martinović, and Janez Brest. A population initialization method for evolutionary algorithms based on clustering and Cauchy deviates. *Expert Systems with Applications*, 60:294–310, 2016. ISSN 09574174. doi: 10.1016/j.eswa.2016.05.009.

K R Baker. *Introduction to Sequence and Scheduling*. Wiley, NY, 1974.

Kenneth R. Baker and Gary D. Scudder. Sequencing with Earliness and Tardiness Penalties: A Review. *Operations Research*, 38(1):22–36, feb 1990. ISSN 0030-364X. doi: 10.1287/opre.38.1.22. URL http://pubsonline.informs.org/doi/10.1287/opre.38.1.22.

Adil Baykasoğlu and Fatma S. Karaslan. Solving comprehensive dynamic job shop scheduling problem by using a GRASP-based approach. *International Journal of*

Production Research, 55(11):3308–3325, jun 2017. ISSN 0020-7543. doi: 10.1080/00207543.2017.1306134. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2017.1306134.

Dennis Behnke and Martin Josef Geiger. Test Instances for the Flexible Job Shop Scheduling Problem with Work Centers. Technical report, Helmut-Schmidt-University, Logistics Management Department, Hamburg, Germany, 2012. URL http://edoc.sub.uni-hamburg.de/hsu/volltexte/2012/2982/%0Ahttp://opus.unibw-hamburg.de/volltexte/2012/2982/.

Abir Ben Hmida, Mohamed Haouari, Marie Jos Huguet, and Pierre Lopez. Discrepancy search for the flexible job shop scheduling problem. *Computers and Operations Research*, 37(12):2192–2201, dec 2010. ISSN 03050548. doi: 10.1016/j.cor.2010.03.009. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054810000638.

D. Bergamaschi, R. Cigolini, M. Perona, and A. Portioli. Order review and release strategies ina job shop environment: A review and a classification. *International Journal of Production Research*, 35(2):399–420, feb 1997. ISSN 0020-7543. doi: 10.1080/002075497195821. URL http://www.tandfonline.com/doi/abs/10.1080/002075497195821.

Andrzej Bożek and Frank Werner. Flexible job shop scheduling with lot streaming and sublot size optimisation. *International Journal of Production Research*, 56(19):6391–6411, 2018. ISSN 1366588X. doi: 10.1080/00207543.2017.1346322. URL https://doi.org/10.1080/00207543.2017.1346322.

Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, sep 1993. ISSN 0254-5330. doi: 10.1007/BF02023073. URL http://link.springer.com/10.1007/BF02023073.

P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, dec 1990. ISSN 0010-485X. doi: 10.1007/BF02238804. URL http://link.springer.com/10.1007/BF02238804.

U Buscher and L Shen. An Integrated Tabu Search Algorithm for the Lot Streaming Problem in Job Shops. *European Journal of Operational Research*, In Press(DOI: 10.1016/j.ejor.2008.11.046), 2008.

Banu Çaliş and Serol Bulkan. A research survey: review of AI solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5):961–973, oct 2015. ISSN 0956-5515. doi: 10.1007/s10845-013-0837-8. URL http://link.springer.com/10.1007/s10845-013-0837-8.

R. Caponetto, L. Fortuna, S. Fazzino, and M.G. Xibilia. Chaotic sequences to improve the performance of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):289–304, jun 2003. ISSN 1089-778X. doi: 10.1109/TEVC.2003.810069. URL http://ieeexplore.ieee.org/document/1206449/.

F. T.S. Chan, T. C. Wong, and L. Y. Chan. Lot streaming for product assembly in job shop environment. *Robotics and Computer-Integrated Manufacturing*, 24(3):321–331, 2008. ISSN 07365845. doi: 10.1016/j.rcim.2007.01.001.

Felix T.S. Chan, T. C. Wong, and P. L.Y. Chan. Equal size lot streaming to job-shop scheduling problem using genetic algorithms. In *IEEE International Symposium on Intelligent Control - Proceedings*, Proceedings of the 2004 IEEE International Symposium on Intelligent Control, pages 472–476, Taipei, Taiwan, September 2-4, 2004, 2004. IEEE. doi: 10.1109/isic.2004.1387729.

Felix T.S. Chan, T. C. Wong, and L. Y. Chan. The application of genetic algorithms to lot streaming in a job-shop scheduling problem. *International Journal of Production Research*, 47(12):3387–3412, 2009. ISSN 00207543. doi: 10.1080/00207540701577369.

Jen Huei Chang and Huan Neng Chiu. A comprehensive review of lot streaming. *International Journal of Production Research*, 43(8):1515–1536, apr 2005. ISSN 0020-7543. doi: 10.1080/00207540412331325396. URL http://www.tandfonline.com/doi/full/10.1080/00207543.2013.774506http://www.tandfonline.com/doi/abs/10.1080/00207540412331325396.

Jingru Chang, Dong Yu, Yi Hu, Wuwei He, and Haoyu Yu. Deep Reinforcement Learning for Dynamic Flexible Job Shop Scheduling with Random Job Arrival. *Processes*, 10 (4):760, apr 2022. ISSN 2227-9717. doi: 10.3390/pr10040760. URL https://www.mdpi.com/2227-9717/10/4/760.

Imran A. Chaudhry, Abdul Munem Khan, and Abid Ali Khan. A genetic algorithm for flexible job shop scheduling. In *Lecture Notes in Engineering and Computer Science*, volume 1 LNECS of *Proceedings of the World Congress on Engineering*, pages 703–708, London, U.K., 2013. ISBN 9789881925107. doi: 10.1109/robot.1999.772512.

Imran Ali Chaudhry and Abid Ali Khan. A research survey: Review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23 (3):551–591, may 2016. ISSN 14753995. doi: 10.1111/itor.12199. URL http://doi.wiley.com/10.1111/itor.12199.

Binchao Chen and Timothy I. Matis. A flexible dispatching rule for minimizing tardiness in job shop scheduling. *International Journal of Production Economics*, 141 (1):360–365, jan 2013. ISSN 09255273. doi: 10.1016/j.ijpe.2012.08.019. URL https://linkinghub.elsevier.com/retrieve/pii/S0925527312003672.

Haoxun Chen, Juergen Ihlow, and Carsten Lehmann. Genetic algorithm for flexible job-shop scheduling. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2, pages 1120–1125, 1999.

K. J. Chen and P. Ji. Development of a genetic algorithm for scheduling products with a multi-level structure. *The International Journal of Advanced Manufacturing Technology*, 33(11-12):1229–1236, aug 2007. ISSN 0268-3768. doi: 10.1007/s00170-006-0561-z. URL http://link.springer.com/10.1007/s00170-006-0561-z.

Tzu-Li Chen, Chen-Yang Cheng, and Yi-Han Chou. Multi-objective genetic algorithm for energy-efficient hybrid flow shop scheduling with lot streaming. *Annals of Operations Research*, 290(1-2):813–836, jul 2020. ISSN 0254-5330.

doi: 10.1007/s10479-018-2969-x. URL http://link.springer.com/10.1007/s10479-018-2969-x.

M. Cheng, N. J. Mukherjee, and S. C. Sarin. A review of lot streaming. *International Journal of Production Research*, 51(23-24):7023–7046, 2013. ISSN 00207543. doi: 10.1080/00207543.2013.774506.

Ming Cheng, Subhash C. Sarin, and Sanchit Singh. Two-stage, single-lot, lot streaming problem for a 1 + 2 hybrid flow shop. *Journal of Global Optimization*, 66(2):263–290, oct 2016. ISSN 15732916. doi: 10.1007/s10898-015-0298-z. URL http://link.springer.com/10.1007/s10898-015-0298-z.

