Linear Programming Assisted Genetic Algorithm for Solving a Comprehensive Job shop Lot Streaming Problem

by

Saber Bayat Movahed

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

LINEAR PROGRAMMING ASSISTED GENETIC ALGORITHM FOR SOLVING A COMPREHENSIVE JOB SHOP LOT STREAMING PROBLEM

Saber Bayat Movahed
University of Guelph, 2014

Advisor:
Professor F.M. Defersha

The hybridization of metaheuristics with other techniques for optimization has been one of the most interesting trends in recent years. The focus of research on metaheuristics has also notably shifted from an algorithm-oriented point of view to a problem-oriented one. Many researchers focus on solving a problem at hand as best as possible rather than promoting a certain metaheuristic. This has led researchers to try combining different algorithmic components in order to design algorithms that are more powerful than the ones resulting from the implementation of a pure metaheuristic. In this thesis, a linear programming assisted genetic algorithm is developed for solving a flexible job-shop scheduling problem with lot streaming. The genetic algorithm searches over both discrete and continuous variables in the problem/ solution space. Linear programming model is used to further refine promising solutions in the initial population and during the genetic search process by determining the optimal values of the continuous variables corresponding to the values of the integer variables of these promising solutions. Numerical examples showed that the hybridization of the genetic algorithm with the linear programming greatly improves its convergence behavior.

Keywords: *Flexible Job-shop Scheduling; Lot Streaming; Hybrid Genetic Algorithm; Linear Programming.*

Dedicated to my mother and father, Susan and Reza

ACKNOWLEDGEMENTS

First and foremost, I would like to express my most sincere gratitude and appreciation to my wonderful supervisor, Professor Dr. Fantahun Defersha, whose support, insight and patience have been invaluable throughout my time at the University of Guelph. I can not thank him enough for the time and effort he spent in teaching and guiding me through my research. Without his advice, guidance and encouragement this MASc thesis would not have been achievable. I also would like to extend my acknowledgement to my advisory committee member, Professor Dr. Soha Moussa, for reviewing my thesis and the knowledge I have gained from her lectures.

My most heartfelt thanks go to my beloved parents for their endless support and unwavering encouragement. I am greatly indebted to them for the countless sacrifices they have made for me throughout my life. I would also like to express my special thank to my brother Hanif who is the greatest indirect contributor to this thesis. He gave me tremendous support and encouragements during my stay in Canada. Also, I am deeply thankful to my brother Saeed for his continuous love and support despite the long distance between us.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$A_{o,j}$	A binary data equal to 1 if setup of operation o of a sublot of job j is attached (non-anticipatory), or 0 if this setup is detached (anticipatory)
B_j	The number of parts in a batch of job j
block	A system with limited buffer between the workstations
brkdwn	A system where machine breakdowns are considered
$b_{s,j}$	Size of sublot s of job j
btchj	A system where jobs are made in bulks, however, lot sizing is not permitted
btchls	A system where jobs are made in bulks, and lot sizing is permitted
btchm	A system where machines with ability of batch processing exit
chain	A system where precedence relation is in form of a chain
c_{max}	Makespan of the schedule
$c_{o,j}$	Completion time of operation o of job j
$c_{o,j,m}$	Completion time of operation o of job j on machine m
$c_{o,s,j,m}$	Completion time of operation o of sublot s of job j on machine m
$\widehat{c}_{r,m}$	Completion time of the r^{th} run of machine m
d	Degeneration limit
Dc	A system with continuous sublots
Dd	A system with discrete sublots
D_m	Machine m release date
E_m	A set of operation, which can be performed on machine m
Ffc	Flexible flow shop system

$FixN$	A system where number of sublots in each lot is predetermined
FJm	Flexible job shop system
$FlexN$	A system where number of sublots in each lot is not given
Fm	Flow shop system
j	Job index
J	Maximum number of jobs
Jm	Job shop system
L_1	Single product system
L_n	Multi products system
lag	A system where with there are time lags between successive operations of a job
L_{max}	Maximum lateness
$L_{o,j}$	The amount of time which operation o of a sublot of job j must wait (lag time) before being processed on eligible machine m
M	Maximum number of machines
m	Machine index
$M_{o,j}$	A set of eligible machines of operation o of job j
Ni	A system where machines are not allowed to have idle time.
nwt	A system where a particular job must never stop between two consecutive workstations
o	Operation index
O_j	Maximum number of operations of job j
Om	Open shop system
Pa	A system with detached setup time for certain machines
Pm	A system with identical parallel machines
Pn	A system with attached setup time for all machines

$P_{o,j,m}$	A binary data equal to 1 if operation o of job j can be processed on machine m, 0 otherwise
prec	A system where precedence relation is in form of an arbitrary acyclic graph
prmp	A system with preemption
Qm	A system with uniform parallel machines
$q_{o,j,m}$	Starting time of operation o of job j on machine m
$\hat{q}_{r,m}$	Starting time of production run r of machine m
r	Run index
r_j	A system with release date for jobs
R_m	Maximum number of production runs of machine m
r_m	A system with release date for machines
Rm	A system with unrelated parallel machines
routf	A system with routing flexibility
s	Sublot index
S_j	Maximum number of sublots of job j
S_{jk}	A system where setup time for a job is only dependent on the preceding job
S_k	A system with sequence independent setup times
S_{mjk}	A system where Setup time for a job is dependent on the machine in addition to the preceding job
$S^*_{o,j,m}$	Setup time for operation o of sublot s of job j if it is the first operation to be processed on machine m
S_{omjk}	A system setup time for a job is dependent on it operation in addition to the machine and the preceding job
$S_{o,j,m,o',j'}$	Setup time for operation o of a sublot of job j, where operation o' of a sublot of job j' is the preceding operation on machine m

Si	A system where sublots from different lots can be intermingled
Sn	A system where sublots from different lots are not allowed to be intermingled
$t_{o,j}$	Processing time of operation o of job j, after select a machine
$T_{o,j,m}$	Unit processing time for operation o of job j on machine m
Tc	A system with consistent sublot sizes
Tf	A system with fixed sublot sizes
Tq	A system with equal sublot sizes
tree	A system where precedence relation is in form of a tree
Tv	A system with variable sublot sizes
$v_{o,j,o',j',m}$	A binary variable that takes the value 1 if the o^{th} operation of job j has precedence over o'^{th} operation of job j' on machine m, 0 otherwise
$\hat{v}_{o,j,o',j',m}$	A binary variable that takes the value 1 if the o'^{th} operation of job j' must be processed immediately after completion of o^{th} operation of job j on machine m, 0 otherwise
$w_{o,j}$	Starting time of operation o of job j
$x_{r,m,o,j}$	A binary variable which takes the value 1 if the r^{th} run on machine m is for operation o of job j, 0 otherwise
$x_{r,m,o,s,j}$	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
$y_{m,o,j}$	A binary variable which takes the value 1 if operation o of job j is performed on machine m,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
$z_{r,m}$	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

α	A field which represents machine environment
$\alpha_{s,j}$	Represents each gene in the left hand side of the chromosome, which takes a random value in the interval [0, 1]
β	A field which represents job and resource characteristics
β'	A field which represents sublot-related features
γ	A field which represents optimality criteria
$\gamma_{s,j}$	A binary variable that takes the value 1 if sublot s of job j is non-zero ($b_{s,j} \geq 1$), 0 otherwise
Θ	Step amount
ρ	Crossover probability
σ	Mutation probability
Ω	Large positive number
$\sum w_j C_j$	Total weighted flow time
$\sum w_j T_j$	Total weighted tardiness
$\sum w_j U_j$	Weighted number of tardy tasks

LIST OF ACRONYMS

ACO	Ant Colony Optimization
AI	Artificial Intelligence
BB	Branch and Bound
CS	Computer Science
EA	Evolutionary Algorithm
EP	Evaluation Programming
ES	Evaluation Strategy
FJSP	Flexible Job Shop Scheduling Problem
FJSP-LS	Flexible Job Shop Scheduling Problem with Lot Streaming
FNJS	Fixed Number Job Splitting
FSSP	Flow Shop Scheduling Problem
GA	Genetic Algorithm
GAJS	Genetic Algorithm based on Job Splitting
GT	Group Technology
HEA	Hybrid Evolutionary Algorithm
HFS	Hyrbid Flow Shop
HGA	Hybrid Genetic Algorithm
HPSO	Hybrid Particle Swarm Optimization
IOAM	Intelligent Operations Assignment Mutation
JIT	Just-In-Time
JLOSC	Job level operation Sequence Crossover
JSP	Job Shop Scheduling Problem
LHS	Left Hand Side

LP	Linear Programming
LRH	Low-level rely hybrid
LS	Lot Streaming
MILP	Mixed Integer-Linear Programming
MLT	Manufacturing Lead Time
NGA	New Genetic Algorithm
OMAC	Operation-to-Machine Assignment Crossover
OR	Operations Research
OSSM	Operations Sequence Shift Mutation
OSSP	Open Shop Scheduling Problem
PGA	Parallel Genetic Algorithm
PI	Pairwise Interchange
PMS	Parallel Machine Scheduling
RHS	Right Hand Side
ROAM	Random Operation Assignment Mutation
SA	Simulated Annealing
SGA	Sequential Genetic Algorithm
SLOSC	Sublot Level Operation Sequence Crossover
SMS	Single Machine Scheduling
SPC	Single Point Crossover
SSD	Sublot Size Degenerator
SStM	Sublot Step Mutation
SSwM	Sublot Swap Mutation
TBC	Time-Based Competition
TS	Tabu Search
WIP	Work-In-Process

Chapter 1

Introduction

Scheduling is a decision-making process, which has become an imperative requirement for survival of various industries in the current competitive marketplace. Henry Gantt was among the pioneers who considered the significance of scheduling in manufacturing industries in 1910's (Pinedo, 2012). Nevertheless, research on scheduling theory did not receive considerable attention until the early 1950's. The research on scheduling theory is prompted by the wide range of its application in both industry and non-industry areas including production planning, computer control, agriculture, hospitals, transportation, etc. Generally, scheduling is an ongoing process of allocating a limited variety of resources to tasks over time, so as to optimize one or more objectives (Aarts and Lenstra, 1997; Lawler *et al.*, 1993). In this thesis, we restrict our attention to machine scheduling problems.

1.1. Machine Scheduling

A general machine scheduling problem consists of two sub problems: (1) allocation and (2) sequencing problems. When using a number of **machines** or

processors to process a number of **items** or **jobs**, each having one or more **operations** or **tasks** or **activities**, generally the operations must be performed in a specified order on various eligible machines. In this situation, given an objective function, we need to find the optimum allocation of the jobs to the machines (allocation problem), and the optimum processing order of the jobs on each machine (sequencing problem) such that the corresponding objective function is minimized or maximized. The optimality criteria on machine scheduling can be based on completion times, due dates, inventory cost and utilization (Eiselt and Sandblom, 2004; Kan, 1976). These types of machine scheduling problems can be classified into different categories based on three fields: machine environment, job characteristics, and performance measure. In the following, the description of these categories in each field are introduced.

1.1.1. Machine environment

Different configuration of machines can be encountered in a scheduling problem. In literature, machine environment is categorized as the single machine, parallel machines, and dedicated machine problems (Multi-stage problems). Each of these environments may include different sub-categories.

Single Machine Problems

In single machine scheduling (SMS), a group of jobs are assigned to a single machine. Since the 1950's, with the initial works of Jackson (1955) and Smith (1956), single machine problems have been given considerable attention (Brucker, 2007). The innate characteristic of these types of problems in addition to their ability in providing a basis for more general and complicated problems has given rise to much research in this area. Sometimes, in multi-processor environments, single machines can be used in bottlenecks, or as task organizers for expensive processors. Also, an entire production line may be considered as a single machine or

more complicated scheduling problems may be divided into sub problems, which are solvable with single machine solution procedures. Therefore, single machine scheduling has a fundamental role in providing vision for further developments in more general scheduling problems (Błażewicz *et al.*, 2007).

Parallel Machines Problems

In parallel machine scheduling (PMS), we have a number of jobs that need to be processed on a number of parallel machines so as to optimize objective functions of minimization of the total cost, the total completion time, and the maximum lateness (van den Akker *et al.*, 1999). Parallel machine scheduling is among the most widely-studied subjects in scheduling due to its application in both theory and practice (Błażewicz *et al.*, 2007). Regarding the theoretical aspect, parallel machine scheduling is a more general form of single machine problem and also a special case of flexible flow shop scheduling. Concerning the practical aspect, this type of problem received considerable attention because it is typical to have a parallel machines system in a real-world situation, and also the solution techniques for this type of problem can be used in decomposition procedures for multi-stage systems (Pinedo, 2012).

In general, this kind of scheduling problem can be divided into three main categories including identical, uniform and unrelated parallel machines. In the identical category, all machines work with the same speed and the processing time for each job is the same on every machine. In the uniform category, every machine has a different speed factor and each job contains only one activity. In the unrelated parallel machine scheduling category, there is no specific relation between the processing times of jobs on different machines (Chaudhry *et al.*, 2010).

Open Shop Problems

In an open shop scheduling problem (OSSP), jobs are comprised of a number of operations. Each operation has a specific processing time and needs only one machine for processing. However, no restrictions are specified for the sequence of operations of a job on the various machines, i.e., the sequence in which the operations of a job must be processed can be chosen arbitrarily. At any time, each machine cannot carry out more than one operation, neither can each job be processed by more than one machine. The goal of this type of problem is to find a schedule of operations on machines, i.e., to find the starting time of the operations, which minimize the overall finishing time known as makespan. This type of problem first was introduced by Gonzalez and Sahni (1976) to model real-world situations, like testing facilities, where operations can be processed by any order on machines (Błażewicz *et al.*, 2007; Low and Yeh, 2009; Khuri and Miryala, 1999).

Flow Shop Problems

Since Johnson (1954), many researchers have dedicated their effort to study flow shop scheduling problems (FSSP). These type of systems have extensive practical application in various industries where jobs should be processed continuously on multiple machines in series (Grabowski and Pempera, 2007). Almost one out of four manufacturing systems, assembly lines and service information facilities are represented by flow shop systems (Qian *et al.*, 2009).

In a classical flow shop problem, operation sequence of jobs are identical indicating that all jobs should follow a specific order to visit workstations, where each workstation includes only one machine. Each machine can carry out one and only one part at a time, and also each part can be processed on one machine at the same time. It is required for each job to visit each work station only one time. Consequently, the number of operations for every job is equal to the number of

machines. However, jobs may have operations with no processing time; that is to say, a job may skip over some of the machines (Reza Hejazi and Saghafian, 2005; Pinedo, 2012; Emmons and Vairaktarakis, 2012). As illustrated in Figure 1.1, in flow shop environment, there is an entrance work station that first operation of each job is processed there, and also there is an exit work station which processes the last operation of each job.

Although, a flow shop system may seem similar to an assembly line, the two differ in three major ways. Unlike the assembly lines, which can only manufacture a standard product, a flow shop is capable of processing a variety of jobs. Moreover, in flow shops, it is not essential for a job to visit every work-station to be processed; whereas in an assembly line, a job must be processed on all the machines. Also, while in a flow shop, machines can be loaded independent of the previous machine, it is not possible for a workstation to work independently in an assembly line. Due to the mentioned differences, flow shop systems may be classified as conservative assembly lines (Heller, 1959; Gupta and Stafford Jr, 2006).

The counterpart of classical flow shops are hybrid flow shops (HFS). These types of flow shops are production systems consisting of the series of work stations, in each of which a job can be executed by multiple parallel machines. In this type of production system, it is possible to have only one machine in some work stations, however, it is necessary to have at least one workstation with more than one machine in the shop. Hybrid flow shop problems can be considered a general combination of a parallel machines system and a classic flow shop. Hybrid flow shops are productions systems where the goal in parallel machine scheduling is to find the best allocation of jobs to the machines so as to optimize the given objective function. Whereas, in classic flow shop scheduling, the aim is to determine the optimum sequence of the jobs throughout the shop. Therefore, in a hybrid flow shop, the objective is to determine the allocation of machines to

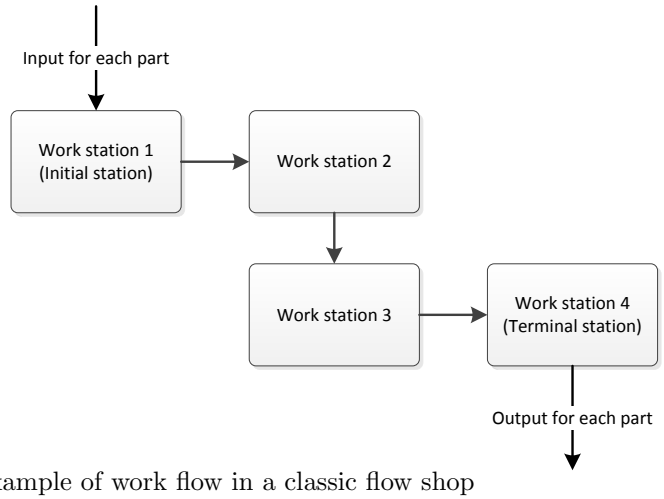

Figure 1.1: An example of work flow in a classic flow shop

the jobs, and also the processing order of the jobs in each work station according to one or more optimality criteria (Linn and Zhang, 1999; Ribas *et al.*, 2010).

Job Shop Problems

In flow shop scheduling, all the jobs are restricted to be processed in a specific order, i.e, the flow of jobs are unidirectional. However, in job shop scheduling, the technological sequence of machines for each job is different; that is to say, unlike the flow shops, there is no entrance workstation that processes the first operation of each job and neither is there a terminal workstation that processes the final operation of each job (Baker and Trietsch, 2009) as depicted in Figure 1.2. A classic job-shop scheduling problem (JSP) consists of scheduling a set of jobs which must be processed on a set of machines, where each job is composed of a sequence of consecutive operations. Each machine is able to process one operation at a time without interruption, and the processing sequence of operations of each job is prescribed (Adams *et al.*, 1988).

The more general type of JSP is called flexible job-shop scheduling (FJSP). Brucker and Schlie (1990) were recognized by the majority of authors who first introduced FJSP. Unlike JSP, in FJSP, operations are allowed to be processed on

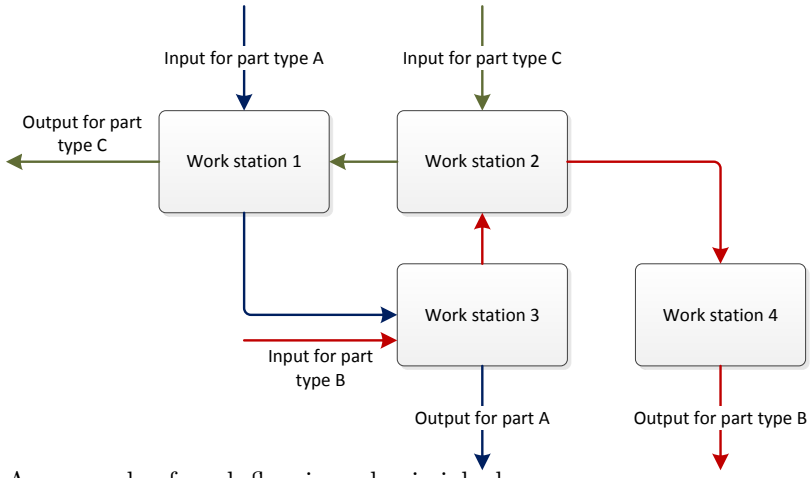

Figure 1.2: An example of work flow in a classic job shop

more than one machine from a given set of available machines. This extension makes FJSP a more complex problem than classical JSP due to the fact that in addition to the operations assignment, routing of jobs should also be determined.

1.1.2. Job and resource characteristics

In machine scheduling problems, several attributes and restrictions can be considered for tasks and resources. In the following, possible constraints which can be placed on jobs and machines are described.

Job release date (ready time)

This is the time at which a job becomes available for processing. If a release date is specified for a task, then that job can not be started before the specified date.

Preemption

In preemptive scheduling, processing of a job or an operation on a machine may repeatedly be interrupted by a scheduler in order to allocate the machine to a different task with the higher priority. The interrupted job (preemptive job) will be resumed at a later time on the same machine or another machine.

Precedence relation

This determines the precedence constraints, if any, between the jobs. Job j may not be allowed to start being processed before the completion of one or more particular jobs. In this case, those particular jobs have precedence over job j. There are several forms of job dependencies in scheduling. If, in a system, at most one job has an immediate precedence over job j, and job j has an immediate precedence over at most one another job, the job relations is referred to as *chains*. If at most one job has an immediate precedence over job j, but job j has an immediate precedence over more than one job, the precedence constraint is called *intree*. If job j has an immediate precedence at most over one job, but more than one job has an immediate precedence over job j, the job relations is referred to as *outtree*. If jobs have no precedence on a production system, they are referred to as *independent tasks* (Pinedo, 2012).

Sequence-dependent setup

In many real life situations such as pressing in plastic manufacturing, die changing in metalworking industries and role slitting in converter industries, setup time for a job is dependent on its sequence. A sequence-dependent setup time implies that setup time for an operation on a machine is the function of the immediate preceding operation on the same machine. The dependency of setup time to the preceding job arises when there are setup operations such as cleaning-up and changing the tools which are necessary to be done to make the machine ready for processing the next job (Kim and Bobrowski, 1994; Naderi *et al.*, 2009). Thus, in this type of manufacturing shop, setup for a job is formed by a setup that depends on the preceding operation on the machine (sequence-dependent setup) and a setup which depends on the previous operation of the job (sequence-independent setup) (Rossi and Dini, 2007).

Routing flexibility

In the presence of routing flexibility in a manufacturing system, for a given process plan, alternative routes may exist for certain jobs. Routing flexibility will provide more options to assign operations to machines, and also it enables a scheduler to cope with costly stops in production lines resulting from busy or inactive machines. As stated before, by considering routing flexibility, classical job shop scheduling is extended to flexible job shop scheduling where alternative machines are available for processing an operation (Stecke and Raman, 1995; Rossi and Dini, 2007).

Machine release date

This is the time at which a machine is released from preceding schedule and ready to to begin processing of the current schedule. In many real-life situations, the presence of ongoing processes from previous schedule is a very common occurrence. Machine release date can significantly affect the production schedule, when alternative schedule plans (Routing flexibility) is also considered in a problem. Since, selection of a machine with earlier release date may be more desirable for a scheduler rather than a machine with later release date.

Lag time

Sometimes, in typical production systems, it may be required to have a waiting time between successive operations of the same job since a job may need time to get prepared for further operations after completion of earlier operations. This situation may occur when, for instance, products need to be dried or cooled before going to the next workstation. This waiting time is referred to as Lag time in literature (Ruiz *et al.*, 2008).

Machine availability

In many manufacturing systems, a certain machine may not be available continuously at all times, and all the machines in a shop may not be able to work simultaneously. This phenomena frequently occurs in various industries due to machine breakdowns (stochastic), preventive maintenance and shift changes (deterministic) (Lee, 1996).

Batch-processing machines

A Batch-processing machine is the one which is capable of processing several jobs simultaneously. Heat treatment furnaces in metalworking, oxidation tubes in wafer fabrication process, tanks or kilns in chemical processes and burn-in ovens in semiconductor testing are examples of batch- processing machines (Lee, 1999; Melouk *et al.*, 2004).

Blocking

Scheduling problems with blocking constraints are usually found in flow shop environments. Blocking phenomena implies that, intermediate queues of parts waiting for their next operation are not permitted in the production system due to having zero buffer, or limited buffer between the workstations. It means that, the upstream machine should hold a completed job until the next machine becomes available for processing, which prevents the upstream machine from processing the next job (Pinedo, 2012). Just-in-time production systems are one of the reasons for the occurrence of blocking environments, where efforts are to limit the work-in-process inventory (Hall and Sriskandarajah, 1996). Also, in some manufacturing systems, like concrete blocks manufacturing and chemical industries, because of production technology or characteristics of the material, the intermediate buffers (storage) are not permitted, which results in occurrence of blocking environments (Ronconi, 2004).

No-wait

Scheduling problems with no-wait constraint regularly arise in flow shops. No-wait production environments are the more restricted version of systems with blocking constraint. In a no-wait environment, a job is required to be processed through the shop without any interruption (waiting time) between two consecutive workstations or on the machines. Thus, the production cycle time for a job must be exactly equal to the sum of processing times of all operations. With no-wait restriction, a scheduler may be required to delay the starting time of a job in order to guarantee that the job will not wait for any machine through the production line. Scheduling systems with no-wait constraint arise when due to the characteristic of materials (e.g., temperature and viscosity), a job is required to move to the next workstation immediately after completion of processing. Examples of such a system can be found in steel production, plastic molding, silverware production, anodizing of aluminum products, chemical and pharmaceutical industries, and canning operation in food processing industries (Pinedo, 2012; Hall and Sriskandarajah, 1996).

Recirculation

Recirculation implies that a job may need to visit certain workstations more than one time. This phenomena may occur in both flow shop and job shop environments.

1.1.3. Optimality criteria in scheduling problems

Generally, the goal in any scheduling environment is to find a feasible schedule which minimizes the costs that are attributed to the scheduling decisions. The objective functions in scheduling problems can be categorized into bottleneck objectives and sum objectives. In the bottleneck objectives, the maximum of the

costs associated with the finishing time of the total jobs should be minimized. However, in sum objectives, the sum of the costs should be minimized. Optimality criteria, can also be categorized into those based on the completion time, and those based on the due dates (Kan, 1976; Brucker, 2007). In the following, the most common objective functions in scheduling are described.

Minimizing makespan

Makespan or maximum completion time, is the amount of time needed to finish all the jobs. This value is obtained by subtracting the completion time of the last job from the start time of the first job. This objective is usually considered whenever the emphasis is on increasing the machines utilization and production flow.

Minimizing total weighted completion time / weighted flow time

In some situations, jobs have different priority levels, which is usually represented by assigning weights to the jobs. Total weighted completion time and weighted flow time are two typical metrics in such cases. The flow time of a job is the interval time (turn around time) between its arrival to the system and its completion time. Thus, if a job has a release date (ready time), the value of its completion time will be greater than the value of its flow time (Chekuri *et al.*, 2001).

Minimizing maximum lateness (tardiness)

Lateness of a job is defined as the amount of time the job actual completion time exceeds its original due date. Maximum lateness measures the worst due date violation. Lateness gets a negative value if a job is completed earlier than its due date. Usually, in real-life situations positive lateness values are associated with penalties, whereas there is no advantage in completing a job earlier than its due date. Therefore, most of the time, only the positive value of lateness (also

known as tardiness) is applied in problems. This objective is considered whenever the goal is to increase customer satisfaction and to meet the demand (Baker and Trietsch, 2009).

Weighted number of tardy jobs

As it mentioned earlier, due to the customer requirements, some jobs may be given higher priority than the others. Also, it is necessary for every job to meet its internal or external due dates. Tardy job refers to the jobs that violate their due dates. The weighted number of tardy jobs is one of the popular metrics in practical situations, since it is can be easily measured (Ghasemi, 2008; Pinedo, 2012).

1.2. Lot streaming

There exist several techniques in planning and scheduling which are used by industries to achieve reduction in manufacturing lead time and work-in-process (WIP) inventory levels. The implementation of some of theses strategies like just-in-time (JIT) and group technology (GT) requires a dramatic change in the way business is organized, which can be extremely costly and time consuming. However, a less expensive and more convenient solution for improving the efficiency is batch production.

In a traditional batch production, similar jobs are batched together. In other words, jobs are made in groups. The main reason for that is to decrease the total amount of time to set up the machines and transport the jobs. In this type of system, a finished part in a batch can move to the next station only if all parts in that batch are completed their processing. Consequently, a part may spend a large amount of time in a buffer waiting for the other parts in a same batch to be processed, whereas the next machine may be available to process that

job. This waiting time can be extremely long when larger production batches are considered. One promising solution to overcome this problem is the application of lot streaming (LS) or lot sizing procedures (Potts and Van Wassenhove, 1992; Zipkin, 1986).

The lot streaming concept, which was first introduced by Reiter (1966) is a manufacturing technique in which production lots (consisting of several identical jobs) are split into sublots (transfer batch) in order to benefit from simultaneous processing of different sublots at a lot at different workstations (Potts and Baker, 1989). In today's era of time-based competition (TBC), this concept has been implemented by many top-notch companies in order to reduce their manufacturing lead time (MLT) and to improve their customer service (Blackburn, 1991; Bockerstette and Shell, 1993; Chang and Chiu, 2005).

Lot streaming refers to the decision of determining the number of sublots in a production lot, the magnitude of each sublot and their processing sequence so as to optimize the scheduler objective. Unlike the batch production in which size of production lots are fixed, in a system where lot streaming is allowed, a finished sublot can be transported to the next work station and start processing while other jobs from the same lot but of a different sublot are still waiting to be processed on the earlier workstation (Potts and Van Wassenhove, 1992). In other words, lot streaming makes it possible to process an operation and its preceding operation of a same job simultaneously. As a result of this overlapping of production, the amount of time a part spends in the system (cycle time) is remarkably reduced. Thus, the makespan and work in process can be minimized. However, by increasing lot splitting, the total handling and setup times (cost) may increase due to the increment in the number of sublots, whereas the total inventory cost decreases (Low *et al.*, 2004b).

In fact, if no setup and/or transfer times are considered for the jobs in a problem, the optimum solution may be to have only one job in each sublot (Chan

et al., 2009). Evidently, in practical situations where machine setups are present, a tradeoff exists between the amount of time saved by splitting lots to the sublots and the additional time required because of extra setups. Nevertheless, despite the negative influence of additional setup times on makespan, lot streaming is still significantly efficient (Dauzère-Pérès and Lasserre, 1997; Buscher and Shen, 2011). The potential benefits of lot streaming in flow-shop and job-shop environments are addressed in Kalir and Sarin (2000) and Low *et al.* (2004b), respectively.

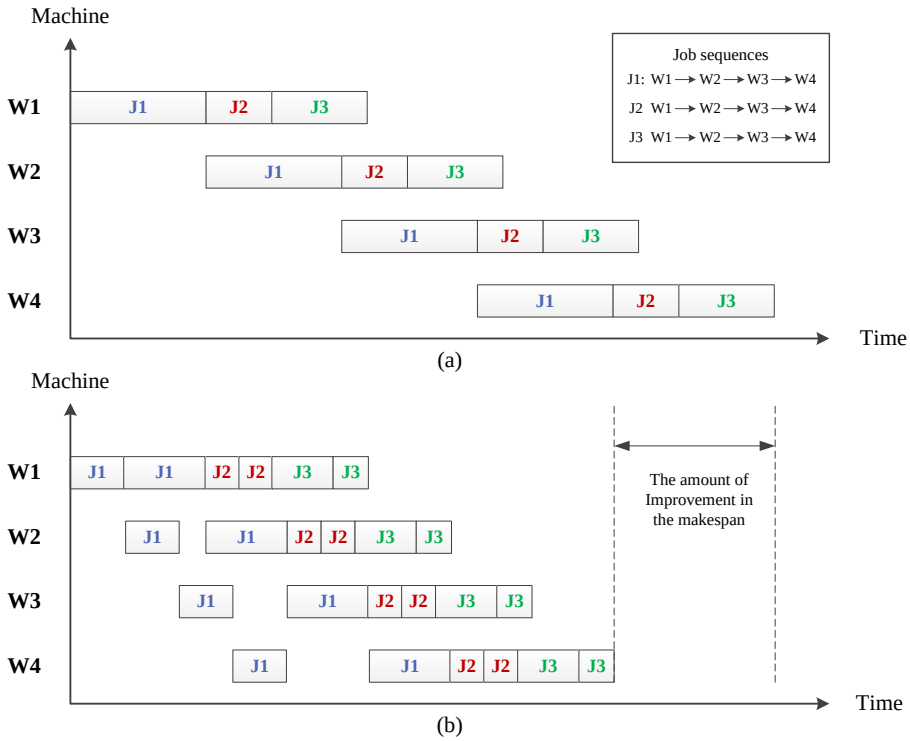

Figure 1.3: An example of lot streaming in a classic flow shop

Figures 1.3 and 1.4 illustrate the impact of lot streaming in a classical flow shop and a classical job shop, respectively. In these examples, it is assumed that the processing sequence of sublots can be different than the original lot. As can be seen in these figures, by application of lot streaming the makespan has been improved in both environments. In flow shop environments, it is extremely beneficial to divide lots to sublots, since lots are processed in the same order and have the same characteristics. However, application of lot streaming cannot be

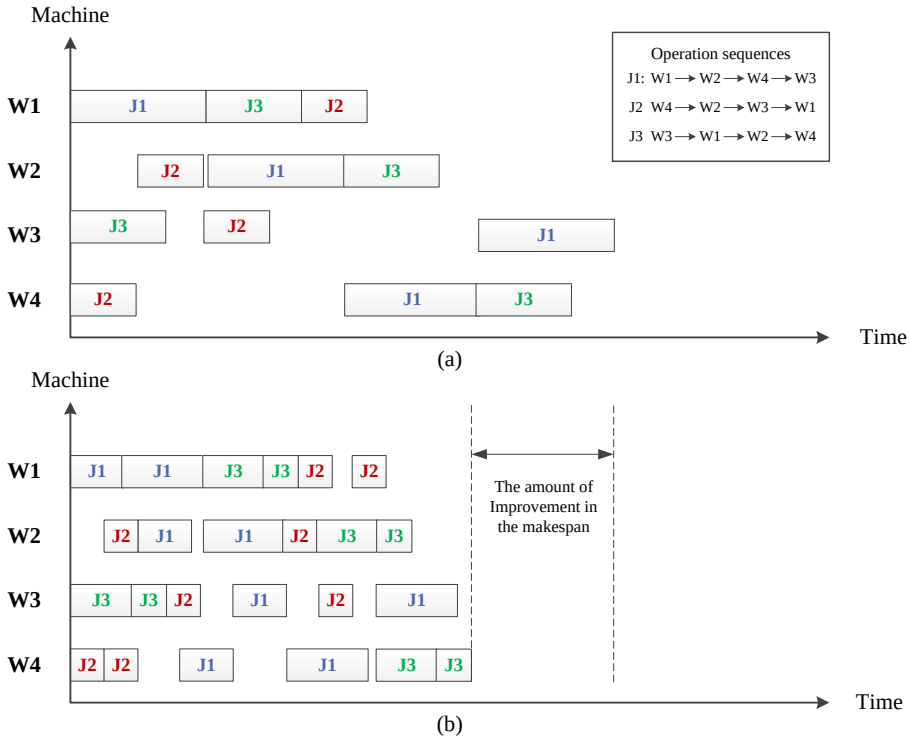

Figure 1.4: An example of lot streaming in a classic job shop

guaranteed to be fruitful, due to the fact that there are more inherent constraints in a job shop problem than in a flow shop problem (Low *et al.*, 2004b; Chan *et al.*, 2009).

1.3. Lot streaming terminology

In this section different attributes for a scheduling problem with lot streaming are presented which are adopted from Feldmann and Biskup (2008).

- *Single product/multiple products*: A lot streaming problem may deal with either a single product or multiple products.

- *Fixed/Equal/consistent/variable sublots*: Sublots are termed fixed, when the number of jobs in each sublot in the system is identical on all workstations. Equal sublots means that every sublot of a product consists of the

same number of jobs. A consistent sublot refers to the case that sublot sizes can not alter over time. The sublot is called variable if the size of a sublot is not restricted to be consistent.

- *Discrete/continuous sublots*: A discrete sublot must consist of integer number of parts (e.g., Automotive industries), whereas the number of jobs in a continuous sublot can be decimal and not just integer (e.g., chemical industries).

- *Non-idling/intermittent idling*: In a non-idling system, sublots must be processed immediately after the pervious sublot on the machine completes its processing. However, in a system where intermittent idling is allowed machines can have idle times between the sublots.

- *Attached setups/detached setups*: When it is possible to perform machine setup for a job prior to its arrival, the machine setup is called detached (anticipatory) setup . However, the setup for a job is called attached (non-anticipatory) setup time, provided that the presence of that job is necessary during the setup procedure.

- *Intermingling/non-intermingling sublots*: Intermingling sublots can only exist in multi-product settings. Assuming that Intermingling sublots are allowed, the sequence of sublots of a particular lot in a workstation can be interrupted by sublots from different lots. However, for non-intermingling sublots, it is not permitted to interrupt the sequence of sublots of a lot in a workstation. In other words, the sequence of sublots of a product can not start processing in a workstation unless the sublots of the preceding product on that workstation complete their processing (Ghasemi, 2008).

1.4. A classification scheme

It is clearly obvious that by combining the aforementioned parameters, attributes and objectives, a large variety of scheduling problems are created. A useful and convenient way to characterize this huge number of scheduling problems is to adopt a comprehensive representation scheme. In this section, a novel comprehensive classification scheme is presented to cover a wide range of scheduling models. This classification is an extension and combination of the existing classifications in Graham *et al.* (1977), Aarts and Lenstra (1997), Eiselt and Sandblom (2004), Chang* and Chiu (2005), Pinedo (2012), and Cheng *et al.* (2013).

In table 1.1, a typical scheduling problem is described by a four-field representation $\alpha|\beta|\beta'|\gamma$. The α field represents the machine environment, and it can only contain two or less items. The β field provides information about the job and resource characteristics and also the restrictions and constraints existing in the model. β field can contain no item, single item or multiple items. If lot streaming is permitted in a system, sublot-related features appear in β' field. At the end, γ field gives information about the model optimality criteria. Unlike the previous fields, γ field can only contain one item.

Table 1.1: Four field classification scheme

Comprehensive classification of scheduling problems		
Main field	Subfield	Level
α	Production type(α_1)	Single machine (ϕ)[1]
		Identical parallel machines (Pm)
		Uniform parallel machines (Qm)
		Unrelated parallel machines (Rm)
		Open shop system (Om)
		Flow shop system (Fm)

continued ...

[1]If the symbol ϕ is assumed for any parameters in this table, it will appear as an empty spot in the representation scheme

... continued

Comprehensive classification of scheduling problems		
Main field	Subfield	Level
		Flexible flow shop system (*Ffc*)
		Job shop system (*Jm*)
		Flexible job shop system (*FJm*)
	Number of parts(α_2)	Single product is available(L_1)
		Multiple products are available(L_n)
β	Job release date (β_1)	No release date is specified for the jobs(ϕ)
		Release date is specified for the jobs(r_j)
	Preemption(β_2)	Preemption is allowed(*prmp*)
		Preemption is not allowed (ϕ)
	Precedence relation(β_3)	There is no precedence relation between the jobs (ϕ)
		Precedence relation is in form of an arbitrary acyclic graph (*prec*)
		Precedence relation is in form of a tree (*tree*)
		Precedence relation is in form of a chain (*chain*)
	Job setup type(β_4)	No setup time is specified (ϕ)
		Sequence independent setup times is specified for the jobs (S_k)
		Setup time for a job is only dependent on the preceding job (S_{jk})
		Setup time for a job is dependent on the machine in addition to the preceding job (S_{mjk})
		Setup time for a job is dependent on its operation in addition to the machine and the preceding job (S_{omjk})
	Routing flexibility (β_5)	No alternative route exists for the jobs. (ϕ)
		Alternative routes exist for the jobs. (*routf*)

continued ...

... continued

Comprehensive classification of scheduling problems		
Main field	Subfield	Level
	Machine release date (β_6)	All the machines are ready at the beginning of the scheduling. (ϕ) Release dates are specified for the machines (r_m)
	Lag time (β_7)	Waiting time between successive operations of any job is not required. (ϕ) Waiting time between successive operations of particular jobs is necessary. (lag)
	Machine availability (β_8)	All the machines are working non-stop all the time (no breakdown) (ϕ) Particular machines are not available continuously all the time. ($brkdwn$)
	Batch-processing (β_9)	Machines capable of doing batch processing are not available. (ϕ) There exist machines with ability to do batch processing. ($btchm$)
	Blocking (β_{10})	There is unlimited buffer between the workstations (ϕ) The intermediate buffers (storage) are extremely limited or not permitted ($block$)
	No-wait (β_{11})	Waiting time (queues) between two consecutive workstations is permitted. (ϕ) A particular job must never stop between two consecutive workstations. (nwt)
	Batch production (β_{11})	Jobs are scheduled individually. (ϕ) Jobs are made in bulk, however, lot sizing is not permitted ($btchj$)

continued ...

. . . continued

Comprehensive classification of scheduling problems		
Main field	Subfield	Level
		Jobs are made in bulk, and lot sizing is allowed ($btchls$)
β' [2]	Sublot type (β'_1)	Sublot sizes are fixed. (Tf)
		Sublot sizes are equal. (Tq)
		Sublot sizes are consistent. (Tc)
		Sublot sizes are variable. (Tv)
	Number of sublots (β'_2)	Number of sublots in each lot is predetermined ($FixN$)
		Number of sublots in each lot is not given ($FlexN$)
	Divisibility of sublot size (β'_3)	Sublots are discrete. (Dd)
		Sublots are continuous. (Dc)
	Operation continuity (β'_4)	Machines can have idle time. (ϕ)
		Machines are not allowed to have idle time. (Ni)
	Setup predictability (β'_5)	Setup time for certain machines are anticipatory (Pa)
		All machine setups are not anticipatory (Pn)
	sequence continuity (β'_6)	sublots from different lots can be intermingled. (Si)
		sublots from different lots are not allowed to be intermingled (Sn)
γ	Performance measurement(γ_1)	Makespan (C_{max})
		Total weighted flow time ($\sum w_j C_j$)
		Maximum lateness (L_{max})
		Total weighted tardiness ($\sum w_j T_j$)
		Weighted number of tardy tasks ($\sum w_j U_j$)

[2] It is obvious that if $\beta_{11} \neq btchls$, no symbol appears in this field.