Tsung Che Chiang and Hsiao Jou Lin. A simple and effective evolutionary algorithm for multiobjective flexible job shop scheduling. *International Journal of Production Economics*, 141(1):87–98, jan 2013. ISSN 09255273. doi: 10.1016/j.ijpe.2012.03.034. URL https://linkinghub.elsevier.com/retrieve/pii/S0925527312001429.

Daeyoung Chung, Kichang Lee, Kitae Shin, and Jinwoo Park. A new approach to job shop scheduling problems with due date constraints considering operation subcontracts. *International Journal of Production Economics*, 98(2):238–250, nov 2005. ISSN 09255273. doi: 10.1016/j.ijpe.2004.05.023. URL https://linkinghub.elsevier.com/retrieve/pii/S0925527305000149.

R.W. Conway, W.L. Maxwell, and W.L Miller. *Theory of Scheduling*. Addison-Wesley, MA, 1967.

Marco Antonio Cruz-Chávez, Martín G. Martínez-Rangel, and Martín H. Cruz-Rosales. Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem. *International Transactions in Operational Research*, 24(5):1119–1137, 2017. ISSN 14753995. doi: 10.1111/itor.12195.

Min Dai, Dunbing Tang, Adriana Giret, and Miguel A. Salido. Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. *Robotics and Computer-Integrated Manufacturing*, 59(April):143–157, 2019. ISSN 07365845. doi: 10.1016/j.rcim.2019.04.006.

Fatemeh Daneshamooz, Parviz Fattahi, and Seyed Mohammad Hassan Hosseini. Scheduling in a flexible job shop followed by some parallel assembly stations considering lot streaming. *Engineering Optimization*, pages 1–20, apr 2021. ISSN 0305-215X. doi: 10.1080/0305215X.2021.1887168. URL https://www.tandfonline.com/doi/full/10.1080/0305215X.2021.1887168.

G Davis and M S Fox. ODO: Constraint-Based Architecture for Representing and Reasoning about Scheduling Problems. In *Proceedings of the 3rd Industrial Engineering Research Conference*, pages 461–466, Atlanta GA, 1994. URL http://www.eil.utoronto.ca/wp-content/uploads/scheduling/papers/davis-ierc94.pdf.

L. De Giovanni and F. Pezzella. An Improved Genetic Algorithm for the Distributed and Flexible Job-shop Scheduling problem. *European Journal of Operational Research*, 200 (2):395–408, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.01.008.

F. M. Defersha and M. Chen. A parallel genetic algorithm for dynamic cell formation in cellular manufacturing systems. *International Journal of Production Research*, 46 (22):6389–6413, 2008. ISSN 00207543. doi: 10.1080/00207540701441962.

Fantahun M. Defersha and Saber Bayat Movahed. Linear programming assisted (not embedded) genetic algorithm for flexible jobshop scheduling with lot streaming. *Computers and Industrial Engineering*, 117(February):319–335, 2018. ISSN 03608352. doi: 10.1016/j.cie.2018.02.010. URL https://doi.org/10.1016/j.cie.2018.02.010.

Fantahun M. Defersha and Mingyuan Chen. A hybrid genetic algorithm for flowshop lot streaming with setups and variable sublots. *International Journal of Production Research*, 48(6):1705–1726, 2010a. ISSN 00207543. doi: 10.1080/00207540802660544.

Fantahun M. Defersha and Mingyuan Chen. A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *International Journal of Advanced Manufacturing Technology*, 49(1-4):263–279, 2010b. ISSN 02683768. doi: 10.1007/s00170-009-2388-x.

Fantahun M. Defersha and Mingyuan Chen. Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. *International Journal of Production Research*, 50(8):2331–2352, 2012a. ISSN 00207543. doi: 10.1080/00207543.2011.574952. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2011.574952.

Fantahun M. Defersha and Danial Rooyani. An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Computers and Industrial Engineering*, 147:106605, sep 2020. ISSN 03608352. doi: 10.1016/j.cie.2020.106605. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835220303399.

Fantahun Melaku Defersha and Mingyuan Chen. Mathematical model and parallel genetic algorithm for hybrid flexible flowshop lot streaming problem. *International Journal of Advanced Manufacturing Technology*, 62(1-4):249–265, 2012b. ISSN 02683768. doi: 10.1007/s00170-011-3798-0.

Yunus Demir and Selçuk Kürşat Işleyen. An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations. *International Journal of Production Research*, 52(13):3905–3921, 2014. ISSN 1366588X. doi: 10.1080/00207543.2014.889328.

Yunus Demir and S. Kürşat Işleyen. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 37(3):977–988, 2013. ISSN 0307904X. doi: 10.1016/j.apm.2012.03.020.

Imen Driss, Kinza Nadia Mouss, and Assia Laggoun. A new genetic algorithm for flexible

job-shop scheduling problems. *Journal of Mechanical Science and Technology*, 29(3): 1273–1281, 2015. ISSN 19763824. doi: 10.1007/s12206-015-0242-7.

Hoda A. ElMaraghy. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, 17(4):261–276, oct 2005. ISSN 0920-6299. doi: 10.1007/s10696-006-9028-7. URL http://link.springer.com/10.1007/s10696-006-9028-7.

Meriem Ennigrou and Khaled Ghédira. New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach. *Autonomous Agents and Multi-Agent Systems*, 17(2):270–287, oct 2008. ISSN 1387-2532. doi: 10.1007/s10458-008-9031-3. URL http://link.springer.com/10.1007/s10458-008-9031-3.

Imen Essafi, Yazid Mati, and Stéphane Dauzère-Pérès. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers and Operations Research*, 35(8):2599–2616, aug 2008. ISSN 03050548. doi: 10.1016/j.cor.2006.12.019. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054806003194.

M. B. Fakhrzad. A New Multi-objective Job Shop Scheduling with Setup Times Using a Hybrid Genetic Algorithm. *International Journal of Engineering*, 26(2 (B)):207–218, feb 2013. ISSN 10252495. doi: 10.5829/idosi.ije.2013.26.02b.11. URL http://www.ije.ir/Vol26/No2/B/11.pdf.

E. Falkenauer and S. Bouffouix. A genetic algorithm for job shop. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 824–829. IEEE Comput. Soc. Press, 1991. ISBN 0-8186-2163-X. doi: 10.1109/ROBOT.1991.131689. URL http://ieeexplore.ieee.org/document/131689/.

Huali Fan, Hegen Xiong, and Mark Goh. Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints. *Computers and Operations Research*, 134:105401, oct 2021. ISSN

03050548. doi: 10.1016/j.cor.2021.105401. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054821001672.

Kan Fang, Wenchang Luo, and Ada Che. Speed scaling in two-machine lot-streaming flow shops with consistent sublots. *Journal of the Operational Research Society*, 72(11): 2429–2441, nov 2021. ISSN 0160-5682. doi: 10.1080/01605682.2020.1796533. URL https://www.tandfonline.com/doi/full/10.1080/01605682.2020.1796533.

Miguel A. Fernández Romero, Antonin Ponsich, Eric A. Rincón García, and Roman A. Mora Gutiérrez. A heuristic algorithm based on tabu search for the solution of flexible job shop scheduling problems with lot streaming. *GECCO 2018 - Proceedings of the 2018 Genetic and Evolutionary Computation Conference*, pages 285–292, 2018. doi: 10.1145/3205455.3205534.

Mark S. Fox and Stephen F. Smith. ISIS-a knowledge-based system for factory scheduling. *Expert Systems*, 1(1):25–49, jul 1984. ISSN 0266-4720. doi: 10.1111/j.1468-0394.1984.tb00424.x. URL https://onlinelibrary.wiley.com/doi/10.1111/j.1468-0394.1984.tb00424.x.

Jose M. Framinan, Rainer Leisten, and Rubén Ruiz García. *Manufacturing Scheduling Systems*. Springer London, London, 2014. ISBN 978-1-4471-6271-1. doi: 10.1007/978-1-4471-6272-8. URL http://link.springer.com/10.1007/978-1-4471-6272-8.

Jose M. Framinan, Paz Perez-Gonzalez, and Victor Fernandez-Viagas. Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. *European Journal of Operational Research*, 273(2):401–417, mar 2019. ISSN 03772217. doi: 10.1016/j.ejor.2018.04.033. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221718303369.

Helio Yochihiro Fuchigami and Socorro Rangel. A survey of case studies in production

scheduling: Analysis and perspectives. *Journal of Computational Science*, 25:425–436, mar 2018. ISSN 18777503. doi: 10.1016/j.jocs.2017.06.004. URL https://linkinghub.elsevier.com/retrieve/pii/S1877750317300741.

Jie Gao, Mitsuo Gen, and Linyan Sun. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, 17 (4):493–507, aug 2006. ISSN 09565515. doi: 10.1007/s10845-005-0021-x. URL http://link.springer.com/10.1007/s10845-005-0021-x.

Jie Gao, Mitsuo Gen, Linyan Sun, and Xiaohui Zhao. A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers and Industrial Engineering*, 53(1):149–162, 2007. ISSN 03608352. doi: 10.1016/j.cie.2007.04.010.

Jie Gao, Linyan Sun, and Mitsuo Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research*, 35(9):2892–2907, 2008. ISSN 03050548. doi: 10.1016/j.cor.2007.01.001.

Jinsheng Gao, Xiaomin Zhu, Kaiyuan Bai, and Runtong Zhang. New controllable processing time scheduling with subcontracting strategy for no-wait job shop problem. *International Journal of Production Research*, 60(7):2254–2274, apr 2022. ISSN 0020-7543. doi: 10.1080/00207543.2021.1886368. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2021.1886368.

Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Tay Jin Chua, Chin Soon Chong, Tian Xiang Cai, and Qan Ke Pan. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Systems with Applications*, 42(21):7652–7663, nov 2015. ISSN 09574174. doi: 10.1016/j.eswa.2015.06.004. URL https://linkinghub.elsevier.com/retrieve/pii/S095741741500398X.

Kaizhou Gao, Zhiguang Cao, Le Zhang, Zhenghua Chen, Yuyan Han, and Quanke Pan. A review on swarm intelligence and evolutionary algorithms for solving flexible job

shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4):904–916, 2019. ISSN 23299274. doi: 10.1109/JAS.2019.1911540.

Mitsuo Gen and Lin Lin. Multiobjective evolutionary algorithm for manufacturing scheduling problems: State-of-the-art survey. *Journal of Intelligent Manufacturing*, 25(5):849–866, 2014. ISSN 15728145. doi: 10.1007/s10845-013-0804-4.

M. Gholami and M. Zandieh. Integrating simulation and genetic algorithm to schedule a dynamic flexible job shop. *Journal of Intelligent Manufacturing*, 20(4):481–498, 2009. ISSN 09565515. doi: 10.1007/s10845-008-0150-0.

Dunwei Gong, Yuyan Han, and Jianyong Sun. A novel hybrid multi-objective artificial bee colony algorithm for blocking lot-streaming flow shop scheduling problems. *Knowledge-Based Systems*, 148:115–130, may 2018. ISSN 09507051. doi: 10.1016/j.knosys.2018.02.029. URL https://linkinghub.elsevier.com/retrieve/pii/S0950705118300923.

Kairon Freitas Guimarães and Márcia Aparecida Fernandes. An approach for flexible Job-Shop Scheduling with separable sequence-dependent setup time. *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 5: 3727–3731, 2006. ISSN 1062922X. doi: 10.1109/ICSMC.2006.384709.

Xiuping Guo and Deming Lei. Bi-objective job shop scheduling with outsourcing options. *International Journal of Production Research*, 52(13):3832–3841, jul 2014. ISSN 0020-7543. doi: 10.1080/00207543.2013.848488. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2013.848488.

M. Hajibabaei and J. Behnamian. Flexible job-shop scheduling problem with unrelated parallel machines and resources-dependent processing times: a tabu search algorithm. *International Journal of Management Science and Engineering Management*, 16(4): 242–253, oct 2021. ISSN 1750-9653. doi: 10.1080/17509653.2021.1941368. URL https://www.tandfonline.com/doi/full/10.1080/17509653.2021.1941368.

Coşkun Hamzaçebi. Improving genetic algorithms' performance by local search for continuous function optimization. *Applied Mathematics and Computation*, 196(1): 309–317, feb 2008. ISSN 00963003. doi: 10.1016/j.amc.2007.05.068. URL https://linkinghub.elsevier.com/retrieve/pii/S009630030700673X.

Yu-Yan Han, Dun-wei Gong, Xiao-Yan Sun, and Quan-Ke Pan. An improved NSGA-II algorithm for multi-objective lot-streaming flow shop scheduling problem. *International Journal of Production Research*, 52(8):2211–2231, apr 2014. ISSN 0020-7543. doi: 10.1080/00207543.2013.848492. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2013.848492.

Nhu Binh Ho, Joc Cing Tay, and Edmund M.-K. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research*, 179(2):316–333, jun 2007. ISSN 03772217. doi: 10.1016/j.ejor.2006.04.007. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221706002219.

Tzung Pei Hong, Hong Shung Wang, and Wei Chou Chen. Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, 6(4):439–455, 2000. ISSN 13811231. doi: 10.1023/A:1009642825198.

PHILIP Y. HUANG. A comparative study of priority dispatching rules in a hybrid assembly/job shop. *International Journal of Production Research*, 22(3):375–387, may 1984. ISSN 0020-7543. doi: 10.1080/00207548408942460. URL http://www.tandfonline.com/doi/abs/10.1080/00207548408942460.

Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, 15(4):205–215, 1994. ISSN 01716468. doi: 10.1007/BF01719451.

Kenichi Ida and Kensaku Oka. Flexible job-shop scheduling problem by genetic algorithm. *Electrical Engineering in Japan (English translation of Denki Gakkai Ronbunshi)*, 177(3):28–35, 2011. ISSN 04247760. doi: 10.1002/eej.21194.

Aya Ishigaki and Shun Takaki. Iterated Local Search Algorithm for Flexible Job Shop Scheduling. *Proceedings - 2017 6th IIAI International Congress on Advanced Applied Informatics, IIAI-AAI 2017*, pages 947–952, 2017. doi: 10.1109/IIAI-AAI.2017.126.

Shudai Ishikawa, Ryosuke Kubota, and Keiichi Horio. Effective hierarchical optimization by a hierarchical multi-space competitive genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 42(24):9434–9440, 2015. ISSN 09574174. doi: 10.1016/j.eswa.2015.08.003.

Shuai Jia and Zhi Hua Hu. Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem. *Computers and Operations Research*, 47:11–26, 2014. ISSN 03050548. doi: 10.1016/j.cor.2014.01.010. URL http://dx.doi.org/10.1016/j.cor.2014.01.010.

Jinghui Zhong, Xiaomin Hu, Jun Zhang, and Min Gu. Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 2, pages 1115–1121. IEEE, 2005. ISBN 0-7695-2504-0. doi: 10.1109/CIMCA.2005.1631619. URL http://ieeexplore.ieee.org/document/1631619/.

S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, mar 1954. ISSN 00281441. doi: 10.1002/nav.3800010110. URL https://onlinelibrary.wiley.com/doi/10.1002/nav.3800010110.

Imed Kacem. Genetic algorithm for the flexible job-shop scheduling problem. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4 of *In the Proceedings of the IEEE International Conference on Systems, Man,*

and Cybernetics, pages 3464–3469, Washington, DC, 2003. IEEE. ISBN 0-7803-7952-7. doi: 10.1109/icsmc.2003.1244425. URL http://ieeexplore.ieee.org/document/1244425/.