1.5. Research in this thesis

This study provides a novel hybrid genetic algorithm based on the method provided in Defersha and Chen (2012) for scheduling and lot streaming of multi products in flexible job shop environment. The comprehensive problem studied in this thesis takes into account routing flexibility, sequence-dependent setups, machine release dates and lag time. In addition, the sublots of products can have attached or detached setup times, and also are allowed to be intermingled. According to the above classification scheme, the problem considered in this research denotes by : FJm, L_n | S_{omjk}, $routf$, r_m, lag, $btchls$ | Tv, $FixN$, Dc, xPn,Si | C_{max}.

1.6. Organization of the thesis

The remainder of this study proceeds as follows. In Chapter 2, a review of literature on hybridization of metaheuristics, and especially evolutionary algorithms is presented. In addition, Chapter 2 reviews earlier studies on methods for lot streaming problems. In Chapter 3, a mixed integer-linear programming (MILP) model for flexible job shop problem with lot streaming (FJSP-LS) is represented based on Defersha and Chen (2012). In Chapter 4, a detailed solution procedure for both pure GA and the proposed hybrid GA is depicted. Numerical examples are introduced in Chapter 5. And finally, discussion and conclusions are presented in Chapter 6.

Chapter 2

Literature Review

2.1. Introduction

Over the last years, many efforts have been made by researchers to explore the field of metaheuristics. This is due to the fact that metaheuristics have proven themselves to be effective alternatives to classical heuristics and optimization methods for complex optimization problems. The term metaheuristic which was first devised by Glover (1986), originates in computer science (CS), artificial intelligence (AI), and operations research (OR) communities. Metaheuristics are a class of approximate methods which are designed to obtain a near-optimal solution at relatively low computational cost. In addition, like other approximate methods - in metaheuristics - in order to reduce the computational time substantially, the guarantee for obtaining an optimal solution should be sacrificed (Osman and Laporte, 1996; Gendreau and Potvin, 2005; Blum *et al.*, 2008; Bianchi *et al.*, 2009). The salient examples of metaheuristics include Simulated Annealing (SA), Tabu Search (TS), Evolutionary Algorithms (EA), and Ant Colony Optimization (ACO). For further details on metaheuristics, refer to Ribeiro and Hansen (2002); Kochenberger *et al.* (2003); and Rayward-Smith *et al.* (1996).

2.2. Hybrid metaheuristics

In recent years, it has become apparent that pure application of classical metaheuristics is limiting. Therefore, many approaches have emerged using a combination of algorithmic concepts of various methods both from inside and outside of the metaheuristic field with a view to attaining higher efficiency. Generally, this type of approach is called hybrid-metaheuristics. The main purpose of hybridization is to benefit from unified advantages of the individual pure strategies and produce new algorithms in order to achieve an effective performance and a profitable synergy in solving hard optimization problems (Raidl, 2006; Blum *et al.*, 2011; Yi *et al.*, 2012).

Several classifications and categorizations on hybrid metaheuristics have been developed in recent years. Talbi (2002) provided a classification based on the design issues in hybridization. According to this paper, in the first level, hybrid metaheuristics were characterized in low-level versus high-level hybridizations. In low-level hybridization, a functional composition of an optimization algorithm is exchanged with other metaheuristics. Unlike low-level hybridization, in high-level hybridization, not only is there no internal relation between optimization algorithms which are to be combined together, but also all algorithms preserve their identities.

In the second level of this hierarchical classification, a combination of optimization algorithms is classified into teamwork versus rely hybridizations. Similar to the way a pipeline works, rely hybridizations are those in which the output of an optimization algorithm is used as an input for the other algorithm. On the contrary, in teamwork hybridizations, optimization algorithms cooperate with each other so that each algorithm performs a search in the solution space. With respect to this classification, our proposed method in this thesis lies in Low-level rely hybrid (LRH) category (Talbi, 2002).

Moreover, in Raidl (2006), hybridization methods are further distinguished

based on the types of algorithms which may be combined, level of hybridization, order of execution, and control strategy. According to this categorization, metaheuristics may be hybridized with other metaheuristics, problem-specific algorithms, simulations, exact techniques, heuristics, and soft computing methods. Furthermore, control strategies in hybrid metaheuristics are also divided into integrative and collaborative categories. In the integrative approach, an optimization approach is embedded in a primary approach to work as a subordinate algorithm. However, in the collaborative approach, the relation between the optimization algorithms is limited to exchanging the information and the algorithms work separately (Raidl, 2006).

With regard to above classification, the proposed method in this thesis can be deemed an integrated approach, in which a metaheuristic method is combined with an exact method. Exact method encompasses branch and bound (BB), dynamic programming, and linear and integer programming. For broader discussions on the categories of hybrid metaheuristics, refer to Blum *et al.* (2011). Likewise, further information regarding the parallel hybrid metaheuristic classification can be found at Cotta *et al.* (2005).

Moreover, in Puchinger and Raidl (2005), hybrid metaheuristics - which employ exact methods as a complement, are categorized as the following classes: The exact methods are incorporated with metaheuristics :

- To solve the relaxed problem in order to provide an auspicious initial solution or to provide a guide for local search or constructive algorithms;
- To explore large neighborhood in local search-based metaheuristics;
- To merge their solutions with metaheuristics in different stages-similar to using an exact method as a crossover operator in Genetic Algorithm; and
- To complete the missing part of a solution representation in evolutionary algorithms, while a complete representation of the solution via evolutionary

algorithms is not possible.

With regard to this classification, our proposed method follows the "merging solutions" category. A comprehensive classification of the hybridization of metaheuristics with exact methods can be found in Jourdan *et al.* (2009).

2.2.1. Hybridization of evolutionary algorithms

Recently, hybridization of evolutionary algorithms have received noteworthy interest owing to their outstanding performance in solving several hard optimization problems (Grosan and Abraham, 2007). Evolutionary algorithms are a class of stochastic algorithms garnered from models of organic evolution. The prominent examples of these computational algorithms include Genetic Algorithms (GAs), Evaluation Strategies (ESs), Genetic Programming, and Evaluation Programming (EP) (Bäck, 1996; Lozano and García-Martínez, 2010). There exist several possibilities for a combination of evolutionary algorithms with other optimization techniques. Some techniques/algorithms like local search may be used in the initialization stage or among the offspring created by mutation or recombination. Also, optimization techniques may be drawn upon as an operator in the hybridization of evolutionary algorithms (Grosan and Abraham, 2007).

In Moscato (1989), hybridization between evolutionary algorithms and local search algorithms is termed memetic algorithms. Moreover, a combination of evolutionary algorithms with constructive heuristics and exact method is similarly categorized in memetic algorithms. In addition to the term - memetic algorithm, several names have been assigned in literature to evolutionary algorithms incorporated with local search approaches; these names include hybrid genetic algorithms, genetic local searches, and Lamarckian genetic algorithms. For more details about memetic algorithms, an interested reader is referred to Moscato *et al.* (2004); Hart *et al.* (2005); and Krasnogor and Smith (2005).

2.2.2. Hybrid genetic algorithm

In this thesis, a novel hybrid genetic algorithm is introduced. The genetic algorithm (GA) is an evolutionary algorithm proposed by (Holland, 1975) based on ideas from Darwin's theory of evolution (Goldberg, 1989). Similar to any metaheuristics, in order to have an efficient performance in the GA, the explorative and exploitative capabilities of the GA should be well balanced. The explorative ability (diversification) refers to the ability of the GA to conduct global search in the solution space in order to find the most promising regions, whereas the exploitative ability (intensification) refers to the ability of the GA to conduct local search in promising regions of the solution space in order to find best solutions (El-Mihoub *et al.*, 2006; Drezner and MisevičIus, 2012). However, while a typical genetic algorithm is designed to swiftly find the most promising regions of the solution space, the GA's ability to exploit best solutions in convergence regions is comparatively weak. In other words, by performing a pure GA on a problem, a moderately good solution can be rapidly obtained. Nevertheless, it takes a considerably long time to improve that good solution. Therefore, the slow convergence rate of a pure GA renders it computationally expansive to solve hard optimization problems (Yen *et al.*, 1995; Partheepan, 2004; El-Mihoub *et al.*, 2006).

One effective way to minimize this problem is to hybridize the pure GA by means of incorporating the other optimization algorithms within a GA. Researchers in the hybrid genetic algorithms (HGA) field have been very active in recent decades. Indeed, there exists abundant research in literature on various categorizations and classifications for the hybridization of genetic algorithms. The interested reader is referred to relevant reviews (Talbi, 2002); (El-Mihoub *et al.*, 2006); (Bianchi *et al.*, 2009); and (Drezner and MisevičIus, 2012).

2.3. Machine scheduling problems

Generally, several approaches have been devised to solve JSP. Exact methods, branch and bound, heuristic algorithms, and shifting bottle neck are among the methods which have been used for JSP with dimensions less than 15 parts by 15 machines. However, since JSP and particularly FJSP are in the class of NP-hard problems, exact methods are incapable of tackling the problems with greater dimensions in a reasonable amount of time. Therefore, various heuristics such as dispatching rules and local search as well as meta-heuristics approaches such as TS, SA, and GA have been utilized to solve these problems with a practical computational cost (Gonçalves *et al.*, 2005; Bagheri *et al.*, 2010).

Furthermore, these solution procedures for solving JSP and FJSP can be categorized into hierarchical and integrated approaches. In hierarchical approaches, the solution procedure is based on the decomposition of the original problem with the intention of reducing the problem complexity. On the other hand, integrated approaches are harder to solve and more optimal in terms of eventual outcome. In this type of approach, sequencing and assignment sub problems are treated simultaneously (Pezzella *et al.*, 2008). Among these approaches, the GA has been broadly applied to FJS problem in recent years.

2.3.1. Flow shop scheduling with lot streaming

In literature, numerous research on lot streaming can be found; however, most of these investigations are devoted to lot streaming in flow shop manufacturing systems. In contrast, research on job shop lot streaming is relatively limited (Low *et al.*, 2004a). In the majority of these investigations into lot streaming in flow shop, metaheuristics have been used as a main tool to solve the problem. (See for example (Tseng and Liao, 2008; Pan *et al.*, 2010, 2011b; Marimuthu *et al.*, 2013)). In this subsection, we restrict our attention to the application of GA and

hybrid GA in the lot streaming flow shop scheduling problems.

Flow shop scheduling under lot streaming environment can be categorized into single-job and multi-job problems. In single-job problems, the primary objective is to determine the optimal sublot sizes (best allocation of sublots). However, in problems with more than one job, the main goal is to simultaneously find the optimal sublot sizes, and their best processing sequence (Pan *et al.*, 2011a).

In Kumar *et al.* (2000), different heuristic methods for solving a no-wait flow shop scheduling problem with lot streaming is presented and evaluated. Their lot streaming problem involves three interdependent decision variables including, the number of sublots for each lot, the size of the sublots, and finally processing sequence of these sublots. They achieved promising results, when GA is employed to find the number of sublots and their processing sequence, and linear programming (LP) is used to determind the sublot sizes. Yoon and Ventura (2002) presented a hybrid GA for a lot streaming flow shop scheduling problem in which the number of equal sized sublots is fixed for each lot. Their proposed HGA incorporates LP and a Pairwise Interchange (PI) method, in order to prevent the premature convergence of GA and to preserve explorative ability of the GA.

In Martin (2009), a mixed-integer linear programming (MILP) assisted GA is developed for a multi-family flow shop lot streaming scheduling problem. In this study, MILP is adopted to find the optimum sublot sizes and GA is applied to determine the number of sublots in each lot and their processing sequence. In Ventura and Yoon (2012), a new genetic algorithm (NGA) is devised to solve a lot streaming flow shop problem with blocking (limited buffer capacity), in which sublot sizes are equal. In their new algorithm, selection and mating operators of a classical GA has been replaced by new operators. In Marimuthu *et al.* (2008), a genetic algorithm and a GA-based hybrid evolutionary algorithm (HEA) are proposed to obtain the best processing sequence of the sublots. Kim and

Jeong (2009) proposed an adaptive GA to solve a no-wait flexible lot streaming problem in flow shop environment. Furthermore, Defersha and Chen (2010a) developed a hybrid GA for a lot streaming flow shop problem with setup time and variable sublots. In their method, GA is applied to obtain the optimum value of integer variables (e.g, processing sequence of sublots), whereas LP is employed to simultaneously find the value of the continuous variables corresponding to each integer solution visited (e.g, size of each sublot). For a comprehensive review of flow shop scheduling problems with lot streaming, an interested reader is referred to Sarin and Jaiprakash (2007).

2.3.2. Job shop scheduling with lot streaming

As was mentioned earlier, a very limited number of researches in job shop scheduling with lot streaming (JSP-LS) have been reported in literature. In Dauzere-Peres and Lasserre (1997), an iterative approach for solving job shop scheduling with lot streaming is developed. This approach solves the problem once with given sublot sizes and once with given processing sequence of sublots in an iterative way. Later, in Chan *et al.* (2004) a procedure using genetic algorithm for solving equal-sized lot streaming job shop problem is introduced. Also, Chan *et al.* (2008) proposed a method based on genetic algorithms and simple dispatching rule to solve assembly job shop scheduling problems with lot streaming. Likewise, Chan *et al.* (2009), developed an approach - named LSGAVS - based on the GA. In this approach, lot sizing and job shop problems are solved simultaneously.

Buscher and Shen (2009) introduced an algorithm comprised of three phases to minimize the makespan of a job shop problem with lot streaming. These three phases encompass the predetermination of sublot sizes (generation of equal sublot sizes), the determination of schedules (based on tabu search), and variation of sublot sizes. Furthermore, Liu *et al.* (2013) investigated the expected benefit of lot streaming in a special case of job shop problems where job values (i.e, profitability

of a job being completed) exponentially deteriorating over time. They applied and compared a fixed number job splitting method (FNJS) and a GA based on job splitting approach (GAJS) to maximize the total value of the jobs. Their proposed GA determines the number of sublots and their magnitudes. In Wong and Ngan (2013), two hybrid evolutionary algorithms, namely, HGA and hybrid particle swarm optimization (HPSO) are introduced and compared as solution procedures for an assembly job shop problem with lot streaming. Moreover, in Defersha and Chen (2009), and specifically in Defersha and Chen (2012), a more comprehensive flexible job shop problem with lot streaming is solved using the GA by considering (1) unequal lot sizes, (2) sequence-dependent set-up time,(3) attached/detached set ups, (4) machine release dates, and (5) lag time. Their proposed approach constitutes the basis of the work in this thesis. Furthermore, an interested reader is referred to Cheng *et al.* (2013) for a comprehensive review of lot streaming in scheduling problems.

2.4. Concluding Remarks

As was discussed earlier, slow convergence is one major weakness in a pure GA. While the GA is able to find most promising regions in the solution space, other optimization methods should be incorporated within the GA in order to find near-optimal solutions or optimal solutions in promising regions with reasonable time consumption. Hence, we used linear programming (LP) as assistance to the GA and achieved a significant improvement in computational time. The technique which is used to reduce looping in the proposed LP and the way LP is incorporated with the GA in the whole solution procedure are two distinctive features of this novel approach. In this approach, LP is used both in the initial population and during the genetic search process in order to improve promising solutions. The role of LP is to determine the optimal values of the continuous

variables corresponding to the values of the integer variables of these promising solutions.

Chapter 3

Mathematical Model

The main objective of this thesis is to present a linear programming assisted GA to solve the flexible job shop problem with lot streaming presented in Defersha and Chen (2012). As stated before, classical JSP is a strongly NP-hard problem (Garey *et al.*, 1976). FJSP is an extension of JSP in the sense that it requires the additional decision of assigning of operations to the machines (routing sub-problem), as well as the decision of sequencing of operations on the machines (scheduling sub-problem). Hence, FJSP not only incorporates all the complexities of it predecessor JPS but also it is a more complex problem. Thus, it can be easily concluded that FJSP is also in the class of NP-hard problems (Xia and Wu, 2005).

Traditional optimization methods are incapable of tackling large-sized NP-hard problems due to the high computational complexity. However, mathematical programming formulation is the key step to develop an efficient heuristic for these problems. In this chapter, for the sake of better comprehension of the mathematical model presented in this thesis, previously developed mathematical formulations in literature for JSP, FJSP and FJSP with sequence-dependent setup time are reviewed. Finally, at the end of this chapter, the problem in this thesis (FJSP-LS) is described, and the corresponding mathematical model is presented.

3.1. Preliminary study

According to Demir and Kürşat İşleyen (2013), mathematical models for JSP and FJSP can be distinguished based on the type of binary variable that they are dependent on for determining the sequence of operations on the machines. These binary variables are categorized into sequence-position variables, precedence variables and time-indexed variables. These three types of variables are developed by Wagner (1959), Manne (1960) and Bowman (1959), respectively. In this section, three broadly used MILP models in literature for FJSP are presented which are adopted from Demir and Kürşat İşleyen (2013). The first MILP model is placed in the category of sequence-based models, and the remaining two MILP models are placed in the category of precedence-based models. Furthermore, at the end of this section, a widely used MILP model in literature for FJSP with sequence dependent setup time is provided.

3.1.1. Notations for introductory mathematical models

Here, problem description and notations for review of the mathematical models in literature are provided. Consider a job shop consisting of M machines and a total number of J independent jobs needing to be scheduled in the system. Each 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. All the machines are available at time zero. Also, preemption is not permitted, and no setup times are considered for the machines. In all models, the objective is to minimize the makespan of the schedule. We next introduce the notations used in the following introductory models.

Parameters:

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

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

$M_{o,j}$ A set of eligible machines of operation o of job j;

E_m A set of operations, which can be performed on machine m; and

Ω Large positive number.

Variables:

Continuous Variables:

c_{max} Makespan of the schedule;

$c_{o,j,m}$ Completion time of operation o of job j on machine m;

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

$q_{o,j,m}$ Starting time of operation o of job j on machine m;

$w_{o,j}$ Starting time of operation o of job j;

$t_{o,j}$ Processing time of operation o of job j, after select a machine; and

$\hat{q}_{r,m}$ Starting time of production run r of machine m.

Binary Integer Variables:

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

$y_{m,o,j}$ A binary variable which takes the value 1 if operation o of job j is performed on machine m, 0 otherwise; and

$v_{o,j,o',j',m}$ A binary variable that takes the value 1 if the o^{th} operation of job j has precedence over o'^{th} operation of job j' on machine m, 0 otherwise.

3.1.2. Sequence-position variable based models for FJSP

These type of models which were first proposed by Wagner (1959) are based on the concept that machine m has a fixed number of production runs (positions)

R_m (where $r = 1, \ \ldots, \ R_m$) and each of these production runs can be assigned at most to one job; thus, the assignment of operations to production runs of a given machine determines the sequence of the jobs on that machine. The following model is originally adopted from Fattahi *et al.* (2007).

MILP Model A

Minimize:

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

Subject to:

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

$$\sum_{m} T_{o,j,m} \cdot y_{m,o,j} = t_{o,j} \ ; \quad \forall (o,j) \tag{3.3}$$

$$w_{o,j} + t_{o,j} \leq w_{o+1,j} \ ; \quad \forall (o,j) | (o < O_j) \tag{3.4}$$

$$\hat{q}_{r,m} + t_{o,j} \cdot x_{r,m,o,j} \leq \hat{q}_{r+1,m} \ ; \quad \forall (r,o,j,m) | (r < R_m) \tag{3.5}$$

$$\hat{q}_{r,m} \leq w_{o,j} + (1 - x_{r,m,o,j}) \cdot \Omega \ ; \quad \forall (r,m,o,j) \tag{3.6}$$

$$\hat{q}_{r,m} + (1 - x_{r,m,o,j}) \cdot \Omega \geq w_{o,j} \ ; \quad \forall (r,m,j,o) \tag{3.7}$$

$$y_{m,o,j} \leq P_{o,j,m} \ ; \quad \forall (o,j,m) \tag{3.8}$$

$$\sum_{j} \sum_{o} x_{r,m,o,j} = 1 \ ; \quad \forall (r,m) \tag{3.9}$$

$$\sum_{m} y_{m,o,j} = 1 \ ; \quad \forall (o,j) \tag{3.10}$$

$$\sum_{r} x_{r,m,o,j} = y_{m,o,j} \ ; \quad \forall (o,j,m) \tag{3.11}$$

$$w_{o,j}, \ t_{o,j} \text{ and } \hat{q}_{r,m} \text{ are greater than equal zero} \tag{3.12}$$

$$x_{r,m,o,j} \text{ and } y_{m,o,j} \text{ are binary} \tag{3.13}$$

The constraint in Eq. (3.2), along with the objective function, determine the makespan. The constraint given in Eq. (3.3) determines the processing time of operation o of job j by the assigned machine. The constraint in Eq. (3.4) is to enforce the requirement that each operation $o+1$ of job j cannot be started before the completion time of operation o of that job. The constraint in Eq. (3.5) states that machine m can only process one operation at a time. The constraints in Eqs. (3.6) and (3.7) together state that the start time of the o^{th} operation of job j is equal to the start time of the r^{th} run of machine m if production run r is assigned to operation o. The constraint given in Eq. (3.8) is for defining the eligible machines for each operation. The constraint in Eq. (3.9) states that each operation can be assigned to exactly one production run of a machine. The constraint in Eq. (3.11) enforces the restrictive assumption that job j can be processed exactly on one machine at a time. Finally, in the constraint given in 3.12, the logical relation between the binary variable $y_{m,o,j}$ and $x_{r,m,o,s,j}$ is depicted.