Imed Kacem, Slim Hammadi, and Pierre Borne. Approach by localization and genetic manipulation algorithm for flexible job-shop scheduling problem. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 4 of *IEEE International Conference on Systems, Man and Cybernetics*, pages 2599–2604, Tucson, AZ, USA, 2001. doi: 10.1109/icsmc.2001.972955.

Voratas Kachitvichyanukul and Siriwan Sitthitham. A two-stage genetic algorithm for multi-objective job shop scheduling problems. *Journal of Intelligent Manufacturing*, 22(3):355–365, 2011. ISSN 09565515. doi: 10.1007/s10845-009-0294-6.

Jan Kelbel and Zdeněk Hanzálek. Solving production scheduling with earliness/tardiness penalties by constraint programming. *Journal of Intelligent Manufacturing*, 22(4): 553–562, aug 2011. ISSN 0956-5515. doi: 10.1007/s10845-009-0318-2. URL http://link.springer.com/10.1007/s10845-009-0318-2.

J-S. Kim, S-H. Kang, and S.M. Lee. Transfer batch scheduling for a two-stage flow-shop with identical parallel machines at each stage. *Omega*, 25(5):547–555, oct 1997. ISSN 03050483. doi: 10.1016/S0305-0483(97)00015-7. URL https://linkinghub.elsevier.com/retrieve/pii/S0305048397000157.

Jonas Krause, Jelson Cordeiro, Rafael Stubs Parpinelli, and Heitor Silvério A. Lopes. A Survey of Swarm Algorithms Applied to Discrete Optimization Problems. In *Swarm Intelligence and Bio-Inspired Computation*, pages 169–191. Elsevier, 2013. ISBN 9780124051638. doi: 10.1016/B978-0-12-405163-8.00007-7. URL https://linkinghub.elsevier.com/retrieve/pii/B9780124051638000077.

Dominik Kress, David Müller, and Jenny Nossack. A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, 41

(1):179–217, 2019. ISSN 14366304. doi: 10.1007/s00291-018-0537-z. URL https://doi.org/10.1007/s00291-018-0537-z.

Nilsen Kundakcı and Osman Kulak. Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem. *Computers and Industrial Engineering*, 96:31–51, jun 2016. ISSN 03608352. doi: 10.1016/j.cie.2016.03.011. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835216300742.

K K Kusuma and A Maruf. Job shop scheduling model for non-identic machine with fixed delivery time to minimize tardiness. *IOP Conference Series: Materials Science and Engineering*, 114:012062, feb 2016. ISSN 1757-8981. doi: 10.1088/1757-899X/114/1/012062. URL https://iopscience.iop.org/article/10.1088/1757-899X/114/1/012062.

Vincent Lal and C. Anand Deva Durai. A Survey on Various Optimization Techniques with Respect to Flexible Job Shop Scheduling. *International Journal of Scientific and Research Publications*, 4(2):1–7, 2014. URL http://www.ijsrp.org/research-paper-0214.php?rp=P262299.

V. Lauff and F. Werner. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modelling*, 40(5-6):637–655, sep 2004. ISSN 08957177. doi: 10.1016/j.mcm.2003.05.019. URL https://linkinghub.elsevier.com/retrieve/pii/S0895717704803285.

Deming Lei. A genetic algorithm for flexible job shop scheduling with fuzzy processing time. *International Journal of Production Research*, 48(10):2995–3013, may 2010. ISSN 0020-7543. doi: 10.1080/00207540902814348. URL https://www.tandfonline.com/doi/full/10.1080/00207540902814348.

Deming Lei. Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Applied Soft Computing Journal*, 12(8):2237–2245, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2012.03.025.

Deming Lei and Xiuping Guo. Scheduling job shop with lot streaming and transportation through a modified artificial bee colony. *International Journal of Production Research*, 51(16):4930–4941, aug 2013. ISSN 00207543. doi: 10.1080/00207543.2013.784404. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2013.784404.

Deming Lei and Xiuping Guo. A shuffled frog-leaping algorithm for job shop scheduling with outsourcing options. *International Journal of Production Research*, 54(16):4793–4804, aug 2016. ISSN 0020-7543. doi: 10.1080/00207543.2015.1088970. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2015.1088970.

Kun Lei, Peng Guo, Yi Wang, Jianyu Xiong, and Wenchao Zhao. An End-to-end Hierarchical Reinforcement Learning Framework for Large-scale Dynamic Flexible Job-shop Scheduling Problem. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, jul 2022. ISBN 978-1-7281-8671-9. doi: 10.1109/IJCNN55064.2022.9892005. URL https://ieeexplore.ieee.org/document/9892005/.

Hui Li, Xi Wang, and Jianbiao Peng. A hybrid differential evolution algorithm for flexible job shop scheduling with outsourcing operations and job priority constraints. *Expert Systems with Applications*, 201:117182, sep 2022a. ISSN 09574174. doi: 10.1016/j.eswa.2022.117182. URL https://linkinghub.elsevier.com/retrieve/pii/S0957417422005693.

Jun-Qing Li, Quan-Ke Pan, P. N. Suganthan, and T. J. Chua. A hybrid tabu search algorithm with an efficient neighborhood structure for the flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 52(5-8):683–697, feb 2011. ISSN 0268-3768. doi: 10.1007/s00170-010-2743-y. URL http://link.springer.com/10.1007/s00170-010-2743-y.

Lele Li. Research on discrete intelligent workshop lot-streaming scheduling with variable sublots under engineer to order. *Computers and Industrial Engineering*, 165:

107928, mar 2022. ISSN 03608352. doi: 10.1016/j.cie.2021.107928. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835221008329.

Nailiang Li and Caihong Feng. Research on Machining Workshop Batch Scheduling Incorporating the Completion Time and Non-Processing Energy Consumption Considering Product Structure. *Energies*, 14(19):6079, sep 2021. ISSN 1996-1073. doi: 10.3390/en14196079. URL https://www.mdpi.com/1996-1073/14/19/6079.

Xinyu Li and Liang Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174:93–110, 2016. ISSN 09255273. doi: 10.1016/j.ijpe.2016.01.016. URL http://dx.doi.org/10.1016/j.ijpe.2016.01.016.

Xiulin Li, Jiansha Lu, Chenxi Yang, and Jiale Wang. Research of Flexible Assembly Job-Shop Batch–Scheduling Problem Based on Improved Artificial Bee Colony. *Frontiers in Bioengineering and Biotechnology*, 10, aug 2022b. ISSN 2296-4185. doi: 10.3389/fbioe.2022.909548. URL https://www.frontiersin.org/articles/10.3389/fbioe.2022.909548/full.

Yali Lin and Peng Zhang. Improved Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem with Machine Deterioration Effect. *Proceedings of IEEE 7th International Conference on Computer Science and Network Technology, ICCSNT 2019*, pages 131–134, 2019. doi: 10.1109/ICCSNT47585.2019.8962439.

C. H. Liu. Lot streaming for customer order scheduling problem in job shop environments. *International Journal of Computer Integrated Manufacturing*, 22(9):890–907, sep 2009. ISSN 13623052. doi: 10.1080/09511920902866104. URL http://www.tandfonline.com/doi/abs/10.1080/09511920902866104.

Changchun Liu, Xi Xiang, Li Zheng, and Jing Ma. An integrated model for multi-resource constrained scheduling problem considering multi-product and resource-sharing. *International Journal of Production Research*, 56(19):6491–6511, oct

2018. ISSN 0020-7543. doi: 10.1080/00207543.2017.1363428. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2017.1363428.