3.1.3. Precedence variable based models for FJSP

These type of FJSP models are based on precedence variable $v_{o,j,o',j',m}$, which was first introduced by Manne (1960). This variable is employed to indicate the sequence of operations, which are assigned on the same machine. If $v_{o,j,o',j',m}$ takes the value 1, then operation o of job j has precedence over operation o' of job j' on machine m. However, it is not necessary for operation o' of job j' to be performed immediately after the completion of operation o of job j. In this section two precedence-variable based models which are broadly used in literature are presented.

MILP Model B

This kind of MILP model was first devised by Gao *et al.* (2006), and it mainly relies on completion time of operations $c_{o,j}$ and precedence variable $v_{o,j,o',j',m}$.

Minimize:

$$Objective = c_{max} \tag{3.14}$$

Subject to:

$$c_{max} \geq c_{o,j} \ ; \quad \forall(o,j) \tag{3.15}$$

$$c_{o,j} - c_{o-1,j} \geq T_{o,j,m} \cdot y_{m,o,j} \ ; \quad \forall(o,j,m)|(o>1) \tag{3.16}$$

$$c_{1,j} \geq T_{1,j,m} \cdot y_{m,1,j} \ ; \quad \forall(j,m) \tag{3.17}$$

$$\begin{gathered}(c_{o',j'} - c_{o,j} - T_{o',j',m}) \cdot y_{m,o,j} \cdot y_{m,o',j'} \cdot v_{o,j,o',j',m} \geq 0 \ ; \\ \forall(o,j,o',j',m)|(m \in (M_{o,j} \cap M_{o',j'}))\end{gathered} \tag{3.18}$$

$$\begin{gathered}(c_{o,j} - c_{o',j'} - T_{o,j,m}) \cdot y_{m,o,j} \cdot y_{m,o',j'} \cdot v_{o',j',o,j,m} \geq 0 \ ; \\ \forall(o,j,o',j',m)|(m \in (M_{o,j} \cap M_{o',j'}))\end{gathered} \tag{3.19}$$

$$\sum_{k \in M_{o,j}} y_{m,o,j} = 1 \ ; \quad \forall(o,j) \tag{3.20}$$

$$v_{o,j,o',j',m} + v_{o',j',o,j,m} = y_{m,o,j} \cdot y_{m,o',j'} \ ; \quad \forall(o,j,o',j',m)|(m \in (M_{o,j} \cap M_{o',j'})) \tag{3.21}$$

$$c_{o,j} \text{ is greater than equal zero} \tag{3.22}$$

$$y_{m,o,j} \text{ is binary} \tag{3.23}$$

The constraint in Eq. (3.15), along with the objective function, determines the makespan. The constraint in Eq. (3.16) makes sure that the processing sequence of operations for each job corresponds to the prescribed order. The constraint in Eq. (3.17) ensures that the completion time of the fist operation of job j is always greater than its processing time. The constraints in Eqs.

(3.18) and (3.19) together state that for any operation pair o and o', two possible constraints exist, namely, operation o should not be started before the completion of operation o' ,and vice versa. Since, either one or the other constraint must hold, they are referred to as disjunctive constraints. The constraint given in Eq. (3.20) is to enforce the restrictive assumption that each operation must be assigned only to one machine. The constraint in Eq. (3.21) states that only one precedence relation can be chosen for operation pair o and o'.

MILP Model C

This type of model was first proposed by Kim and Egbelu (1999) to formulate FJSP. This model is mainly based on variable $q_{o,j,m}$, the starting time of operation o of job j on machine m, and the precedence variable $v_{o,j,o',j',m}$.

Minimize:

$$Objective = c_{max} \tag{3.24}$$

Subject to:

$$c_{max} \geq w_{o,j} + \sum_{m} T_{o,j,m} \cdot y_{m,o,j} \ ; \quad \forall (o,j) \tag{3.25}$$

$$w_{o,j} + \sum_{m} T_{o,j,m} \cdot y_{m,o,j} \leq w_{o+1,j} \ ; \quad \forall (o,j) | (o < O_j) \tag{3.26}$$

$$\sum_{k \in M_{o,j}} y_{m,o,j} = 1 \ ; \quad \forall (o,j) \tag{3.27}$$

$$q_{o,j,m} = w_{o,j} \cdot y_{m,o,j} \ ; \quad \forall (m,o,j) \tag{3.28}$$

$$\begin{gathered} q_{o,j,m} + T_{o,j,m} \cdot y_{m,o,j} - \Omega \cdot (1 - v_{o,j,o',j',m}) \leq q_{o',j',m} \ ; \\ \forall (o,j,o',j',m) | ((o_{o,j} \neq o_{o',j'}) \wedge (o,o' \in E_m)) \end{gathered} \tag{3.29}$$

$$\begin{gathered} q_{o',j',m} + T_{o',j',m} \cdot y_{m,o',j'} - \Omega \cdot (1 - v_{o',j',o,j,m}) \leq q_{o,j,m} \ ; \\ \forall (o,j,o',j',m) | ((o_{o,j} \neq o_{o',j'}) \wedge (o,o' \in E_m)) \end{gathered} \tag{3.30}$$

$$v_{o,j,o',j',m} + v_{o',j',o,j,m} = y_{m,o,j} \cdot y_{m,o',j'} \ ; \\ \forall (o,j,o',j',m) | ((o_{o,j} \neq o_{o',j'}) \wedge (o,o' \in E_m)) \tag{3.31}$$

$$w_{o,j} \ \ and \ q_{o,j,m} \ \text{ are greater than equal zero} \tag{3.32}$$

$$v_{o,j,o',j',m}, \ v_{o',j',o,j,m}, \ y_{m,o,j} \text{ and } y_{m,o',j'} \text{ are binary} \tag{3.33}$$

The constraint in Eq. (3.25), along with the objective function, determine the makespan. The constraint in Eq. (3.26) is to make sure that the processing sequence of operations for each job corresponds to the prescribed order. The constraint in Eq. (3.27) forces each operation to be assigned only to one machine. In constraint given in Eq. (3.28), the logical relation between the variable $q_{o,j,m}$ and $w_{o,j}$ is presented. The constraints in Eqs. (3.29) and (3.30) determine the orientation of operation pair o and o', if both of these operations are processed on the same machine, and also force each machine to process only one operation at a time. The constraint in Eq. (3.31) states that only one precedence relation can be chosen for operation pair o and o'.

3.1.4. Time-indexed models for FJSP

These type of models are based on the time-index variables, which are first proposed by Bowman (1959). In this approach, operations of jobs are assigned to time periods of eligible machines. The application of this method for FJSP with unparallel machines is extremely limited in literature. Thus, we skip the full description of these type of models. Lastly, the other existing formulations in literature are the hybrid combination of the three aforementioned approaches. An interested reader is encouraged to see Demir and Kürşat İşleyen (2013); Roshanaei *et al.* (2013) for additional details regarding JSP and FJPS mathematical models.

3.1.5. FJSP model with sequence-dependent setup time

Review of the literature reveals that almost all of the proposed MILP models for FJSP with sequence-dependent setup time except the proposed model in Defersha and Chen (2010b) lie in the category of precedence-variable based models (see for example Imanipour (2006); Low and Wu (2001)). In this section, a precedence-variable based MILP model which is proposed in Saidi and Fattahi (2007) is presented. In addition to notations described in section 3.1.1, two extra variables are required for the following model.

In the following formulation, the setup time for an operation o of job 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. Also binary variable $\hat{v}_{o,j,o',j',m}$ takes the value 1 if the $o'^{\,th}$ operation of job j' is the operation to be processed immediately after the completion of operation o of job j on machine m , otherwise, it takes the value zero.

MILP Model D

Minimize:

$$Objective = c_{max} \tag{3.34}$$

Subject to:

$$c_{max} \geq c_{o,j} \ ; \quad \forall(o,j) \tag{3.35}$$

$$w_{o,j} + T_{o,j,m} \cdot y_{m,o,j} \leq c_{o,j} \ ; \quad \forall(m,o,j) \tag{3.36}$$

$$c_{o,j} \leq w_{o+1,j} \ ; \quad \forall(o,j)|(o < O_j) \tag{3.37}$$

$$w_{o,j} + T_{o,j,m} + S_{o,j,m,o',j'} \leq w_{o',j'} + \Omega \cdot (1 - \hat{v}_{o,j,o',j',m}) \ ; \quad \forall(o,j,o',j',m) \tag{3.38}$$

$$c_{o,j} + S_{o,j,m,o',j'} \leq w_{o+1,j} + \Omega \cdot (1 - \hat{v}_{o',j',o+1,j,m}) \ ; \quad \forall(o,j,o',j',m)|(o < O_j) \tag{3.39}$$

$$y_{m,o,j} \leq P_{o,j,m} \; ; \quad \forall (o,j,m) \tag{3.40}$$

$$\sum_{m} y_{m,o,j} = 1 \; ; \quad \forall (m,o,j) \tag{3.41}$$

$$\sum_{o} \sum_{j} \hat{v}_{o,j,o',j',m} = y_{m,o',j'} \; ; \quad \forall (o',j',m) \tag{3.42}$$

$$\sum_{o'} \sum_{j'} \hat{v}_{o,j,o',j',m} = y_{m,o,j} \; ; \quad \forall (o,j,m) \tag{3.43}$$

$$\hat{v}_{o,j,o,j,m} = 0 \; ; \quad \forall (o,j,m) \tag{3.44}$$

$$w_{o,j} \quad \text{and} \quad c_{o,j} \text{ are greater than equal zero} \tag{3.45}$$

$$\hat{v}_{o,j,o',j',m} \text{ and } y_{m,o,j} \text{ are binary} \tag{3.46}$$

The constraint in Eq. (3.35), along with the objective function, determine the makespan. The constraints in Eqs. (3.36) and (3.37) are to make sure that the processing sequence of operations for each job corresponds to the prescribed order. The constraints in Eqs. (3.38) and (3.39) together ensure that only one operation at a time is processed on a machine and also consider the setup time. The constraint in Eq. (3.40) enforces each operation to be assigned only to the eligible machines. Constraint Eq. (3.41) states that only one machine from set of alternative machines can be selected for operation o of job j. The constraints in Eqs. (3.42) and (3.43) together define the circular permutations of operations on each machine. It means that the constraint given in Eq. (3.42) picks exactly operation o' of job j' that has precedence over operation o of job j on machine m, and the constraint in Eq. (3.43) picks exactly operation o of job j that immediately follows operation o' of job j' on machine m. The circular permutations of operations generates the processing sequence of operations on each machine.

3.2. Problem Description and Notations

In this section, a problem description and notions for FJSP-LS problem in this thesis are provided. 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 previous schedules and each machine m (where $m = 1, \ldots, M$) has a release date D_m at which time it will be available for next schedule. Consider also a total number 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 an operation o of a sublot of job j on machine m. An 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 simply 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 it is a detached setup and machine m is available for setup. The problem is to determine the size of each sublot, to assign the operation of each sublot to one of the eligible machines and to 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 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 sublot. Thus the assignment of the operations to production runs of a given machine determines the sequence of the sublots on that machine;

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

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

Ω Large positive number.

Variables:

Continuous Variables:

c_{max} Makespan of the schedule;

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

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

$b_{s,j}$ 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; and

$z_{r,m}$ 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;

3.3. 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:

$$Objective = c_{max} \tag{3.47}$$

Subject to:

$$c_{max} \geq c_{o,s,j,m} \ ; \quad \forall(o,s,j,m) \tag{3.48}$$

$$\widehat{c}_{r,m} \geq c_{o,s,j,m} + \Omega \cdot x_{r,m,o,s,j} - \Omega \ ; \quad \forall(r,m,o,s,j) \tag{3.49}$$

$$\widehat{c}_{r,m} \leq c_{o,s,j,m} - \Omega \cdot x_{r,m,o,s,j} + \Omega \ ; \quad \forall(r,m,o,s,j) \tag{3.50}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} - \Omega \cdot x_{1,m,o,s,j} + \Omega \geq D_m \ ; \quad \forall(m,o,s,j) \tag{3.51}$$

$$\widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j}) + 2\Omega \geq \widehat{c}_{r-1,m} \ ;$$
$$\forall(r,m,o,s,j,o',j') | (r > 1) \tag{3.52}$$

$$\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \cdot A_{o,j} - \Omega \cdot (x_{1,m,o,s,j} + x_{r',m',o-1,s,j}) + 2\Omega \geq \widehat{c}_{r',m'} + L_{o,j} \ ;$$
$$\forall(m,r',m',o,s,j) | \{((1,m) \neq (r',m')) \wedge (o > 1)\} \tag{3.53}$$

$$\widehat{c}_{r',m'} + L_{o,j} \;\leq\; \widehat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m,o',j'} \cdot A_{o,j} \;-$$
$$- \;\Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j}) + 3\Omega \;;$$
$$\forall (r, m, r', m', o, s, j, o', j') | \{(r > 1) \wedge (o > 1) \wedge (r, m) \neq (r', m') \wedge (o, j) \neq (o', j')\} \tag{3.54}$$

$$y_{r,m,o,j} \leq P_{o,j,m} \;;\quad \forall (r, m, o, j) \tag{3.55}$$

$$y_{r,m,o,j} = \sum_{s=1}^{S_j} x_{r,m,o,s,j} \;;\quad \forall (r, m, o, j) \tag{3.56}$$

$$\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,s,j} = \gamma_{s,j} \;;\quad \forall (o, s, j) \tag{3.57}$$

$$b_{s,j} \leq B_j \cdot \gamma_{s,j} \;;\quad \forall (s, j) \tag{3.58}$$

$$\gamma_{s,j} \leq b_{s,j} \;;\quad \forall (s, j) \tag{3.59}$$

$$\sum_{s=1}^{S_j} b_{s,j} = B_j \;;\quad \forall (j) \tag{3.60}$$

$$\sum_{j=1}^{J} \sum_{s=1}^{S_j} \sum_{o=1}^{O_j} x_{r,m,o,s,j} = z_{r,m} \;;\quad \forall (r, m) \tag{3.61}$$

$$z_{r+1,m} \leq z_{r,m} \;;\quad \forall (r, m) \tag{3.62}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} \;;\quad \forall (r, r', m, o, o', s, j) | \{(o' > o) \wedge (r' < r)\} \tag{3.63}$$

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} \;;\quad \forall (r, r', m, o, o', s, j) | \{(o' < o) \wedge (r' > r)\} \tag{3.64}$$

$$x_{r,m,o,s,j},\; y_{r,m,o,j},\; \gamma_{s,j} \text{ and } z_{r,m} \text{ are binary} \tag{3.65}$$

The objective function in Eq. (3.47) is to minimize the makespan of the schedule. The constraint in Eq. (3.48), along with the objective function, determines the makespan. The constraints in Eqs. (3.49) and (3.50) together state that the completion time of the o^{th} operation of sublot s of job j is equal to the completion time of the r^{th} run of machine m if this production run is assigned to that particular operation. The starting time of the setup for the first run ($r = 1$) of machine m is given by $\widehat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m}$ if the o^{th} operation of

sublot s of job j is assigned to this first run. This starting time cannot be less than the release date of machine D_m as enforced by the constraint in Eq. (3.51). The constraint in Eq. (3.52) is to enforce the requirement that the setup of any production run $r > 1$ of a given machine cannot be started before the completion time of run $r - 1$ of that machine. The constraint in Eq. (3.53) states that for any pair of machines (m, m'), the setup (if $A_{o,j} = 1$) or the actual processing (if $A_{o,j} = 0$) of the first run on machine m cannot be started before the completion time of run r' of machine m' plus lag time $L_{o,j}$. This constraint is applied if first run of machine m is assigned to operation o of sublot s of job j and run r' of machine m' is assigned to operation $o - 1$ of this same sublot. The constraint in Eq. (3.54) is similar to that in Eq. (3.53) except that Eq. (3.54) is for run $r > 1$ of machine m. In this case, the sequence dependent setup time has to be considered by taking into account the operation that was processed in run $r - 1$ of machine m. The constraint in Eq. (3.55) states that a production run r of machine m can be assigned to operation o of any one of sublots of job j if this operation can be performed on this machine. Constraint 3.56 depicts the logical relation between the binary variable $y_{r,m,o,j}$ and $x_{r,m,o,s,j}$. If the size of sublot s of job j is positive ($\gamma_{s,j} = 1$), an operation o of this sublot must be assigned to exactly one production run of one machine (Eq. 3.57). However, if the size of this sublot is zero, it should not be assigned to any production run. The constraint in Eq. (3.58) forces the binary variable $\gamma_{s,j}$ to take the value 1 if the sublot size $b_{s,j}$ is greater than zero. If the sublot size $b_{s,j} = 0$, the binary variable $\gamma_{s,j}$ is forced to take the value 0 by the constraint in Eq. (3.59). The constraint in Eq. (3.60) states that the sum of the sizes of the sublots of job j equals the batch size of this job. Each production run of a given machine can be assigned to at most one operation (Eq. 3.61), and production run $r + 1$ can be assigned to an operation if and only if run r of that machine is already assigned (Eq. 3.62). The constraints given in Eqs. (3.63) and (3.64) are used to speed up the branch

and bound procedure in solving small size problems. These constraint sets are not required to model the problem as the relations have been imposed by the constraints in Eqs. (3.53) and (3.54). The constraint in Eq. (3.63) accounts for the fact that if an operation o of sublot s of job j is assigned to a production run r of machine m, any upcoming operation o' of this sublot cannot be assigned to any earlier run r' of machine m. The constraint in Eq. (3.64) is a mirror image of constraint Eq. (3.63). It states that if an operation of a sublot of a given job is assigned to a production run of a machine, any earlier operation of that sublot cannot be assigned to any upcoming production run of that machine. Integral requirements on the variable $x_{r,m,o,s,j}$, $y_{r,m,o,j}$, $\gamma_{s,j}$ and $z_{r,m}$ are given Eq. (3.65).

Chapter 4

The Proposed Algorithm

A parallel pure genetic algorithm was developed in Defersha and Chen (2012) to solve the FJSP-LS model presented in the previous section. In this section, we present a way of combining the pure genetic algorithm with linear programming to create an efficient sequential hybrid algorithm. The resulting hybrid algorithm utilizes a single computational resource and still outperforms or performs equally with the resource-intensive parallel pure genetic algorithm. The following sections convey the common and distinct features of both the pure and the hybrid genetic algorithms.

4.1. Pure Genetic Algorithm

A Genetic algorithm evolves a population of individuals. Each individual serves as a candidate solution of the problem to be solved. In this section, various elements of the pure genetic algorithm developed in Defersha and Chen (2012) are presented. Moreover, the description of the main components of the genetic algorithm developed in Defersha and Chen (2010b) for FJSP is also demonstrated in some subsections to better understand the evolution of the algorithm. This will help the reader to see how the GA for FJSP is developed to be employed in

the FJSP-LS problem.

Solution representation for FJSP

The first key step in genetic algorithm implementation is to construct a proper representation scheme for a particular problem. In this subsection, firstly, we have brief review of the solution representation addressed in Defersha and Chen (2010b) for FJS problem. Then, the solution representation technique which is used in this thesis for FJSP-LS is fully discussed. In applying genetic algorithm to FJS problems, Chen *et al.* (1999); Kacem (2003); Pezzella *et al.* (2008); Gao *et al.* (2008); Zhang *et al.* (2011); Wang *et al.* (2013) used solution representations, capable of encoding both assignment and sequencing of operations on the various machines. Here, a solution representation, which is used in Defersha and Chen (2010b) for FJS problem with sequence dependent setup time is presented. In this representation, every gene in the chromosome is represented by a triplet (j, o, m) denoting the assignment of the o^{th} operation of job j to machine m. The sequence of the genes in the chromosome expresses the sequences of the operations on every machine. The assignment of feasible operations to the machines and a job's operation processing sequence for the small example in Table 4.1 is illustrated in the Figure 4.1. It is noteworthy to point out that the solution representation for classical JSP is very similar to FJSP, with this difference, that every gene in JSP is represented by a twin (j, o).