Cheng-Hsiang Liu and Cheng-I Hsu. Dynamic job shop scheduling with fixed interval deliveries. *Production Engineering*, 9(3):377–391, aug 2015. ISSN 0944-6524. doi: 10.1007/s11740-015-0605-z. URL http://link.springer.com/10.1007/s11740-015-0605-z.

Cheng Hsiang Liu, Long Sheng Chen, and Pei Shiun Lin. Lot streaming multiple jobs with values exponentially deteriorating over time in a job-shop environment. *International Journal of Production Research*, 51(1):202–214, jan 2013. ISSN 00207543. doi: 10.1080/00207543.2012.657255. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2012.657255.

Jiyin Liu. Single-job lot streaming in m - 1 two-stage hybrid flowshops. *European Journal of Operational Research*, 187(3):1171–1183, jun 2008. ISSN 03772217. doi: 10.1016/j.ejor.2006.06.066. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221706008290.

Jia Luo and Didier El Baz. A survey on parallel genetic algorithms for shop scheduling problems. In *Proceedings - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018*, pages 629–636. IEEE, may 2018. ISBN 9781538655559. doi: 10.1109/IPDPSW.2018.00103. URL https://ieeexplore.ieee.org/document/8425470/.

Jia Luo, Didier El Baz, Rui Xue, and Jinglu Hu. Solving the dynamic energy aware job shop scheduling problem with the heterogeneous parallel genetic algorithm. *Future Generation Computer Systems*, 108:119–134, 2020a. ISSN 0167739X. doi: 10.1016/j.future.2020.02.019. URL https://doi.org/10.1016/j.future.2020.02.019.

Xiong Luo, Qian Qian, and Yun Fa Fu. Improved genetic algorithm for solving flexible job shop scheduling problem. *Procedia Computer Science*, 166:480–485, 2020b.

ISSN 18770509. doi: 10.1016/j.procs.2020.02.061. URL https://doi.org/10.1016/j.procs.2020.02.061.

Jia Ma, Gang Shi, Li Qun Gao, and Dan Li. Adaptive immune genetic algorithm for flexible job-shop scheduling problem. *Xitong Fangzhen Xuebao / Journal of System Simulation*, 21(12):3609–3613, 2009. ISSN 1004731X.

Bart MacCarthy and Jiyin Liu. Addressing the gap in scheduling research: a review of optimization and heuristic methods in production scheduling. *International Journal of Production Research*, 31(1):59–79, jan 1993. ISSN 0020-7543. doi: 10.1080/00207549308956713. URL http://www.tandfonline.com/doi/abs/10.1080/00207549308956713.

S. Afshin Mansouri. A Multi-Objective Genetic Algorithm for mixed-model sequencing on JIT assembly lines. *European Journal of Operational Research*, 167(3):696–716, dec 2005. ISSN 03772217. doi: 10.1016/j.ejor.2004.07.016. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221704004734.

Monaldo Mastrolilli and Luca Maria Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20, jan 2000. ISSN 10946136. doi: 10.1002/(sici)1099-1425(200001/02)3:1⟨3::aid-jos32⟩3.0.co;2-y.

M. McIlhagga. Solving generic scheduling problems with a distributed genetic algorithm. In *Lecture Notes in Computer Science, vol 1305*, pages 199–212. Springer, Berlin, Heidelberg, 2005. doi: 10.1007/BFb0027175. URL http://link.springer.com/10.1007/BFb0027175.

Gonzalo Mejía, Carlos Montoya, Julián Cardona, and Ana Lucía Castro. Petri nets and genetic algorithms for complex manufacturing systems scheduling. *International Journal of Production Research*, 50(3):791–803, feb 2012. ISSN 0020-7543. doi: 10.1080/00207543.2010.543177. URL http://www.tandfonline.com/doi/abs/10.1080/00207543.2010.543177.

Leilei Meng, Chaoyong Zhang, Xinyu Shao, and Yaping Ren. MILP models for energy-aware flexible job shop scheduling problem. *Journal of Cleaner Production*, 210:710–723, 2019. ISSN 09596526. doi: 10.1016/j.jclepro.2018.11.021. URL https://doi.org/10.1016/j.jclepro.2018.11.021.

Tao Meng, Quan-Ke Pan, Jun-Qing Li, and Hong-Yan Sang. An improved migrating birds optimization for an integrated lot-streaming flow shop scheduling problem. *Swarm and Evolutionary Computation*, 38:64–78, feb 2018a. ISSN 22106502. doi: 10.1016/j.swevo.2017.06.003. URL https://linkinghub.elsevier.com/retrieve/pii/S2210650216304965.

Tao Meng, Quan Ke Pan, and Hong Yan Sang. A hybrid artificial bee colony algorithm for a flexible job shop scheduling problem with overlapping in operations. *International Journal of Production Research*, 56(16):5278–5292, aug 2018b. ISSN 1366588X. doi: 10.1080/00207543.2018.1467575. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2018.1467575.

K. Mesghouni, S. Hammadi, and P. Borne. Evolution programs for job-shop scheduling. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, volume 1 of *IEEE International Conference on Systems, Man, and Cybernetics: Computational Cybernetics and Simulation*, pages 720–725, Orlando, FL, USA, 1997. doi: 10.1109/icsmc.1997.625839.

Kostas S. Metaxiotis, John E. Psarras, and Dimitris Askounis. Building ontologies for production scheduling systems: Towards a unified methodology. *Information Management and Computer Security*, 9(1):44–50, mar 2001. ISSN 09685227. doi: 10.1108/09685220110366803. URL https://www.emerald.com/insight/content/doi/10.1108/09685220110366803/full/html.

Kostas S. Metaxiotis, John E. Psarras, and Kostas A. Ergazakis. Production scheduling in ERP systems: An AI-based approach to face the gap. *Business Process Management Journal*, 9(2):221–247, apr 2003. ISSN 14637154. doi: 10.1108/

14637150310468416. URL https://www.emerald.com/insight/content/doi/10.1108/14637150310468416/full/html.

B. Miller and D.E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1996. URL https://wpmedia.wolfram.com/uploads/sites/13/2018/02/09-3-2.pdf.

Ming Huang, Jia Gu, Xu Liang, and Yue Guan. Research on assembly constraints job shop scheduling based on genetic algorithm. In *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, pages 708–712. IEEE, dec 2015. ISBN 978-1-4673-8173-4. doi: 10.1109/ICCSNT.2015.7490842. URL http://ieeexplore.ieee.org/document/7490842/.

Hadi Mokhtari and Aliakbar Hasani. An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. *Computers and Chemical Engineering*, 104: 339–352, sep 2017. ISSN 00981354. doi: 10.1016/j.compchemeng.2017.05.004. URL https://linkinghub.elsevier.com/retrieve/pii/S009813541730203X.

David Montana. How to Make Scheduling Research Relevant. In *GECCO-2002 Workshop: Scheduling: Bringing Together Theory and Practice*, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.194.7766&rep=rep1&type=pdf.

M. Mousakhani. Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51(12): 3476–3487, 2013. ISSN 00207543. doi: 10.1080/00207543.2012.746480.

Hongbum Na and Jinwoo Park. Multi-level job scheduling in a flexible job shop environment. *International Journal of Production Research*, 52(13):3877–3887, 2014. ISSN 1366588X. doi: 10.1080/00207543.2013.848487. URL http://dx.doi.org/10.1080/00207543.2013.848487.

Mohsen Nejati, Iraj Mahdavi, Reza Hassanzadeh, Nezam Mahdavi-Amiri, and Mohamadsailm Mojarad. Multi-job lot streaming to minimize the weighted completion

time in a hybrid flow shop scheduling problem with work shift constraint. *International Journal of Advanced Manufacturing Technology*, 70(1-4):501–514, jan 2014. ISSN 14333015. doi: 10.1007/s00170-013-5265-6. URL http://link.springer.com/10.1007/s00170-013-5265-6.