Table 4.1: An example small flexible job-shop problem (FJSP)

		Set of eligible machines for operation		
Job	No. of Operations	$o1$	$o2$	$o3$
$j1$	3	$\{m1, m2\}$	$\{m3\}$	$\{m2, m4\}$
$j2$	2	$\{m3, m4\}$	$\{m2\}$	
$j3$	3	$\{m3\}$	$\{m2, m4\}$	$\{m1, m3\}$

1	2	3	4	5	6	7	8	
3, 1, 3	**1, 1, 2**	**2, 1, 3**	**1, 2, 3**	**3, 2, 2**	**2, 2, 2**	**1, 3, 4**	**3, 3, 1**	***j, o, m***

***j* = job index, *o*= operations index, *m*=machine index**

Figure 4.1: Solution representation for solving the FJSP using GA

Solution representation for FJSP-LS

In the previous subsection, the representation scheme for applying GA for FJSP was introduced. Comparable representations can be used in solving the FJSP-LS if each sublot is considered as a job and an extra representation is augmented to encode the size and the number of sublots of each job. A way to understand this representation is to think of a situation in which four machines are processing three jobs in a flexible job shop system. The properties of this system, including the number of operations, the maximum number of sublots for each job, and the set of eligible machines for each operation are given in Table 4.2. Using the method introduced in Kacem (2003), assignment of feasible operations to the machines and a job's operation processing sequence can be encoded in a chromosome as depicted in Figure 4.2. Every sublot is considered as a job, and each gene in the chromosome is represented by a quadruple ($j,\ s,\ o,\ m$) denoting the assignment of the o^{th} operation of sublot s of job j to machine m. The sequence of the genes in the chromosome expresses the sequences of the operations on every machine. For instance, through looking to the genes from left to right the assignment and sequencing of operations on machine-1 can be interpreted as follows: $(j1, s3, o1)\rightarrow(j3, s2, o3)\rightarrow(j3, s3, o3)$. These data are obtained from the genes at locations 10, 22 and 23 on the chromosome where $m = 1$. The assignment of operations to the other machines and their sequences as decoded from the chromosome is given in Table 4.3. In this solution representation, in order to

ensure that precedence requirement of the operations of a particular sublot are not violated, for a given j and s, the gene (j, s, o, m) always lies to the right hand side of all the other genes (j, s, o', m') having $o' < o$.

Table 4.2: An example small flexible job-shop problem with lot streaming (FJSP-LS)

			Set of eligible machines for operation		
Job	No. of Operations	Max No. of Sublots	$o1$	$o2$	$o3$
$j1$	3	3	$\{m1, m2\}$	$\{m3\}$	$\{m2, m4\}$
$j2$	2	2	$\{m3, m4\}$	$\{m2\}$	
$j3$	3	3	$\{m3\}$	$\{m2, m4\}$	$\{m1, m3\}$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
3, 2, 1, 3	**1, 2, 1, 2**	**2, 1, 1, 3**	**3, 1, 1, 3**	**2, 2, 1, 4**	**3, 2, 2, 4**	**3, 3, 1, 3**	**1, 3, 1, 1**	**3, 1, 2, 2**	**1, 1, 1, 2**	**3, 3, 2, 2**	**1, 2, 2, 3**	**1, 1, 2, 3**	**3, 1, 3, 3**	**1, 3, 2, 3**	**2, 1, 2, 2**	**1, 1, 3, 4**	**2, 2, 2, 2**	**1, 3, 3, 4**	**3, 2, 3, 1**	**3, 3, 3, 1**	**1, 2, 3, 2**	***j, s, o, m***

***j* = job index, *s* = sublot index, *o* = operations index, *m* = machine index**

Figure 4.2: Representation of the assignment of operations to machines and their sequencing

Table 4.3: Operation assignment and sequencing decoded from Figure 4.2

	Operation assigned to production run							
Machine	$r1$	$r2$	$r3$	$r4$	$r5$	$r6$	$r7$	$r8$
$m1$	$(j1, s3, o1)$	$(j3, s2, o3)$	$(j3, s3, o3)$					
$m2$	$(j1, s2, o1)$	$(j3, s1, o2)$	$(j1, s1, o1)$	$(j3, s3, o2)$	$(j2, s1, o2)$	$(j2, s2, o2)$	$(j1, s2, o3)$	
$m3$	$(j3, s2, o1)$	$(j2, s1, o1)$	$(j3, s1, o1)$	$(j3, s3, o1)$	$(j1, s2, o2)$	$(j1, s1, o2)$	$(j3, s1, o3)$	$(j1, s3, o2)$
$m4$	$(j2, s2, o1)$	$(j3, s2, o2)$	$(j1, s1, o3)$	$(j1, s3, o3)$				

In order to solve the discussed FJSP-LS model using GA, it is essential to include the number of sublots for each job and their sizes into our solution representation. Whereas, the chromosome in Figure 4.2 is capable only of encoding the assignment and sequencing of the operations of the sublots. Therefore, a left hand side segment (LHS-Segment) has been added to this chromosome as depicted in Figure 4.3. In this segment, every gene is represented by $\alpha_{s,j}$, which takes a random value in the interval [0, 1]. The value each $\alpha_{s,j}$ takes is used in Eq. (4.1) in order to compute the size of the s^{th} sublot of operation o of job j. It is possible for a certain sublot to have a size of zero if its corresponding $\alpha_{s,j}$ has a value equal to zero. In this case, the size of a sublot is computed by dividing the number of parts in a batch of job j (B_j) to the maximum number of sublots of j (S_j). Thus, the maximum and actual numbers of sublots for each job and their sizes are encoded in the LHS-Segment.

After the sublot sizes are determined using the equation Eq. (4.1), some sublot may be found too small. To conserve both setup and processing times, overly small sublots should be omitted because it would be ineffective to spend considerable amounts of time setting up a very small sublot. As a result, after calculating the sublot sizes, any sublot with a size less than degeneration limit d (less than d$\times$ 100% of whole lot size), is considered az zero. This means that the corresponding $\alpha_{s,j}$ of those excessively small sublots are reset to zero and excluded from solution procedure. Subsequently, the Eq. (4.1) is used again to calculate the size of sublots. This process is performed using sublot size degenerator (SSD), operator.

$$b_{s,j} = \begin{cases} \frac{\alpha_{s,j}}{\sum_{s=1}^{S_j} \alpha_{s,j}} \times B_j & ; \text{if} \quad \sum_{s=1}^{S_j} \alpha_{s,j} > 0 \\ \\ B_j / S_j & ; \text{otherwise} \end{cases} \tag{4.1}$$

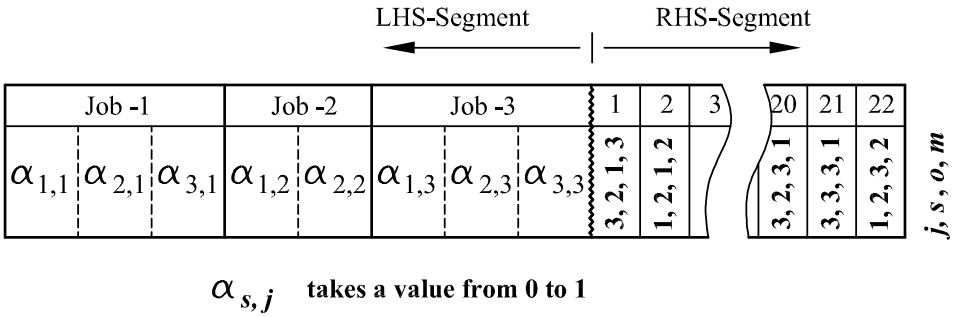

Figure 4.3: Solution representation used in this thesis to solve the FJSP-LS using GA

4.1.1. Selection operator

The process of selecting two parents from the population for reproduction is called selection. The aim of selection in genetic algorithm is to highlight individuals with higher fitness, in hopes that their resulting offsprings are fitter individuals (Sivanandam and Deepa, 2007). Here in this thesis, we used $k - way$ tournament selection operator, which was introduced in Goldberg *et al.* (1989). This selection operator is involved in holding competition among k randomly selected individuals, and choosing the one with the highest fitness (smallest makespan) as a winner of the tournament. The copy of the winner of the competition is then inserted into the mating pool, and all the selected chromosomes are placed back in the mating pool. The procedure is repeated until the size of the mating pool is the same as the current population size.

4.1.2. Crossover operators

After the selection of chromosomes for reproduction, crossover operator is applied to combine the features of two parents in order to produce a new child with a view to enriching the population with better chromosomes (Sivanandam and Deepa, 2007). In this subsection, at the beginning, we provide the description of the crossover operators presented in Defersha and Chen (2010b) for FJS problem. These operators which were first devised in Kacem (2003) are also employed in

the GA used in this thesis for FJSP-LS. Finally, at the end of this subsection, three additional crossover operators, which are required for FJSP-LS are also discussed.

Crossover operators for FJSP

In general, the crossover operators for FJSP can be categorized as assignment or sequence crossover operators. The assignment crossover operators play the role of generating offsprings by exchanging the assignment properties of the mating chromosomes. Operation-to-machine assignment crossover (OMAC) is such an operator. As depicted in Figure 4.4, by using OMAC, two offsprings will be produced from two randomly chosen parent chromosomes, where the sequence of operations in each parent is preserved in its corresponding child. The first step in this approach is to randomly select operations from parent 1. Then, the second step is to copy all the genetic materials of parent 1 to the offspring except the assignment information of the selected operations. The final step is to obtain the assignment properties of the selected operations from parent 2, and then copy them to the corresponding genes in the offspring to produce a new child. The same procedure is repeated to create child 2, but with the difference that the procedure begins from parent 2. It is important to note that assignment crossover operators are not applicable for classical job shop since there exist no alternative routings for operations in JSP.

On the other hand, the role of sequence crossover operators are to produce two new offsprings by exchanging the sequencing properties of parent chromosomes, while the assignment properties of the parents will be inherited to the corresponding offsprings. Job level operation sequence crossover (JLOSC) is such an operator. In this approach, the first step is to randomly select an operation from parent 1. The next step is to copy all genetic materials of the genes, which are associated with the chosen job to the offspring. The final step is to fill the

empty genes in the offspring chromosome, while the machine-assignment information of the empty genes are preserved from parent 1. This is done through copying the remaining operations from parent 2 to empty genes, while operations retain their appearance order from the second parent. The reproduction procedure using this crossover is illustrated in Figure 4.5. Unlike the sequence crossover operators, these types of operators are also applicable for a classical JSP.

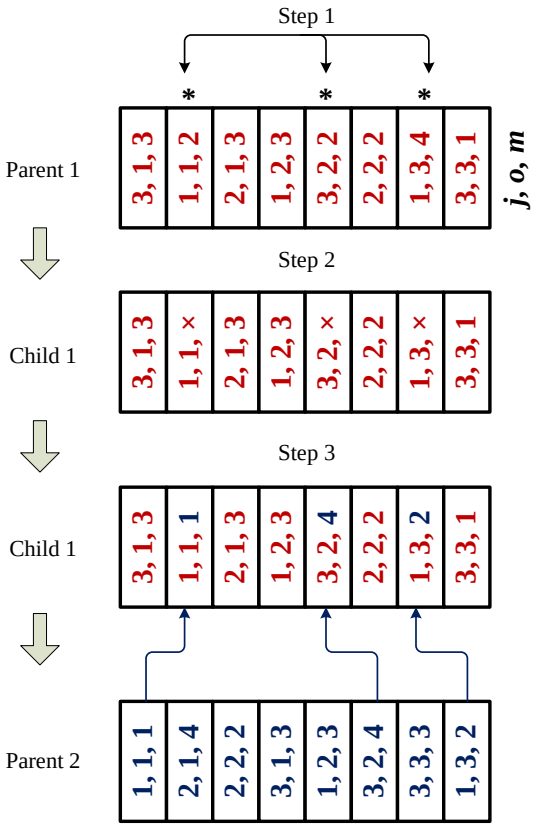

Figure 4.4: Operation-to-machine assignment crossover (OMAC) operator

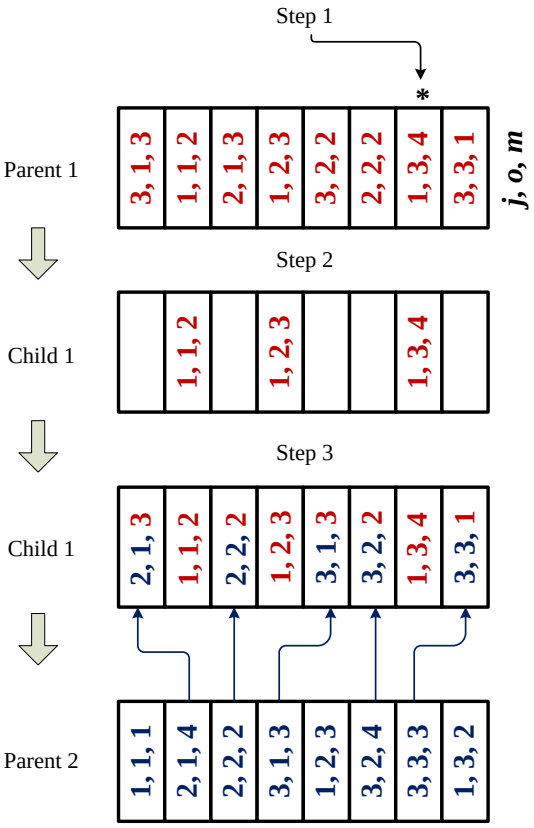

Figure 4.5: Job level operation sequence crossover (JLOSC) operator

Crossover operators for FJSP-LS

All the previously mentioned crossover operators for FJSP are also applied to the RHS segment of the chromosome in FJSP-LS problem in this thesis. However,

three additional crossover operators are required in solving FJSP-LS problem using GA, including two single point crossovers (SPC-1 and SPC-2) and sublot level operations sequence crossover (SLOSC). As illustrated in Figure 4.6, single point crossovers randomly choose a crossover point along the length of the LHS segment of the parent chromosomes. Then, when SPC-1 (SPC-2) is applied, the portion of the LHS segment of the mating chromosomes to the left (right) of the arbitrarily chosen crossover point is exchanged.

SLOSC is almost identical to JLOSC, with one minor difference: in step 2 of SLOSC, only the genetic materials of the arbitrarily chosen genes with the same sublot and job indexes are copied to the offsprings. In the GA used in this thesis for FJSP-LS, an individual chromosome may be subjected to SPC-1, SPC-2, OMAC, JLOSC and SLOSC operators with the probabilities equal to ρ_1, ρ_2, ρ_3, ρ_4 and ρ_5, respectively. It is noteworthy to mention that application of any of the RHS segment specific crossover operators will never violate the precedence constraint of the operations in the newly produced offsprings.

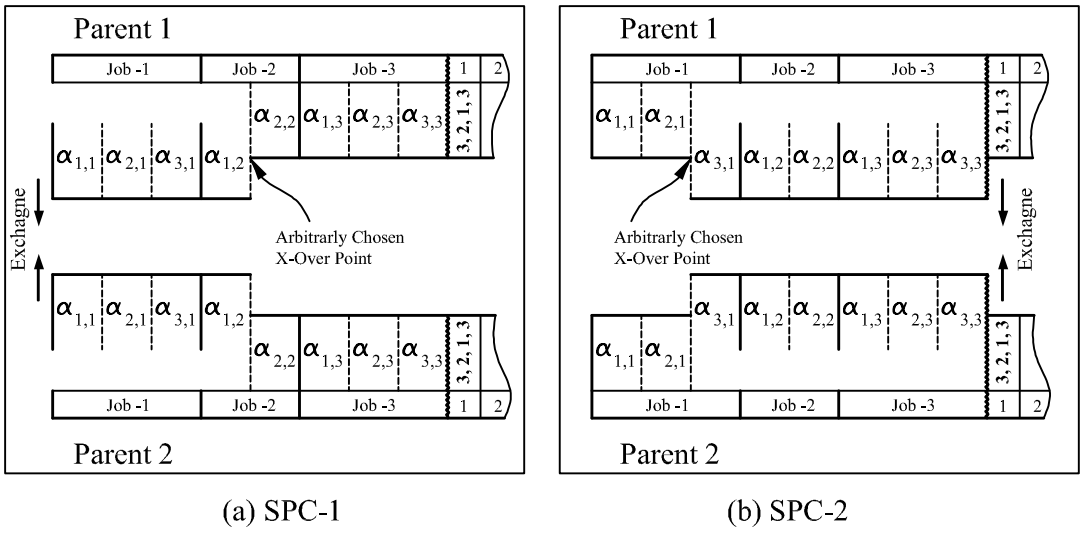

Figure 4.6: Single point crossover operators (SPC-1 and SPC-2)

4.1.3. Mutation operators

After crossover, each obtained offspring may undergo mutation with prespecified probability. The role of mutation operators is to prevent the algorithm from being trapped in local optima, and to maintain genetic diversity in the population. Unlike the role of crossover operator in exploiting the current solution for better ones, mutation operator plays the role of exploring whole search space (Sivanandam and Deepa, 2007). In this subsection, initially, the mutation operators used in Defersha and Chen (2010b) for FJSP are described. These mutation operators which are originally adopted from Kacem (2003) are also applied in the GA used in this thesis for FJSP-LS. Eventually, at the end of this subsection, two additional mutation operators applied in this study are also discussed.

Mutation operators for FJSP

Similar to crossover operators, mutation operators also can be categorized as assignment or sequence mutation operators. The role of assignment mutation operators is to alter the assignment properties of operations in an individual, while the processing sequence of operations on the machines remains unchanged. The random operation assignment mutation (ROAM) is such an operator. ROAM is applied with a small probability on few operations of a given individual chromosome. It changes the assignment property of the selected operation, and assign that operation to one of its alternative machines. Figure 4.7 illustrates the creation of new offsprings using this approach based on the small example in 4.1.

Intelligent operations assignment mutation (IOAM) is another operator in the category of assignment mutation operators. In this approach, an operation on the most loaded machine is reassigned to the least loaded compatible one. The operations sequence shift mutation (OSSM) is in class of sequence mutation operator. Whenever OSSM is applied on an individual, an operation is selected and then moved to another position on the chromosome in such a way that no

precedence constraint is violated.

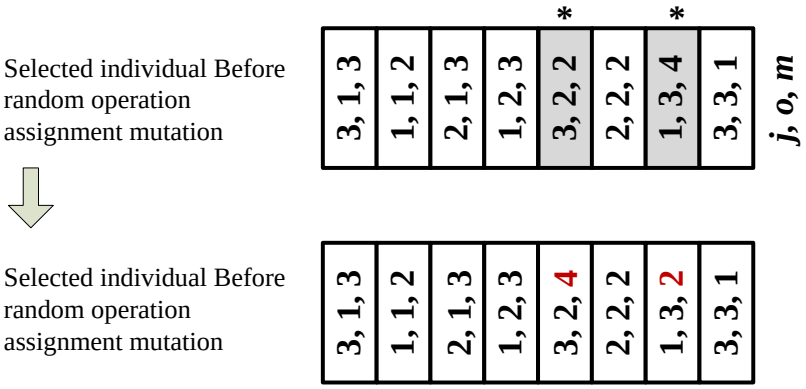

Figure 4.7: Random operation assignment mutation operator

Mutations operators for FJSP-LS

All the previously discussed mutation operators are also applied to the RHS segment of the chromosome in FJSP-LS problem in this thesis. However, three additional mutation operators are applied in the GA used in this thesis for FJSP-LS problem. These operators include sublot step mutation (SStM), sublot swap mutation (SSwM) and sublot size degenerator (SSD).

The operator SStM is applied with small probability σ_1 on each gene $\alpha_{s,j}$ in the LHS segment of each chromosome. Whenever it is applied on a gene, it increases or decreases the value of $\alpha_{s,j}$ using the equations $\alpha_{s,j} = \min\{1, \alpha_{s,j} + \theta\}$ and $\alpha_{s,j} = \min\{0, \alpha_{s,j} - \theta\}$, respectively. In every application of this operator, the step amount θ is calculated using the equation $\theta = \theta_{max} \cdot rand()$, where the value of parameter θ lies between zero and one, and $rand()$ generates random number in range of [0,1]. Another LHS segment specific mutation operator is SSwM, which is applied with small probability σ_2 on a pair of genes with same j index ($\alpha_{s,j}$ *and* $\alpha_{s',j}$), and swaps their values. SSD is another mutation operator for LHS segment of the chromosome. This non-probabilistic operator is employed

to set the value of $\alpha_{s,j} = 0$ if $\alpha_{s,j}/\sum_{s=1}^{S_j} \alpha_{s,j}$ is less than the degeneration limit d. Moreover, in the GA used in this study, an individual chromosome may undergo ROAM, IOAM and OSSM with the probabilities equal to σ_3, σ_4 and σ_5, respectively.

4.1.4. Fitness evaluation in pure GA

The fitness of a chromosome is evaluated using the makespan of the schedule corresponding to that given chromosome. In this subsection, the evaluation procedure used in Defersha and Chen (2012) for the pure GA is elaborated. In our proposed method, whenever we employ pure GA, the same procedure for finding the fitness value of chromosomes is used. When calculating the makespan, following items are considered: (1) the dependence of setup time on the sequence; (2) the nature of the setup, attached or detached; (3) lag time requirement of certain operations; (4) machine release dates; and (5) the possibility of the sizes of certain sublots becoming zero. The fitness evaluation procedure is outlined below.

Step 1. Using the information obtained from the LHS-Segment of the chromosome and Eq. (4.1), calculate the sizes of the sublots of the various jobs.

Step 2. Set $l = 1$

Step 3. Set the values of indices j, s, o and m as obtained from the gene at location l of the RHS-Segment of the chromosome.

Step 4. If $b_{s,j}$ is greater than zero, then go to Step 5; otherwise go to Step 6.