Juan M. Novas. Production scheduling and lot streaming at flexible job-shops environments using constraint programming. *Computers and Industrial Engineering*, 136 (February 2018):252–264, 2019. ISSN 03608352. doi: 10.1016/j.cie.2019.07.011. URL https://doi.org/10.1016/j.cie.2019.07.011.

Franz Oppacher and Mark Wineberg. The shifting balance genetic algorithm: Improving the GA in a dynamic environment. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 504–510, San Francisco, CA, USA, 1999.

Cemal Özgüven, Yasemin Yavuz, and Lale Özbakir. Mixed integer goal programming models for the flexible job-shop scheduling problems with separable and non-separable sequence dependent setup times. *Applied Mathematical Modelling*, 36(2):846–858, 2012. ISSN 0307904X. doi: 10.1016/j.apm.2011.07.037.

Juan José Palacios, Miguel A. González, Camino R. Vela, Inés González-Rodríguez, and Jorge Puente. Genetic tabu search for the fuzzy flexible job shop problem. *Computers and Operations Research*, 54:74–89, feb 2015. ISSN 03050548. doi: 10.1016/j.cor.2014.08.023. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054814002330.

S. S. Panwalkar, R. A. Dudek, and M. L. Smith. Sequencing Research and the Industrial Scheduling Problem. In *Symposium on the theory of scheduling and its applications, In S. E. Elmaghraby (Ed.)*, pages 29–38, Springer-Verlag, 1973. doi: 10.1007/978-3-642-80784-8_2.

F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers and Operations Research*, 35(10):3202–3212, oct

2008. ISSN 03050548. doi: 10.1016/j.cor.2007.02.014. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054807000524.

Jun qing Li, Xin rui Tao, Bao xian Jia, Yu yan Han, Chuang Liu, Peng Duan, Zhi xin Zheng, and Hong yan Sang. Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots. *Swarm and Evolutionary Computation*, 52: 100600, feb 2020. ISSN 22106502. doi: 10.1016/j.swevo.2019.100600. URL https://linkinghub.elsevier.com/retrieve/pii/S2210650219300392.

Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M.A. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Applications*, 53(10):1605–1614, 2007. ISSN 08981221. doi: 10.1016/j.camwa.2006.07.013.

M. Rajkumar, P. Asokan, N. Anilkumar, and T. Page. A GRASP algorithm for flexible job-shop scheduling problem with limited resource constraints. *International Journal of Production Research*, 49(8):2409–2423, 2011. ISSN 00207543. doi: 10.1080/00207541003709544.

A. Reisman, A. Kumar, and J. Motwani. Flowshop scheduling/sequencing research: a statistical review of the literature, 1952-1994. *IEEE Transactions on Engineering Management*, 44(3):316–329, 1997. ISSN 00189391. doi: 10.1109/17.618173. URL http://ieeexplore.ieee.org/document/618173/.

S Reiter. A system for managing job shop production. *Journal of Business*, 34:371–393, 1966.

Mohamad Rohaninejad, Amirsaman Kheirkhah, Parviz Fattahi, and Behdin Vahedi-Nouri. A hybrid multi-objective genetic algorithm based on the ELECTRE method for a capacitated flexible job shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 77(1-4):51–66, 2015. ISSN 14333015. doi: 10.1007/s00170-014-6415-1.

Gustavo Alencar Rolim and Marcelo Seido Nagano. Structural properties and algorithms for earliness and tardiness scheduling against common due dates and windows: A review. *Computers and Industrial Engineering*, 149:106803, nov 2020. ISSN 03608352. doi: 10.1016/j.cie.2020.106803. URL https://linkinghub.elsevier.com/retrieve/pii/S036083522030509X.

Danial Rooyani and Fantahun Defersha. A Two-Stage Multi-Objective Genetic Algorithm for a Flexible Job Shop Scheduling Problem with Lot Streaming. *Algorithms*, 15(7):246, jul 2022. ISSN 1999-4893. doi: 10.3390/a15070246. URL https://www.mdpi.com/1999-4893/15/7/246.

Danial Rooyani and Fantahun M. Defersha. An Efficient Two-Stage Genetic Algorithm for Flexible Job-Shop Scheduling. *IFAC-PapersOnLine*, 52(13):2519–2524, 2019. ISSN 24058963. doi: 10.1016/j.ifacol.2019.11.585. URL https://linkinghub.elsevier.com/retrieve/pii/S2405896319315721.

Andrea Rossi. Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153:253–267, 2014. ISSN 09255273. doi: 10.1016/j.ijpe.2014.03.006. URL http://dx.doi.org/10.1016/j.ijpe.2014.03.006.

Andrea Rossi and Gino Dini. Flexible job-shop scheduling with routing flexibility and separable setup times using ant colony optimisation method. *Robotics and Computer-Integrated Manufacturing*, 23(5):503–516, 2007. ISSN 07365845. doi: 10.1016/j.rcim.2006.06.004.

Norman Sadeh. MICRO-BOSS: A micro-opportunistic factory scheduler. *Expert Systems with Applications*, 6(3):377–392, jul 1993. ISSN 09574174. doi: 10.1016/0957-4174(93)90062-B. URL https://linkinghub.elsevier.com/retrieve/pii/095741749390062B.

Norman Sadeh and Mark S. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86(1):

1–41, sep 1996. ISSN 00043702. doi: 10.1016/0004-3702(95)00098-4. URL https://linkinghub.elsevier.com/retrieve/pii/0004370295000984.

Hamid Safarzadeh and Farhad Kianfar. Job shop scheduling with the option of jobs outsourcing. *International Journal of Production Research*, 57(10):3255–3272, may 2019. ISSN 0020-7543. doi: 10.1080/00207543.2019.1579934. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2019.1579934.

Mohammad Saidi-Mehrabad and Parviz Fattahi. Flexible job shop scheduling with tabu search algorithms. *International Journal of Advanced Manufacturing Technology*, 32 (5-6):563–570, 2007. ISSN 02683768. doi: 10.1007/s00170-005-0375-4.

Masatoshi Sakawa and Ryo Kubota. Fuzzy programming for multiobjective job shop scheduling with fuzzy processing time and fuzzy duedate through genetic algorithms. *European Journal of Operational Research*, 120(2):393–407, jan 2000. ISSN 03772217. doi: 10.1016/S0377-2217(99)00094-6. URL https://linkinghub.elsevier.com/retrieve/pii/S0377221799000946.

Hong Yan Sang and Jun Hua Duan. An efficient discrete artificial bee colony algorithm for total flowtime lot-streaming flowshop. In *Proceedings - 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2012*, pages 1585–1588. IEEE, may 2012. ISBN 9781467300223. doi: 10.1109/FSKD.2012.6234319. URL http://ieeexplore.ieee.org/document/6234319/.

I. V. Sergienko, L. F. Hulianytskyi, and S. I. Sirenko. Classification of applied methods of combinatorial optimization. *Cybernetics and Systems Analysis*, 45(5):732–741, 2009. ISSN 10600396. doi: 10.1007/s10559-009-9134-0.

Melissa Shahgholi Zadeh, Yalda Katebi, and Ali Doniavi. A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. *International Journal of Production Research*, 57(10):3020–3035, may 2019. ISSN 1366588X. doi: 10.1080/00207543.2018.1524165. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2018.1524165.

Liji Shen, Stéphane Dauzère-Pérès, and Janis S. Neufeld. Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2):503–516, 2018. ISSN 03772217. doi: 10.1016/j.ejor.2017.08.021.

X. Shen, M. Zhang, and J. Fu. Multi-Objective dynamic job shop scheduling: A survey and prospects. *International Journal of Innovative Computing, Information and Control*, 10(6):2113–2126, 2014. URL http://www.ijicic.org/ijicic-14-01022.pdf.

Xiao Qiu Shi, Wei Long, Yan Yan Li, Yong Lai Wei, and Ding Shan Deng. Different Performances of Different Intelligent Algorithms for Solving FJSP: A Perspective of Structure. *Computational Intelligence and Neuroscience*, 2018(doi:10.1155/2018/4617816), 2018. ISSN 16875273. doi: 10.1155/2018/4617816.

Gaurav Singh and Rene Weiskircher. A multi-agent system for decentralised fractional shared resource constraint scheduling. *Web Intelligence and Agent Systems: An International Journal*, 9(2):99–108, 2011. ISSN 15701263. doi: 10.3233/WIA-2011-0208. URL https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/WIA-2011-0208.

Stephen Smith and Marcel Becker. An Ontology for Constructing Scheduling Systems. In *Proceedings of AAAI '97 Spring Symposium on Ontological Engineering*, pages 120 – 129. AAAI Press, 1997. URL https://www.aaai.org/Papers/Symposia/Spring/1997/SS-97-06/SS97-06-016.pdf.

Saisumpan Sooncharoen, Pupong Pongcharoen, and Christian Hicks. Grey Wolf production scheduling for the capital goods industry. *Applied Soft Computing*, 94:106480, sep 2020. ISSN 15684946. doi: 10.1016/j.asoc.2020.106480. URL https://linkinghub.elsevier.com/retrieve/pii/S1568494620304191.

Junpeng Su, Han Huang, Gang Li, Xueqiang Li, and Zhifeng Hao. Self-Organizing Neural Scheduler for the Flexible Job Shop Problem With Periodic Maintenance and Mandatory Outsourcing Constraints. *IEEE Transactions on Cybernetics*, pages

1–12, 2022. ISSN 2168-2267. doi: 10.1109/TCYB.2022.3158334. URL https://ieeexplore.ieee.org/document/9750014/.

Jinghe Sun, Guohui Zhang, Jiao Lu, and Wenqiang Zhang. A hybrid many-objective evolutionary algorithm for flexible job-shop scheduling problem with transportation and setup times. *Computers and Operations Research*, 132:105263, aug 2021. ISSN 03050548. doi: 10.1016/j.cor.2021.105263. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054821000551.

Wei Sun, Ying Pan, Xiaohong Lu, and Qinyi Ma. Research on flexible job-shop scheduling problem based on a modified genetic algorithm. *Journal of Mechanical Science and Technology*, 24(10):2119–2125, 2010. ISSN 1738494X. doi: 10.1007/s12206-010-0526-x.

A Tamilarasi and T Anantha kumar. An enhanced genetic algorithm with simulated annealing for job-shop scheduling. *International Journal of Engineering, Science and Technology*, 2(1):144–151, 2010. ISSN 2141-2820. doi: 10.4314/ijest.v2i1.59105.

Jianchao Tang, Guoji Zhang, Binbin Lin, and Bixi Zhang. A hybrid algorithm for flexible job-shop scheduling problem. *Procedia Engineering*, 15:3678–3683, 2011. ISSN 18777058. doi: 10.1016/j.proeng.2011.08.689. URL https://linkinghub.elsevier.com/retrieve/pii/S1877705811021904.

Charles E. Taylor. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Complex Adaptive Systems. John H. Holland* , volume 69. University of Michigan Press, Ann Arbor, MI, 1994. doi: 10.1086/418447.

Wannaporn Teekeng and Arit Thammano. Modified genetic algorithm for flexible job-shop scheduling problems. *Procedia Computer Science*, 12:122–128, 2012. ISSN 18770509. doi: 10.1016/j.procs.2012.09.041.

H. Tsubone, M. Ohba, and T. Uetake. The impact of lot sizing and sequencing on

manufacturing performance in a two-stage hybrid flow shop. *International Journal of Production Research*, 34(11):3037–3053, nov 1996. ISSN 1366588X. doi: 10.1080/00207549608905076. URL http://www.tandfonline.com/doi/abs/10.1080/00207549608905076.

José A. Ventura and Suk Hun Yoon. A new genetic algorithm for lot-streaming flow shop scheduling with limited capacity buffers. *Journal of Intelligent Manufacturing*, 24(6):1185–1196, 2013. ISSN 09565515. doi: 10.1007/s10845-012-0650-9.

Ling Wang, Gang Zhou, Ye Xu, Shengyao Wang, and Min Liu. An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 60(1-4):303–315, apr 2012. ISSN 0268-3768. doi: 10.1007/s00170-011-3610-1. URL http://link.springer.com/10.1007/s00170-011-3610-1.

Ping Wang, Hongyan Sang, Qiuyun Tao, Hengwei Guo, Junqing Li, Kaizhou Gao, and Yuyan Han. Improved Migrating Birds Optimization Algorithm to Solve Hybrid Flowshop Scheduling Problem with Lot-Streaming. *IEEE Access*, 8:89782–89792, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2993881. URL https://ieeexplore.ieee.org/document/9091160/.

Shasha Wang, Mary Kurz, Scott Jennings Mason, and Eghbal Rashidi. Two-stage hybrid flow shop batching and lot streaming with variable sublots and sequence-dependent setups. *International Journal of Production Research*, 57(22):6893–6907, nov 2019. ISSN 1366588X. doi: 10.1080/00207543.2019.1571251. URL https://www.tandfonline.com/doi/full/10.1080/00207543.2019.1571251.

Wenyan Wang, Zhenhao Xu, and Xingsheng Gu. A two-stage discrete water wave optimization algorithm for the flowshop lot-streaming scheduling problem with intermingling and variable lot sizes. *Knowledge-Based Systems*, 238:107874, feb 2022. ISSN 09507051. doi: 10.1016/j.knosys.2021.107874. URL https://linkinghub.elsevier.com/retrieve/pii/S0950705121010480.

Xiaojuan Wang, Liang Gao, Chaoyong Zhang, and Xinyu Shao. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. *International Journal of Advanced Manufacturing Technology*, 51(5-8): 757–767, 2010. ISSN 02683768. doi: 10.1007/s00170-010-2642-2.

Yong Ming Wang, Nan Feng Xiao, Hong Li Yin, En Liang Hu, Cheng Gui Zhao, and Yan Rong Jiang. A two-stage genetic algorithm for large size job shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 39(7-8):813–820, 2008. ISSN 02683768. doi: 10.1007/s00170-007-1260-0.

Hongjing Wei, Shaobo Li, Huafeng Quan, Dacheng Liu, Shu Rao, Chuanjiang Li, and Jianjun Hu. Unified Multi-Objective Genetic Algorithm for Energy Efficient Job Shop Scheduling. *IEEE Access*, 9:54542–54557, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3070981. URL https://ieeexplore.ieee.org/document/9395108/.

Vincent C.S. Wiers and Tjerk W. van der Schaaf. A framework for decision support in production scheduling tasks. *Production Planning and Control*, 8(6):533–544, jan 1997. ISSN 13665871. doi: 10.1080/095372897234876. URL https://www.tandfonline.com/doi/full/10.1080/095372897234876.

T. C. Wong, Felix T.S. Chan, and L. Y. Chan. A resource-constrained assembly job shop scheduling problem with Lot Streaming technique. *Computers and Industrial Engineering*, 57(3):983–995, oct 2009. ISSN 03608352. doi: 10.1016/j.cie.2009.04.002. URL https://linkinghub.elsevier.com/retrieve/pii/S036083520900103X.

World Bank. Manufacturing, value added (% of GDP), 2020. URL https://data.worldbank.org/indicator/NV.IND.MANF.ZS.

D B Wortman. Managing capacity: Getting the most from your companys asset. *Industrial Engineering*, 24:47–49, 1992.