Step 5. Calculate the completion time $c_{o,s,j,m}$

- If (1) operation o of sublot s of job j is the first operation assigned to machine m and (2) $o = 1$, then:
  $c_{o,s,j,m} = D_m + S^*_{o,j,m} + b_{s,j} \cdot T_{o,j,m}$.
- If (1) operation o of sublot s of job j is the first operation assigned to machine m, (2) $o > 1$, and (3) operation o-1 is assigned to machine m', then:
  $c_{o,s,j,m} = \max\{D_m + (1 - A_{o,j}) \times S^*_{o,j,m};\quad c_{o-1,s,j,m'} + L_{o,j}\} + b_{s,j} \times T_{o,j,m} + A_{o,j} \times S^*_{o,j,m}$.
- If (1) operation o' of sublot s' of job j' is the operation to be processed immediately before operation o of job sublot s of j on machine m and (2) $o = 1$, then:
  $c_{o,s,j,m} = c_{o',s',j',m} + S_{o,j,m,o',j'} + b_{s,j} \cdot T_{o,j,m}$.
- If (1) 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, (2) $o > 1$, and (3) operation o-1 is assigned to machine m', then:
  $c_{o,s,j,m} = \max\{c_{o',s',j',m} + (1 - A_{o,j}) \times S_{o,j,m,o',j'};\quad c_{o-1,s,j,m'} + L_{o,j}\} + b_{s,j} \times T_{o,j,m} + A_{o,j} \times S_{o,j,m,o',j'}$.

Step 6. If l is less than the total number of genes of the RHS-Segment of the chromosome, increase its value by 1 and go to Step 3; otherwise go to Step 7

Step 7. Calculate the makespan of the schedule as $c_{max} = \max\{c_{o,s,j,m};\ \forall(o, s, j, m)\}$ and set the fitness of the solution to c_{max}.

The above procedure, in particular Step 5, is based on the property of the chromosomes that, for a given j and s, the gene (j, s, o, m) always lies to the right of all the other genes (j, s, o', m') having $o' < o$. Because of this property

of the chromosome, whenever the completion time of operation (j, s, o, m) on machine m is to be calculated, the completion time of operation $(j, s, o-1, m')$ is already calculated and available, regardless of the machine to which the preceding operation is assigned. Moreover, the completion time of the operation (j', s', o', m) to be processed on machine m immediately before operation (j, s, o, m) will also be calculated and available.

4.2. Linear Programming Subproblem

The MILP model that has been elaborated in section 3.3 is for the purpose of solving FJSP-LS problem on a small scale. It has been used in Defersha and Chen (2012) with the intention of validating the correctness of the pure GA in small-sized problems. However, the MILP model had no role in Pure GA solution procedure. In our proposed algorithm, however, a linear programming subproblem is applied with the aim of determining the optimal values of the continuous variables corresponding to the values of the integer variables of promising solutions. This linear programming (LP) subproblem is formulated based on the MILP model given in section 3.3. In the algorithm proposed here, once GA solves FJPS-Problem, it provides us the information about job assignments, sequence of operations and also the number and size of sublots. Having this information, it enables finding answers to the following questions:

- whether the r^{th} run on machine m is for operation o of sublot s of job j or not;
- whether the r^{th} run on machine m is for operation o of any one of the sublots of job j or not ;
- whether sublot s of job j is non-zero or not; and

- whether the r^{th} potential run of machine m has been assigned to any operation.

Answers to the above questions reveal the values of unknown variables ($x_{r,m,o,s,j}$, $y_{r,m,o,j}$, $\gamma_{s,j}$ and $z_{r,m}$), respectively. As a result, in formulating the new LP model, some of the equations from the main model are omitted as they were composed of only the binary variables. The rest of the equations in MILP model are modified slightly and inserted into the LP subproblem, provided that variable $\gamma_{s,j} = 1$. Also, objective function remains unchanged. In the MILP model, constraints in Eqs. (3.49) and (3.50) were used to set $c_{r,m} = c_{o,s,j,m}$ if variable $x_{r,m,o,s,j} = 1$. Nevertheless, these two equations can be merged together and form Eq. (4.4) in LP model since variable $x_{r,m,o,s,j}$ is known to us. Likewise, knowing the values of $x_{r,m,o,s,j}$, we can replace Eq. (3.51) with Eq. (4.5). In general, by knowing the values of binary variables, all Ω parameters can be removed from Eqs. (3.52), (3.53) and (3.54) which results in Eqs. (4.6), (4.7) and (4.8), respectively. The constraints in Eqs. (3.48) and (3.60) are also applicable in the new LP model; however, constraints in Eqs.(3.58), (3.59), (3.61), (3.62), (3.63) and (3.64) are not required in the LP model, because these equations were originally implemented in the the MILP model to determine binary variables.

LP: given $(x_{r,m,o,s,j},\ y_{r,m,o,j},\ \gamma_{s,j},\ z_{r,m})$ **for all** (r, m, o, s, j)

Minimize:

$$Objective = c_{max} \tag{4.2}$$

Subject to:

$$c_{max} \geq c_{o,s,j,m}\ ;\quad \forall (o, s, j, m) | (\gamma_{s,j} = 1) \tag{4.3}$$

$$\widehat{c}_{r,m} = c_{o,s,j,m}\ ;\quad \forall (r, m, o, s, j) | \{(x_{r,m,o,s,j} = 1) \wedge (\gamma_{s,j} = 1)\} \tag{4.4}$$

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

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

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

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

$$\sum_{\forall s | (\gamma_{s,j} = 1)} b_{s,j} = B_j \; ; \quad \forall (j) \tag{4.9}$$

Once the LP model is solved, the optimum value of continuous variables c_{max}, $c_{o,s,j,m}$, $\widehat{c}_{r,m}$ and $b_{s,j}$ will be determined. The fitness value of each individual is then replaced with the corresponding c_{max}. For each individual, the size of sublot S of Job j is updated by $b_{s,j}$ value obtained from the solver. Afterward, the value of $\alpha_{s,j}$ is updated by dividing the value of the corresponding sublot size by the lot size. It is worth taking into consideration that the optimum value of a particular $b_{s,j}$ may be equal to zero after solving the LP model. In this case, that particular sublot is omitted from solution procedure, and LP model will be solved again.

4.3. Steps of the Algorithm

In the pure GA approach, the right hand side of the chromosome (solution representation) has responsibility to assign jobs to machines and also determine the sequence of operations in each machine. On the other hand, the left hand side of the chromosome has the task of determining the size of sublots for each job. However, in the proposed hybrid GA, while the job assignment and sequence of operations are predetermined by right hand side of chromosome, LP is responsible for lot streaming, and left hand side of the chromosome only provides LP with the information about the number of sublots. As stated before, after determining the sublot sizes using the GA, the sublots with overly small sizes are excluded from solution. Using this information, LP determines the size of the remaining sublots, and updates the corresponding $\alpha_{s,j}$ values in LHS of chromosome. The step of the proposed algorithm are depicted in the flow chart given on the next page. The following notations are used in this flowchart. The steps were coded in C++, and simplex subroutines within ILOG CPLEX package ILOG Inc. (2008) was applied to solve the LP model.

i	Generator counter
p	Population index
i_{max}	Maximum number of generations
PS	Population size

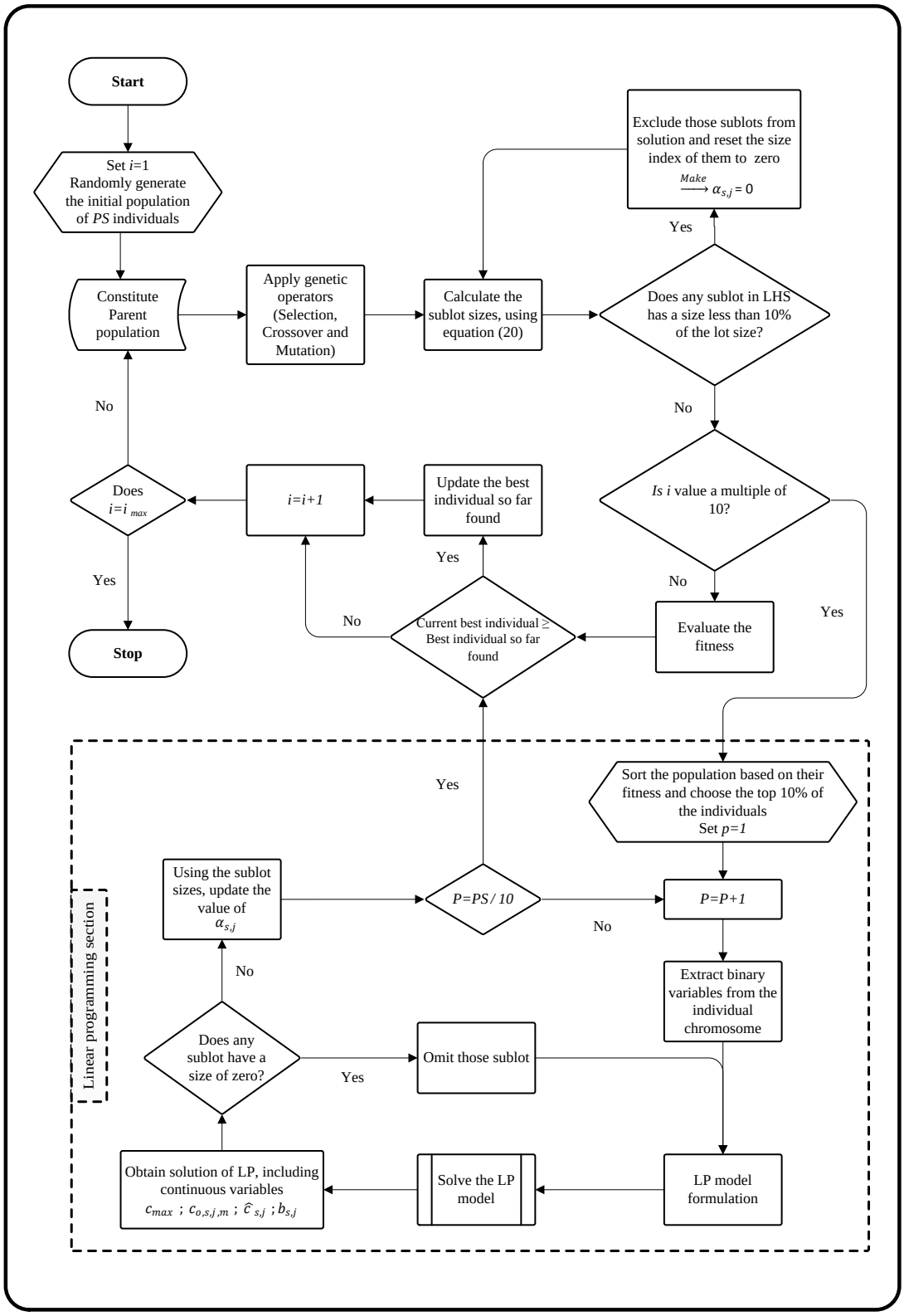

Figure 4.8: Linear programming assisted genetic algorithm flowchart

4.4. Implementation Techniques

The term "assisted" which is used in this thesis has been derived from the way LP is implemented in our approach. In the proposed hybrid GA, LP has not been used in every generation. Instead, it is applied with specific frequency. In addition, before using LP in an iteration, individuals are sorted in ascending order based on their makespan, and LP is applied on only the proportion of population with best fitness values. In this thesis, in every ten iterations, LP is applied on top 10% of population. With this type of application of LP, while the ability of pure GA to find promising regions in a solution space in a relatively short time is preserved, the convergence behavior of pure GA is significantly improved.

In order to use LP in the proposed algorithm, the above LP model needs to be formulated and then implemented in ILOG-CPLEX modelling environment based on a solution generated by GA. It means that in an iteration in which the LP is supposed to be used, the data regarding the job assignments, their sequences and number of sublots are provided by the chromosome, and then based on these data the LP model is formulated and then implemented in the CPLEX software. The method which is used to insert constraints into the CPLEX software can have a great effect on the solution speed. One simple yet lengthy way is, for every possible combination of indexes of an equation, check whether that particular constraint applies to the information obtained from GA, and if it applies that constraint is added to the model in CPLEX software. The following pseudocode in algorithm 1 illustrates how the mentioned method works. We called this implementation method as a direct approach.

As can be seen, using this method for adding constraints into the solver model results in having enormous amount of For-loops which significantly slows up the solution procedure. For a particular equation in LP model, the number of loops which are needed to insert all constraints originated from that equation is equal to the number of possible combinations of indexes in that equation.

Algorithm 1 Direct approach for formulation and implementation of LP

for $m(1) \rightarrow m(MaximumNumberofMachines)$ **do**
  for $j(1) \rightarrow j(MaximumNumberofJobs)$ **do**
    for $s(1) \rightarrow s(MaximumNumberofSublotsForJob.j)$ **do**
      for $o(1) \rightarrow o(MaximumNumberofOperationsForJob.j)$ **do**
        if Operation (1) of Job(j) requires Machine(m) **then** Implement Eq. (4.3)
        end if
        if run(1) on machine(m) is for operation(o) of sublot (s) of job (j) **then** Implement Eq. (4.5) for this j, s, o, m, r
        end if
        for $r(1) \rightarrow r(MaximumNumberofRuns)$ **do**
          if run(r) on machine(m) is for operation(o) of sublot (s) of job (j) **then**
            Eq. (4.4) for this j, s, o, m
          end if
        end for
        for $j'(1) \rightarrow j'(MaximumNumberofJobs)$ **do**
          for $o'(1) \rightarrow o'(MaximumNumberofOperationsForJob.j')$ **do**
            for $r(2) \rightarrow r(MaximumNumberofRuns)$ **do**
              if run(r) on machine(m) is for operation(o) of sublot (s) of job (j) **then**
                if run $(r-1)$ on machine (m) is for operation (o') of any one of the sublots of job (j') **then** Implement Eq. (4.6) for this j, s, o, m, o', j'
                end if
              end if
            end for
          end for
        end for
      end for
      for $o(2) \rightarrow o(MaximumNumberofOperationsForJob.j)$ **do**
        for $m'(1) \rightarrow m'(MaximumNumberofMachines)$ **do**
          for $r'(1) \rightarrow r'(MaximumNumberofRuns)$ **do**
            if run(1) on machine(m) is for operation(o) of sublot (s) of job (j) **then**
              if run(r') on machine(m') is for operation($o-1$) of sublot (s) of job (j) **then**
                Implement Eq. (4.7) for this m, r', m', o, s, j
              end if
            end if
            for $r(2) \rightarrow r(MaximumNumberofRuns)$ **do**
              for $j'(1) \rightarrow j'(MaximumNumberofJobs)$ **do**
                for $o'(1) \rightarrow o'(MaximumNumberofOperationsForJob.j')$ **do**

Algorithm 1 Direct approach for formulation and implementation of LP (cont.)

if run(r) on machine(m) is for operation(o) of sublot (s) of job (j) **then**
if run $(r-1)$ on machine (m) is for operation (o') of any one of the sublots of job (j') **then**
if run(r') on machine(m') is for operation($o-1$) of sublot (s) of job (j) **then** Implement Eq. (4.8) for this $r, m, r', m', o, s, j, o', j'$
end if
end if
end if
end for
end for
end for
end for
end for
end for
end for
end for

In order to clarify the approach mechanism, assume each job has O number of operations and S number of sublots, and maximum number of machines, jobs and runs on the machines are equal to M, J and R, respectively. In this case, the number of FOR-loops required to impellent all constraints in the solver is $M \cdot J \cdot O \cdot S \cdot (2 + R + J^2 \cdot O + R \cdot M + R^2 \cdot M \cdot J \cdot O \cdot S)$. This enormous amount of loops will considerably increase the computational time, and make LP embedding entirely ineffectual.

To alleviate the aforementioned difficulty, we tried to use another method in which the process of implementing the mathematical model is done by reading the information we need from the genes in a chromosome. In this method, we start to scan the genes in RHS of the chromosome from left to right in order to obtain the properties information of every gene. Properties information includes job j, sublot s, operation o, machine m, and production run r data. After getting the properties of a gene, the LP constraints based on these information are inserted into CPLEX solver. We named this approach as an indirect approach. For better

understanding of this method, consider the chromosome in Figure 4.2. Based on this method, the chromosome is scanned from gene g=1 to g=22. Fore gene 1, the corresponding properties (j=3, s=2, o=1 and m=3) are obtained. Because this is the first run on machine m=3, all constraints in the LP model except Eq. (4.6), Eq. (4.7) and Eq. (4.8) are applied to this particular gene. The constraints in Eq. (4.6), and Eq. (4.8) are not for first run of a machine, and constraint in Eq. (4.7) is not for first operation of a job.

Algorithm 2 : Indirect approach for formulation and implementation of LP

for $gene(0) \rightarrow gene(TotalNumberOfProcesses)$ **do**
  Obtain job index of gene(g)
  Obtain sublot index of gene(g)
  Obtain operation index of gene(g)
  Obtain job Machine index of gene(g)
  if Size Index of sublot(s) of job(j)¿ 0 **then**
    Update job, sublot and operation indexes of the current run of machine(m)
    Update machine and run indexes of operation(o) of sublot(s) of job(j)
    Implement Eq. (4.3) and Eq. (4.4) for this r, m, s, o, j
    if run counter of $machine(m) = 1$ **then**
      Implement Eq. (4.5) for this m, s, o, j
      if operation index $o > 1$ **then**
        Update m' and r' , using the machine and run indexes of operation($o-1$) of sublot(s) of job(j)
        Implement Eq. (4.7) for this m, r', m', o, s, j
      end if
    else
      Update o' and j' using operation and job indexes of run($r-1$) of machine(m)
      Implement Eq. (4.6) for this r, m, o, s, j, o', j'
      if operation index $o > 0$ **then**
        Update m' and r' using machine and run indexes of operation($o-1$) of sublot(s) of job(j)
        Implement Eq. (4.8) for this j, s, o, m
      end if
    end if
  end if
  $run \leftarrow run + 1$
end for

This process will repeats for every gene in order to formulate a LP method based on the GA solution. In comparison with previous approach, this method tremendously reduce the number of loops we need to formulate the LP model. For instance, consider small example in Table 4.2, if we assume that maximum number of runs for every machine is 8 runs, then the number of loops needed for adding constraints to the solver model by direct approach is equal to 107864 loops. Whereas, the number of loops required in indirect approach is equal to 22 loops. This significant difference in the number of loops for a small problem implies that in the large problems using second method can considerably reduce computational cost and time. The pseudocode in Algorithm 2 describes the approach which is used in this thesis to implement LP model in CPLEX software.

Chapter 5

Numerical Example

5.1. Model illustration

In this section, we provide several numerical examples with the purpose of showing the advantages of our proposed hybrid GA in comparison with parallel and pure GA methods. As was stated before, the basic difference between the pure and the proposed hybrid GA is in the way lot sizes are determined by these two methods. In order to demonstrate how the application of LP for determining lot sizes has improved our method, a small example of a lot streaming problem is considered. This example consists of processing three jobs in a four-machine flexible job-shop. The batch size of each job and for each operation, the nature of the set up (attached or detached), lag time, alternative machines, and corresponding processing times are given in Table 5.1. The sequence-dependent setup time data are also given in Table 5.2.

This small instance is once solved by the pure GA and once by the proposed hybrid GA, while the job assignment is similar in both solution procedures. Therefore, the use of this example makes it possible to compare the performance of the pure GA and that of the proposed hybrid GA in determining the lot sizes. The sizes of the various sublots and the Gantt charts of the resulting schedules are

given in Figure 5.1. The numerical values of the starting and the ending times of the set ups and operations are presented in Table 5.3. It can be seen from figure 5.1 that despite having the same job assignments, the proposed method achieved a better makespan than did the pure GA. When the pure GA is used for lot streaming, the makespan of this small problem is 22336.3 minutes (Figure 5.1-a). However, when the problem is solved once more with the same job assignments, a makespan of 19823.6 minutes is obtained: this is about an 11% reduction in the makespan from the pure GA results.

Table 5.1: Processing Data for Jobs

					Alternative routes, (m, $T_{o,j,m}$)		
j	B_j	o	$A_{o,j}$	$L_{o,j}$	1	2	3
1	1240	1	na	na	(1, 6.00)	(3, 5.25)	
		2	1	0	(1, 4.50)	(2, 5.00)	
		3	0	0	(1, 2.50)	(3, 2.75)	(4, 2.75)
2	1480	1	na	na	(3, 5.50)	(4, 5.75)	
		2	1	0	(1, 3.50)	(2, 3.25)	(4, 3.50)
3	1290	1	na	na	(1, 4.50)	(4, 4.75)	
		2	1	80	(1, 5.50)	(3, 5.75)	(4, 5.25)
		3	1	0	(1, 6.50)	(3, 6.50)	(4, 6.75)

na = not applicable

Table 5.2: Sequence Dependent Setup Time Data

j	o	m	Setup time $(S^*_{o,j,m})$, $\cdots$,$(j'$, o', $S_{o,j,m,o',j'})$ $\cdots$
1	1	1	(150), (1,1,80), (1,2,160), (1,3,160), (2,2,270), (3,1,270), (3,2,240), (3,3,210)
		3	(200), (1,1,80), (1,3,160), (2,1,210), (3,2,240), (3,3,210)
	2	1	(50), (1,1,120), (1,2,60), (1,3,140), (2,2,150), (3,1,180), (3,2,240), (3,3,300)
		2	(100), (1,2,80), (2,2,180)
	3	1	(150), (1,1,120), (1,2,160), (1,3,60), (2,2,270), (3,1,270), (3,2,270), (3,3,210)
		3	(150), (1,1,140), (1,3,60), (2,1,180), (3,2,210), (3,3,270)
		4	(100), (1,3,60), (2,1,240), (2,2,180), (3,1,270), (3,2,270), (3,3,270)
2	1	3	(200), (1,1,180), (1,3,300), (2,1,70), (3,2,240), (3,3,270)
		4	(100), (1,3,210), (2,1,70), (2,2,160), (3,1,180), (3,2,180), (3,3,180)
	2	1	(100), (1,1,180), (1,2,210), (1,3,210), (2,2,70), (3,1,150), (3,2,300), (3,3,180)
		2	(150), (1,2,180), (2,2,80)
		4	(150), (1,3,180), (2,1,100), (2,2,60), (3,1,270), (3,2,270), (3,3,270)
3	1	1	(100), (1,1,210), (1,2,210), (1,3,210), (2,2,240), (3,1,80), (3,2,180), (3,3,180)
		4	(100), (1,3,210), (2,1,240), (2,2,240), (3,1,60), (3,2,200), (3,3,120)
	2	1	(100), (1,1,180), (1,2,300), (1,3,210), (2,2,270), (3,1,140), (3,2,50), (3,3,140)
		3	(200), (1,1,270), (1,3,270), (2,1,300), (3,2,60), (3,3,180)
		4	(150), (1,3,180), (2,1,270), (2,2,240), (3,1,180), (3,2,80), (3,3,120)
	3	1	(100), (1,1,270), (1,2,180), (1,3,150), (2,2,180), (3,1,160), (3,2,160), (3,3,70)
		3	(100), (1,1,180), (1,3,180), (2,1,270), (3,2,180), (3,3,80)
		4	(50), (1,3,270), (2,1,180), (2,2,150), (3,1,180), (3,2,140), (3,3,70)

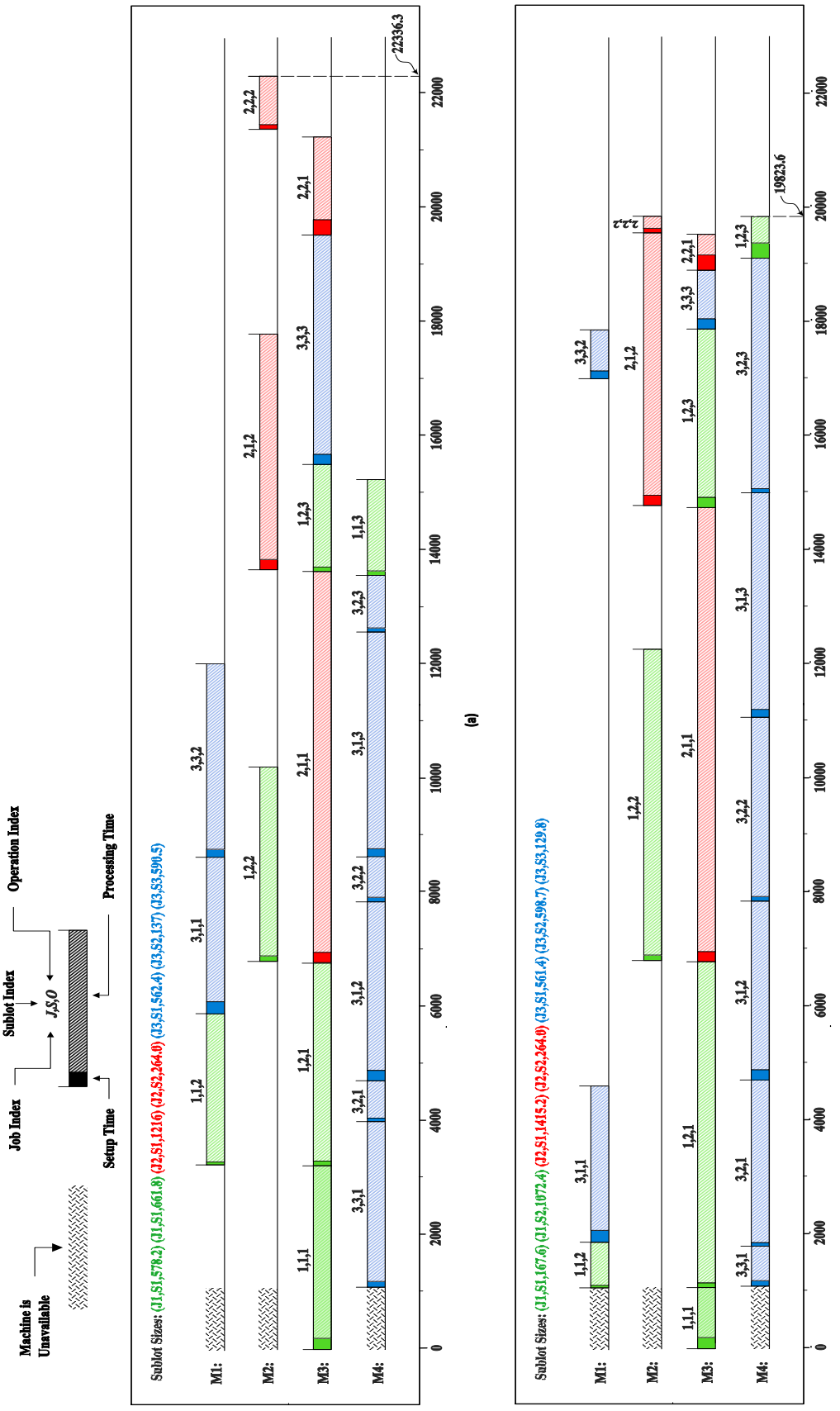

Figure 5.1: Schedule for problem-1: (a) solved by Pure GA (b) Solved by Proposed Hybrid GA Note: The detailed numerical values of the starting and the ending times of the setups and the operations are given in Table 5.3.

Table 5.3: The details of the schedules shown in Figure 5.1

		Solved by pure GA				Solved by proposed hybrid GA			
Machine	Run	(j, s, o)	SB	SE/PB	PE	(j, s, o)	SB	SE/PB	PE
M1	R1	(1,2,2)	3235.6	3285.6	5887.6	(1,2,2)	1080	1130	1884.3
	R2	(3,1,1)	5887.6	6097.6	8628.6	(3,1,1)	1884.3	2094.3	4620.7
	R3	(3,3,2)	8628.6	8768.6	12016.5	(3,3,2)	17028.5	17168.5	17882.6
M2	R1	(1,3,2)	6790	6890	10198.9	(1,3,2)	6790.0	6890.0	12251.9
	R2	(2,2,2)	13658.2	13838.2	17790.4	(2,2,2)	14753.6	14933.6	19533
	R3	(2,3,2)	21398.4	21478.4	22336.3	(2,3,2)	19533	19613	19823.6
M3	R1	(1,2,1)	0.0	200	3235.6	(1,2,1)	0.0	200	1080
	R2	(1,3,1)	3235.6	3315.6	6790.0	(1,3,1)	1080	1160	6790
	R3	(2,2,1)	6790	6970	13658.2	(2,2,1)	6790	6970	14753.6
	R4	(1,3,3)	13658.2	13838.2	15658.1	(1,3,3)	14753.6	14933.6	17882.6
	R5	(3,3,3)	15658.1	15838.1	19676.6	(3,3,3)	17882.6	18062.6	18906.6
	R6	(2,3,1)	19676.6	19946.6	21398.4	(2,3,1)	18906.6	19176.6	19533
M4	R1	(3,3,1)	1080.0	1180.0	3985.1	(3,3,1)	1080	1180	1796.7
	R2	(3,2,1)	3985.1	4045.1	4696	(3,2,1)	1796.7	1856.7	4700.7
	R3	(3,1,2)	8708.6	8888.6	11841.3	(3,1,2)	4700.7	4880.7	7828.2
	R4	(3,2,2)	11841.3	11921.3	12640.7	(3,2,2)	7828.2	7908.2	11051.6
	R5	(3,1,3)	12640.7	12780.7	16577.1	(3,1,3)	11051.6	11191.6	14981.2
	R6	(3,2,3)	16577.1	16647.1	17572.1	(3,2,3)	14981.2	15051.2	19092.6
	R7	(1,2,3)	17572.1	17842.1	19432.2	(1,2,3)	19092.6	19362.6	19823.6

Note: SB, SE, PB, PE stand for setup begins, setup ends, processing begins, and processing ends, respectively.

5.2. Computational Performance

In this section, a comparison between the convergence behavior and makespan of pure genetic algorithm (SGA), parallel genetic algorithm (PGA), and the proposed hybrid genetic algorithm (HGA) is presented. In order to illustrate the improvement attained using the proposed method, problems with much larger dimensions than Problem 1 (presented in the previous section) are considered. The general nature of the considered problems is depicted in Table 5.4. In Figure 5.3(a), provided by Defersha and Chen (2012), convergence behavior of SGA and PGAs in solving Problem 2 is illustrated. Additionally, in Figure 5.3(b), we present a comparison between the proposed method and SGA in terms of solving the same problem. In these figures, each curve represents the average convergence of the genetic algorithm from 10 test runs with different genetic parameters (Table 7). The genetic parameters given in Table 5.5 are generated randomly around those values with which the algorithm performs well. The initial sets of parameters are chosen following the general guidelines provided by Pezzella *et al.* (2008) and other published genetic algorithms.

As can be seen in Figure 5.3(a) , an improvement in the makespan value and also convergence rate is achieved as the number of processors increases in parallel computation. On the other hand, as is shown in Figure 5.3(b), with much less computational cost, the proposed hybrid GA is able to reach the solution in a manner similar to what is achieved by a 24-subpopulation parallel genetic algorithm (PGA-24). The parallel genetic algorithms used in this research are coded in C++ programming language using the MPI message-passing library for communication. The codes are executed in a parallel computation environment, composed of more than 250 interconnected workstations, each having an eight-core Intel Xeon 2.8GHz processor. When solving Problem 2, the sequential genetic algorithm generates a schedule with a makespan of 5227 minutes, on average. By increasing the number of the processors to 8, 16, 24, 32, and 48 - the average

makespan of the test runs is reduced by 109, 125, 165, 170, and 183 minutes, respectively. Our proposed method improves the SGA-generated makespan by 162 minutes using only a five-core Intel 1.7 GHz processor. Moreover, even better solutions are achieved using the proposed hybrid GA in solving other problems considered in this paper. As is shown in Figure 5.4, the average convergence graphs of the pure genetic algorithm (SGA), a 24-subpopulation parallel genetic algorithm, and proposed hybrid GA in solving Problems 2, 3, 4, and 5, are compared with each other. These figures demonstrate that our proposed method achieves a better makespan with a quicker convergence rate than does the 24-subpopulation parallel genetic algorithm.

Table 5.4: The general nature of the problems considered

Problem No.	Number of machines	Number of jobs	Number of sublots for each job	Number of operations for the jobs	Number of alternative routes for the operation
2	8	20	4	3 to 5	1 to 3
3	12	30	4	3 to 6	1 to 3
4	10	25	4	3 to 4	1 to 3
5	12	35	3	2 to 4	1 to 3

Table 5.5: Genetic parameters used for the test runs

		Test Run									
Parameter		1	2	3	4	5	6	7	8	9	10
Population Size		5500	2500	4500	3000	4000	3500	2500	2000	2800	3800
Tournament size factor		0.10	0.20	0.05	0.03	0.20	0.15	0.12	0.25	0.2	0.15
Crossover probability for:											
SPC1	ρ_1	0.85	0.8	0.95	0.80	0.75	0.80	0.75	0.90	0.7	0.8
SPC2	ρ_2	0.85	0.8	0.95	0.80	0.90	0.80	0.75	0.80	0.85	0.8
OMAC	ρ_3	0.95	0.85	0.80	0.75	0.75	0.90	0.85	0.75	0.85	0.9
JLOSC	ρ_4	0.80	0.90	0.85	0.95	0.80	0.75	0.80	0.70	0.8	0.85
SLOSC	ρ_5	0.90	0.8 0	0.85	0.90	0.95	0.90	0.75	0.70	0.9	0.83
Mutation probability for:											
SStM	σ_1	0.15	0.10	0.05	0.10	0.02	0.10	0.15	0.10	0.02	0.1
SSwM	σ_2	0.10	0.08	0.10	0.12	0.10	0.10	0.20	0.15	0.1	0.05
SSD	d	0.10	0.10	0.10	0.10	0.1	0.10	0.10	0.10	0.1	0.1
ROAM	σ_3	0.10	0.05	0.10	0.10	0.03	0.10	0.10	0.15	0.1	0.1
IOAM	σ_4	0.10	0.12	0.20	0.15	0.05	0.20	0.10	0.25	0.2	0.1
OSSM	σ_5	0.20	0.10	0.25	0.10	0.05	0.10	0.05	0.15	0.06	0.15

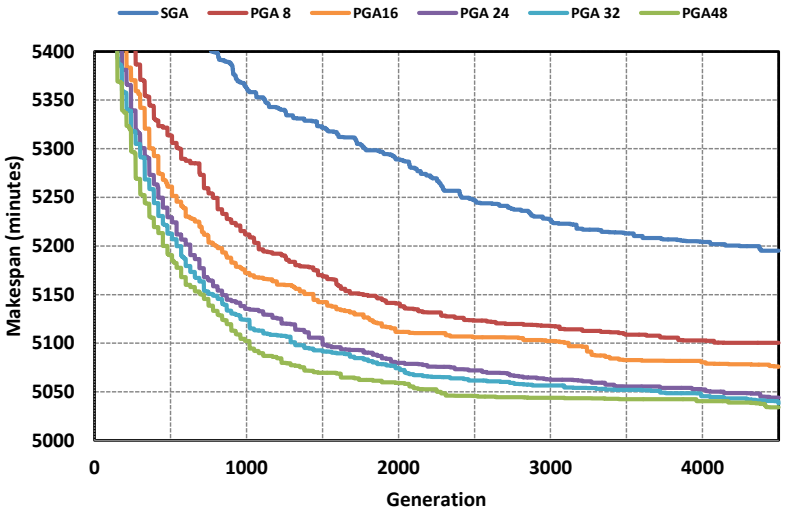

Figure 5.2: Performance improvement through parallelization of the genetic algorithm as the number of processor is increased from 1 to 8, 16, 24, 32, and to 48.

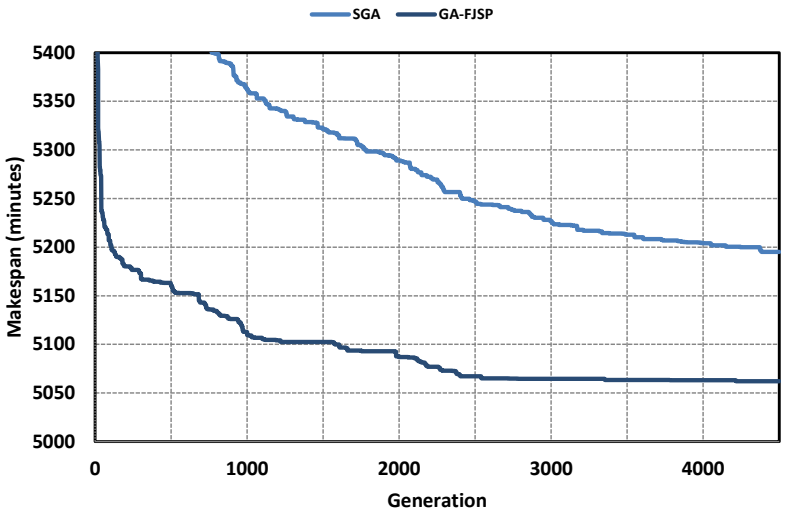

Figure 5.3: Performance improvement through using the proposed hybrid GA

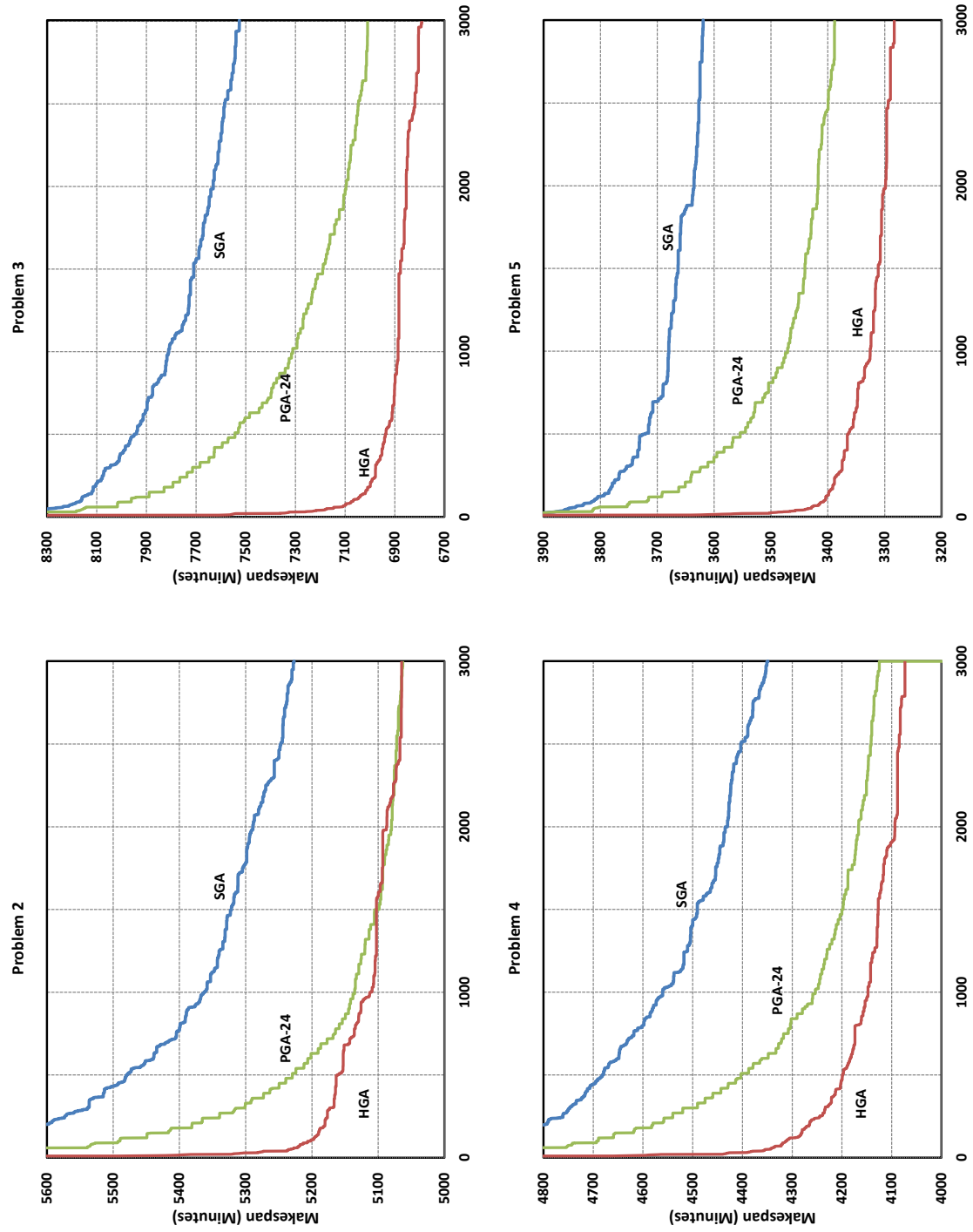

Figure 5.4: Average convergence of the SGA, PGA and HGA for problems 2, 3, 4, and 5.

5.3. Empirical Study

In this section, the impact of the genetic parameters on SGA and the proposed HGA performance is compared. Figure 5.5 shows the plots of the makespan of the schedules obtained after 3000 iterations by SGA and the proposed HGA under different test runs. The test runs are differentiated by the settings of their genetic parameters, as is shown in Table 5.5. In Figure 5.5, each pattern represents the way the final solution quality of the SGA and HGA is affected by the genetic parameters. It can be seen that the HGA final solutions have more stability than does the pure GA against changes in the genetic parameters. In the hybrid approach, 8 from 10 final solutions lie within plus and minus 0.01 of the HGA final solutions mean. However, in the SGA case, only 5 out of 10 answers lie within that range. In other words, it can be concluded that in terms of variation in genetic parameters, our proposed method is more robust than is the pure GA.