Jin Xie, Liang Gao, Kunkun Peng, Xinyu Li, and Haoran Li. Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing*, 1(3):67–77, sep 2019.

ISSN 2516-8398. doi: 10.1049/iet-cim.2018.0009. URL https://digital-library.theiet.org/content/journals/10.1049/iet-cim.2018.0009.

Li-Ning Xing, Ying-Wu Chen, and Ke-Wei Yang. An efficient search method for multi-objective flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 20(3):283–293, jun 2009. ISSN 0956-5515. doi: 10.1007/s10845-008-0216-z. URL http://link.springer.com/10.1007/s10845-008-0216-z.

Wenxiang Xu, Yongwen Hu, Wei Luo, Lei Wang, and Rui Wu. A multi-objective scheduling method for distributed and flexible job shop based on hybrid genetic algorithm and tabu search considering operation outsourcing and carbon emission. *Computers and Industrial Engineering*, 157:107318, jul 2021. ISSN 03608352. doi: 10.1016/j.cie.2021.107318. URL https://linkinghub.elsevier.com/retrieve/pii/S0360835221002229.

Guisen Xue, O. Felix Offodile, Hong Zhou, and Marvin D. Troutt. Integrated production planning with sequence-dependent family setup times. *International Journal of Production Economics*, 131(2):674–681, jun 2011. ISSN 09255273. doi: 10.1016/j.ijpe.2011.02.012. URL https://linkinghub.elsevier.com/retrieve/pii/S0925527311000697.

Hong-an Yang, Qi-feng Sun, and Jie Guo. Genetic algorithm for Job Shop Scheduling with Earliness and Tardiness penalties. In *2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management*, pages 754–758. IEEE, oct 2010. ISBN 978-1-4244-6483-8. doi: 10.1109/ICIEEM.2010.5646512. URL http://ieeexplore.ieee.org/document/5646512/.

Hong-an Yang, Qi-feng Sun, Can Saygin, and Shu-dong Sun. Job shop scheduling based on earliness and tardiness penalties with due dates and deadlines: an enhanced genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, 61(5-8):657–666, jul 2012. ISSN 0268-3768. doi: 10.1007/s00170-011-3746-z. URL http://link.springer.com/10.1007/s00170-011-3746-z.

M. Yazdani, M. Amiri, and M. Zandieh. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37(1): 678–687, jan 2010. ISSN 09574174. doi: 10.1016/j.eswa.2009.06.007. URL https://linkinghub.elsevier.com/retrieve/pii/S0957417409005685.

Yuan Yuan and Hua Xu. An integrated search heuristic for large-scale flexible job shop scheduling problems. *Computers and Operations Research*, 40(12):2864–2877, dec 2013. ISSN 03050548. doi: 10.1016/j.cor.2013.06.010. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054813001676.

Yuan Yuan and Hua Xu. Multiobjective Flexible Job Shop Scheduling Using Memetic Algorithms. *IEEE Transactions on Automation Science and Engineering*, 12(1):336–353, jan 2015. ISSN 1545-5955. doi: 10.1109/TASE.2013.2274517. URL https://ieeexplore.ieee.org/document/6582693/.

Yuan Yuan, Hua Xu, and Jiadong Yang. A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing*, 13(7):3259–3272, jul 2013. ISSN 15684946. doi: 10.1016/j.asoc.2013.02.013. URL https://linkinghub.elsevier.com/retrieve/pii/S1568494613000677.

Gabriel Zambrano Rey, Abdelghani Bekrar, Damien Trentesaux, and Bing-Hai Zhou. Solving the flexible job-shop just-in-time scheduling problem with quadratic earliness and tardiness costs. *The International Journal of Advanced Manufacturing Technology*, 81(9-12):1871–1891, dec 2015. ISSN 0268-3768. doi: 10.1007/s00170-015-7347-0. URL http://link.springer.com/10.1007/s00170-015-7347-0.

Rim Zarrouk, Imed Eddine Bennour, and Abderrazek Jemai. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. *Swarm Intelligence*, 13(2):145–168, 2019. ISSN 19353820. doi: 10.1007/s11721-019-00167-w. URL https://doi.org/10.1007/s11721-019-00167-w.

Biao Zhang, Quan ke Pan, Liang Gao, Xin li Zhang, Hong yan Sang, and Jun qing Li. An effective modified migrating birds optimization for hybrid flowshop scheduling problem

with lot streaming. *Applied Soft Computing Journal*, 52:14–27, mar 2017. ISSN 15684946. doi: 10.1016/j.asoc.2016.12.021. URL https://linkinghub.elsevier.com/retrieve/pii/S1568494616306391.

Biao Zhang, Quan-ke Pan, Lei-lei Meng, Chao Lu, Jian-hui Mou, and Jun-qing Li. An automatic multi-objective evolutionary algorithm for the hybrid flowshop scheduling problem with consistent sublots. *Knowledge-Based Systems*, 238:107819, feb 2022. ISSN 09507051. doi: 10.1016/j.knosys.2021.107819. URL https://linkinghub.elsevier.com/retrieve/pii/S0950705121010133.

Guohui Zhang, Liang Gao, and Yang Shi. Expert Systems with Applications An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems With Applications*, 38(4):3563–3573, 2011. ISSN 0957-4174. URL http://dx.doi.org/10.1016/j.eswa.2010.08.145.

Guohui Zhang, Yifan Hu, Jinghe Sun, and Wenqiang Zhang. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation*, 54(March 2019):100664, 2020. ISSN 22106502. doi: 10.1016/j.swevo.2020.100664. URL https://doi.org/10.1016/j.swevo.2020.100664.

Haipeng Zhang and Mitsuo Gen. Multistage-based genetic algorithm for flexible job-shop scheduling problem. *Complexity International*, 11, 2005. ISSN 13200682. URL https://static.aminer.org/pdf/PDF/000/351/458/a_genetic_algorithm_for_flexible_job_shop_scheduling.pdf.

Hehua Zhang and Ming Gu. Modeling job shop scheduling with batches and setup times by timed Petri nets. *Mathematical and Computer Modelling*, 49(1-2):286–294, 2009. ISSN 08957177. doi: 10.1016/j.mcm.2008.03.010. URL http://dx.doi.org/10.1016/j.mcm.2008.03.010.

Wei Zhang, Changyu Yin, Jiyin Liu, and Richard J. Linn. Multi-job lot streaming to minimize the mean completion time in m-1 hybrid flowshops. *International Journal*

of Production Economics, 96(2):189–200, 2005. ISSN 09255273. doi: 10.1016/j.ijpe.2004.04.005.

Hongyang Zhong, Jianjun Liu, Qingxin Chen, Ning Mao, and Xiaojia Yang. Performance Assessment of Dynamic Flexible Assembly Job Shop Control Methods. *IEEE Access*, 8:226042–226058, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3043880. URL https://ieeexplore.ieee.org/document/9290137/.

Mohsen Ziaee. A heuristic algorithm for solving flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, 71(1-4):519–528, mar 2014. ISSN 0268-3768. doi: 10.1007/s00170-013-5510-z. URL http://link.springer.com/10.1007/s00170-013-5510-z.

Nozha Zribi and Pierre Borne. Hybrid genetic algorithm for the flexible job-shop problem under maintenance constraints. In *Lecture Notes in Computer Science*, volume 3612, pages 259–268. 2005. doi: 10.1007/11539902_31. URL http://link.springer.com/10.1007/11539902_31.

Nozha Zribi, Imed Kacem, Abdelkader El Kamel, and Pierre Borne. Assignment and Scheduling in Flexible Job-Shops by Hierarchical Optimization. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 37(4):652–661, jul 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2007.897494. URL http://ieeexplore.ieee.org/document/4252265/.