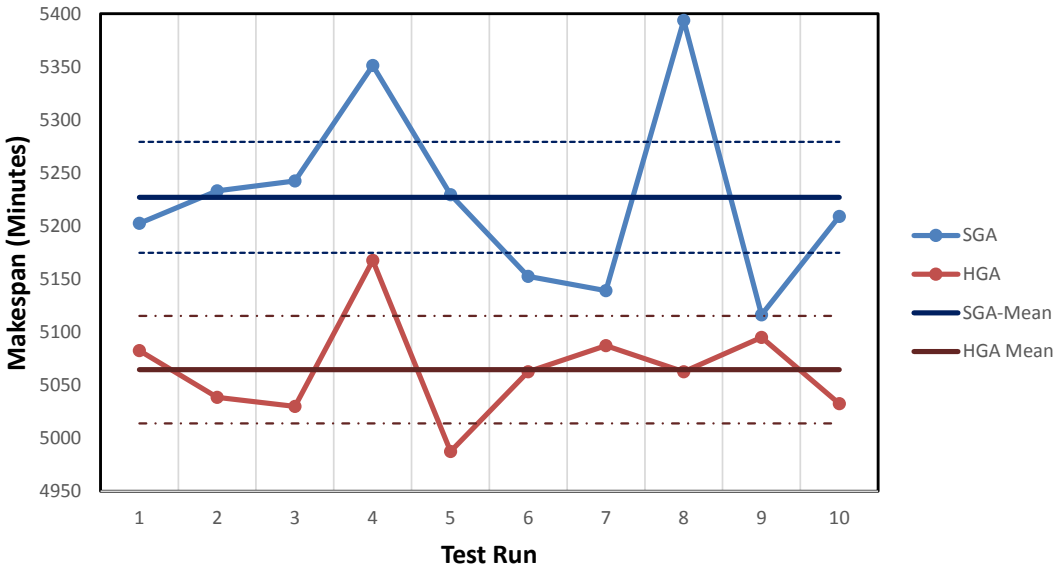

Figure 5.5: The effect of changing genetic parameters on the final solution quality obtained by the SGA and HGA in solving problem 2

Chapter 6

Research Outline

6.1. Summary and Conclusion

This study aimed at minimizing the makespan of lot streaming flexible job shop problems. The comprehensive problem studied in this thesis takes into account routing flexibility, sequence-dependent setups, machine release dates and lag time. In addition, the sublots of products can have attached or detached setup times, and also are allowed to be intermingled. The considered problem in this thesis denotes by : FJc, L_n | S_{omjk}, $routf$, r_m, lag, $btchls$ | Tv, $FixN$, Dc, xPn,Si | C_{max}.

In this thesis, we developed a linear programming assisted genetic algorithm. Linear program is employed as a GA assistant to further improve promising solutions in the initial population and during the genetic search process by determining the optimal values of the continuous variables corresponding to the values of the integer variables of these promising solutions. The method we used to reduce looping in the hybrid genetic algorithm and the way LP is incorporated with the GA in the whole solution procedure are two distinctive features of our novel approach.

A series of numerical examples were used to demonstrate the superiority of

the proposed GA over parallel and pure GA methods. The experiments showed that the hybridization of the genetic algorithm with the linear programming greatly improved it convergence behavior. Based on the results, our proposed algorithm outperformed or performed equally with the resource-intensive parallel pure genetic algorithm, while it only utilized a single computational resource. Also, the results showed that our proposed method achieved significantly better outcomes than the pure genetic algorithm in terms of computational performance.

6.2. Future Research and Recommendations

During the last decades, many efforts have been devoted to applying automated scheduling to real-world situations. Although a few notable successes have been achieved, still a significant amount of scheduling tasks which could derive benefit from automated scheduling are conducted manually. One of the reasons why many organizations still avoid utilizing automated scheduling is that the majority of problems which have been studied by researchers over the year are actually the simplified version of the problems faced by industries. Usually, the constraints and performance criteria in real-world problems tend to be more complex. For instance, the well-known job shop and flow shop problems which are extensively studied by researchers over the years are much easier and less complex than actual problems in manufacturing industries, and in most cases, they are not applicable to real-world situations. Therefore, in future works, researchers should emphasize more on solving the real-life problems rather than focusing only on theory.

Most of the problems in scheduling research are in class of NP-hard problems. As the size of this kind of problem increases the amount of time required to solve them increases dramatically. As a result, in order to reduce the computational time substantially, the guarantee for obtaining an optimal solution should

be sacrificed. This means that there always exists a tradeoff between the solution quality and the computational time. While many of the parameters of this tradeoff are user-dependent, the researchers usually only focus on speeding up their algorithms and reducing computational time. Hence, in future researches, algorithms should be designed in such a way that the users are capable to make tradeoffs between solution quality and computational cost.

Moreover, in future works, researchers should work on developing schedulers which can be configured easily for a wide variety scheduling problems. In this way, high costs of implementing an automated scheduling for an organization can be dramatically reduced, and consequently, automated scheduling becomes more desirable than manual scheduling for industries. Also, most of the organizations need dynamic rescheduling, where the scheduler software monitors the process and if necessary adjusts the schedule. This issue should also be addressed in future works Montana (2002).

Bibliography

Aarts, E. E. H. and Lenstra, J. K., 1997. Local search in combinatorial optimization. Princeton University Press,

Adams, J., Balas, E., and Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. *Management science*, **34 (3)**, 391–401.

Bäck, T., 1996. Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press on Demand,

Bagheri, A., Zandieh, M., Mahdavi, I., and Yazdani, M., 2010. An artificial immune algorithm for the flexible job-shop scheduling problem. *Future Generation Computer Systems*, **26 (4)**, 533–541.

Baker, K. R. and Trietsch, D., 2009. Principles of sequencing and scheduling. Wiley. com,

Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J., 2009. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: an international journal*, **8 (2)**, 239–287.

Blackburn, J., 1991. Time-Based Competition. Business One Irwin, Burr Ridge, IL,

Błażewicz, J., Ecker, K. H., and Pesch, E., 2007. Handbook on scheduling [electronic resource]: from theory to applications. Springer,

Blum, C., Puchinger, J., Raidl, G. R., and Roli, A., 2011. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, **11 (6)**, 4135–4151.

Blum, C., Roli, A., and Sampels, M., 2008. Hybrid metaheuristics: An emerging approach to optimization. Vol. 114. Springer,

Bockerstette, J. and Shell, R., 1993. Time Based Manufacturing. McGraw-Hill, New York,

Bowman, E. H., 1959. The schedule-sequencing problem. *Operations Research*, **7 (5)**, 621–624.

Brucker, P., 2007. Scheduling algorithms. Springer,

Brucker, P. and Schlie, R., 1990. Job-shop scheduling with multi-purpose machines. *Computing*, **45 (4)**, 369–375.

Buscher, U. and Shen, L., 2009. An integrated tabu search algorithm for the lot streaming problem in job shops. *European Journal of Operational Research*, **199 (2)**, 385–399.

Buscher, U. and Shen, L., 2011. An integer programming formulation for the lot streaming problem in a job shop environment with setups. In: Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. 2.

Chan, F., Wong, T., and Chan, P., 2008. Lot streaming for product assembly in job shop environment. *Robotics and Computer-Integrated Manufacturing*, **24**, 321–331.

Chan, F. T., Wong, T., and Chan, L., 2009. The application of genetic algorithms to lot streaming in a job-shop scheduling problem. *International Journal of Production Research*, **47 (12)**, 3387–3412.

Chan, F. T., Wong, T., and Chan, P., 2004. Equal size lot streaming to job-shop scheduling problem using genetic algorithms. In: Intelligent Control, 2004. Proceedings of the 2004 IEEE International Symposium on. IEEE, pp. 472–476.

Chang, J. H. and Chiu, H. N., 2005. A comprehensive review of lot streaming. *International Journal of Production Research,*, **43**, 1515–1536.

Chang*, J. H. and Chiu, H. N., 2005. A comprehensive review of lot streaming. *International Journal of Production Research*, **43 (8)**, 1515–1536.

Chaudhry, I. A., Mahmood, S., and Ahmad, R., 2010. Minimizing makespan for machine scheduling and worker assignment problem in identical parallel machine models using ga. In: Proceedings of the World Congress on Engineering. Vol. 3.

Chekuri, C., Khanna, S., and Zhu, A., 2001. Algorithms for minimizing weighted flow time. In: Proceedings of the thirty-third annual ACM symposium on Theory of computing. ACM, pp. 84–93.

Chen, H., Ihlow, J., and Lehmann, C., 1999. A genetic algorithm for flexible job-shop scheduling. In the proceedings of the 1999 IEEE International Conference on Robotics & Automation. May 1999, Detroit, Michigan, pp. 1120–1125.

Cheng, M., Mukherjee, N., and Sarin, S., 2013. A review of lot streaming. *International Journal of Production Research*, **(ahead-of-print)**, 1–24.

Cotta, C., Talbi, E.-G., and Alba, E., 2005. 15 parallel hybrid metaheuristics. *Parallel Metaheuristics: A New Class of Algorithms*, **47**, 347.

Dauzere-Peres, S. and Lasserre, J., 1997. Lot streaming in job-shop scheduling. *Operations Research*, **45**, 584–595.

Dauzère-Pérès, S. and Lasserre, J.-B., 1997. Lot streaming in job-shop scheduling. *Operations Research*, **45 (4)**, 584–595.

Defersha, F. M. and Chen, M., 2009. A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. In: Computational Science and Engineering, 2009. CSE'09. International Conference on. Vol. 1. IEEE, pp. 201–208.

Defersha, F. M. and Chen, M., 2010a. A hybrid genetic algorithm for flowshop lot streaming with setups and variable sublots. *International Journal of Production Research*, **48 (6)**, 1705–1726.

Defersha, F. M. and Chen, M., 2010b. A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *The International Journal of Advanced Manufacturing Technology*, **49 (1-4)**, 263–279.

Defersha, F. M. and Chen, M., 2012. Jobshop lot streaming with routing flexibility, sequence-dependent setups, machine release dates and lag time. *International Journal of Production Research*, **50 (8)**, 2331–2352.

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

Drezner, Z. and MisevičIus, A., 2012. Enhancing the performance of hybrid genetic algorithms by differential improvement. *Computers & Operations Research*, .

Eiselt, H. A. and Sandblom, C.-L., 2004. Decision analysis, location models, and scheduling problems. Springer,

El-Mihoub, T. A., Hopgood, A. A., Nolle, L., and Battersby, A., 2006. Hybrid genetic algorithms: A review. *Engineering Letters*, **13 (2)**, 124–137.

Emmons, H. and Vairaktarakis, G., 2012. Flow shop schedulin. Vol. 182. Springer,

Fattahi, P., Mehrabad, M. S., and Jolai, F., 2007. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, **18 (3)**, 331–342.

Feldmann, M. and Biskup, D., 2008. Lot streaming in a multiple product permutation flow shop with intermingling. *International Journal of Production Research*, **46**, 197–216.

Gao, J., Gen, M., and Sun, L., 2006. Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm. *Journal of Intelligent Manufacturing*, **17 (4)**, 493–507.

Gao, J., Sun, L., and Gen, M., 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, **35**, 2892–2907.

Garey, M. R., Johnson, D. S., and Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, **1**, 117–129.

Gendreau, M. and Potvin, J.-Y., 2005. Metaheuristics in combinatorial optimization. *Annals of Operations Research*, **140 (1)**, 189–213.

Ghasemi, M., 2008. Lot streaming in hybrid flow shop scheduling. Ph.D. thesis, Concordia University.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, **13 (5)**, 533–549.

Goldberg, D., 1989. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, New York,

Goldberg, D. E., Korb, B., and Deb, K., 1989. Messy genetic algorithms: Motivation, analysis, and first results. *Complex systems*, **3 (5)**, 493–530.

Gonçalves, J. F., de Magalhães Mendes, J. J., and Resende, M. G., 2005. A hybrid genetic algorithm for the job shop scheduling problem. *European journal of operational research*, **167 (1)**, 77–95.

Gonzalez, T. and Sahni, S., 1976. Open shop scheduling to minimize finish time. *Journal of the ACM (JACM)*, **23 (4)**, 665–679.

Grabowski, J. and Pempera, J., 2007. The permutation flow shop problem with blocking. a tabu search approach. *Omega*, **35 (3)**, 302–311.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A., 1977. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics. v5*, , 287–326.

Grosan, C. and Abraham, A., 2007. Hybrid evolutionary algorithms: methodologies, architectures, and reviews. In: Hybrid evolutionary algorithms. Springer, pp. 1–17.

Gupta, J. N. and Stafford Jr, E. F., 2006. Flowshop scheduling research after five decades. *European Journal of Operational Research*, **169 (3)**, 699–711.

Hall, N. G. and Sriskandarajah, C., 1996. A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, **44 (3)**, 510–525.

Hart, W. E., Krasnogor, N., and Smith, J. E., 2005. Memetic evolutionary algorithms. In: Recent advances in memetic algorithms. Springer, pp. 3–27.

Heller, J., 1959. Combinatorial, probabilistic and statistical aspects fo an m x j scheduling problem. Tech. rep., New York Univ., New York. Atomic Energy Commission Computing and Applied Mathematics Center.

Holland, J., 1975. Adaptation in natural and artificial systems. *University of Michigan Press*, .

ILOG Inc., 2008. CPLEX 12.0 Users Manual. 1080 Linda Vista Ave. Mountain View, CA 94043, (http://www.ilog.com).

Imanipour, N., 2006. Modeling & solving flexible job shop problem with sequence dependent setup times. In: Service Systems and Service Management, 2006 International Conference on. Vol. 2. IEEE, pp. 1205–1210.

Jackson, J. R., 1955. Scheduling a production line to minimize maximum tardiness. Tech. rep., DTIC Document.

Johnson, S. M., 1954. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, **1 (1)**, 61–68.

Jourdan, L., Basseur, M., and Talbi, E.-G., 2009. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, **199 (3)**, 620–629.

Kacem, I., 2003. Genetic algorithm for the flexible jobshop scheduling problem. In the Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Washington, DC, pp. 3464–6469.

Kalir, A. and Sarin, S., 2000. Evaluation of the potential benefits of lot streaming in flow-shop systems. *International Journal of Production Economics*, **66**, 131–142.

Kan, A., 1976. Machine scheduling problems: classification, complexity and computations. Nijhoff,
URL http://books.google.ca/books?id=-sMSAQAAMAAJ

Khuri, S. and Miryala, S. R., 1999. Genetic algorithms for solving open shop scheduling problems. In: Progress in Artificial Intelligence. Springer, pp. 357–368.

Kim, K. and Jeong, I.-J., 2009. Flow shop scheduling with no-wait flexible lot streaming using an adaptive genetic algorithm. *The International Journal of Advanced Manufacturing Technology*, **44 (11-12)**, 1181–1190.

Kim, K.-H. and Egbelu, P., 1999. Scheduling in a production environment with multiple process plans per job. *International Journal of Production Research*, **37 (12)**, 2725–2753.

Kim, S. and Bobrowski, P., 1994. Impact of sequence-dependent setup time on job shop scheduling performance. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, **32 (7)**, 1503–1520.

Kochenberger, G. A. *et al.*, 2003. Handbook of metaheuristics. Springer,

Krasnogor, N. and Smith, J., 2005. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *Evolutionary Computation, IEEE Transactions on*, **9 (5)**, 474–488.

Kumar, S., Bagchi, T., and Sriskandarajah, C., 2000. Lot streaming and scheduling heuristics for m-machine no-wait flow shop. *Computers and Industrial Engineering*, **38**, 149–172.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H., and Shmoys, D. B., 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, **4**, 445–522.

Lee, C.-Y., 1996. Machine scheduling with an availability constraint. *Journal of global optimization*, **9 (3-4)**, 395–416.

Lee, C.-Y., 1999. Minimizing makespan on a single batch processing machine with dynamic job arrivals. *International Journal of Production Research*, **37 (1)**, 219–236.

Linn, R. and Zhang, W., 1999. Hybrid flow shop scheduling: a survey. *Computers & Industrial Engineering*, **37 (1)**, 57–61.

Liu, C.-H., Chen, L.-S., and Lin, P.-S., 2013. Lot streaming multiple jobs with values exponentially deteriorating over time in a job-shop environment. *International Journal of Production Research*, **51 (1)**, 202–214.

Low, C., Hsu, C., and Huang, K., 2004a. Benefits of lot splitting in job-shop scheduling. *International Journal of Advanced Manufacturing Technology*, **24**, 773–780.

Low, C., Hsu, C.-M., and Huang, K.-I., 2004b. Benefits of lot splitting in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, **24 (9-10)**, 773–780.

Low, C. and Wu, T.-H., 2001. Mathematical modelling and heuristic approaches to operation scheduling problems in an fms environment. *International Journal of Production Research*, **39 (4)**, 689–708.

Low, C. and Yeh, Y., 2009. Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. *Robotics and Computer-Integrated Manufacturing*, **25 (2)**, 314–322.

Lozano, M. and García-Martínez, C., 2010. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Computers & Operations Research*, **37 (3)**, 481–497.

Manne, A. S., 1960. On the job-shop scheduling problem. *Operations Research*, **8 (2)**, 219–223.

Marimuthu, S., Ponnambalam, S. G., and Jawahar, A. N., 2008. Evolutionary algorithms for scheduling m-machine flow shop with lot streaming. *Robotics and Computer-Integrated Manufacturing*, **24**, 125–139.

Marimuthu, S., Sait, A. N., *et al.*, 2013. Performance evaluation of proposed differential evolution and particle swarm optimization algorithms for scheduling m-machine flow shops with lot streaming. *Journal of Intelligent Manufacturing*, **24 (1)**, 175–191.

Martin, C. H., 2009. A hybrid genetic algorithm/mathematical programming approach to the multi-family flowshop scheduling problem with lot streaming. *OMEGA International Journal of Management Sciences*, **37**, 126–137.

Melouk, S., Damodaran, P., and Chang, P.-Y., 2004. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing. *International Journal of Production Economics*, **87 (2)**, 141–147.

Montana, D., 2002. How to make scheduling research relevant. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO. Vol. 2.

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, **826**, 1989.

Moscato, P., Cotta, C., and Mendes, A., 2004. Memetic algorithms. In: New optimization techniques in engineering. Springer, pp. 53–85.

Naderi, B., Zandieh, M., and Ghomi, S. F., 2009. Scheduling sequence-dependent setup time job shops with preventive maintenance. *The International Journal of Advanced Manufacturing Technology*, **43 (1-2)**, 170–181.

Osman, I. H. and Laporte, G., 1996. Metaheuristics: A bibliography. *Annals of Operations Research*, **63 (5)**, 511–623.

Pan, Q.-K., Duan, J.-h., Liang, J., Gao, K., and Li, J., 2010. A novel discrete harmony search algorithm for scheduling lot-streaming flow shops. In: Control and Decision Conference (CCDC), 2010 Chinese. IEEE, pp. 1531–1536.

Pan, Q.-K., Fatih Tasgetiren, M., Suganthan, P. N., and Chua, T. J., 2011a. A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences*, **181 (12)**, 2455–2468.

Pan, Q.-K., Wang, L., Gao, L., and Li, J., 2011b. An effective shuffled frog-leaping algorithm for lot-streaming flow shop scheduling problem. *The International Journal of Advanced Manufacturing Technology*, **52 (5-8)**, 699–713.

Partheepan, R., 2004. Hybrid genetic algorithms. , .

Pezzella, F., Morganti, G., and Ciaschetti, G., 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers and Operations Research*, **35**, 3202–3212.

Pinedo, M., 2012. Scheduling: theory, algorithms, and systems. Springer,

Potts, C. and Baker, K., 1989. Flow shop scheduling with lot streaming. *Operations Research Letter*, **8**, 297–303.

Potts, C. N. and Van Wassenhove, L. N., 1992. Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity. *Journal of the Operational Research Society*, , 395–406.

Puchinger, J. and Raidl, G. R., 2005. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: Artificial intelligence and knowledge engineering applications: a bioinspired approach. Springer, pp. 41–53.

Qian, B., Wang, L., Huang, D.-x., Wang, W.-l., and Wang, X., 2009. An effective hybrid de-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers & Operations Research*, **36 (1)**, 209–233.

Raidl, G. R., 2006. A unified view on hybrid metaheuristics. In: Hybrid Metaheuristics. Springer, pp. 1–12.

Rayward-Smith, V. J., Osman, I. H., Reeves, C. R., and Smith, G. D., 1996. Modern heuristic search methods. Wiley Chichester,

Reiter, S., 1966. A system for managing job shop production. *Journal of Business*, **34**, 371–393.

Reza Hejazi, S. and Saghafian, S., 2005. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, **43 (14)**, 2895–2929.

Ribas, I., Leisten, R., and Framiñan, J. M., 2010. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, **37 (8)**, 1439–1454.

Ribeiro, C. and Hansen, P., 2002. Essays and surveys in metaheuristics. Kluwer Academic Publishers,

Ronconi, D. P., 2004. A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, **87 (1)**, 39–48.

Roshanaei, V., Azab, A., and ElMaraghy, H., 2013. Mathematical modelling and a meta-heuristic for flexible job shop scheduling. *International Journal of Production Research*, **51 (20)**, 6247–6274.

Rossi, A. and Dini, G., 2007. 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.

Ruiz, R., Şerifoğlub, F. S., and Urlings, T., 2008. Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, **35**, 1151–1175.

Saidi, M. and Fattahi, P., 2007. Flexible job shop scheduling with tabu search algorithm. *International Journal of Advanced Manufacturing Technology*, **35**, 563–570.

Sarin, S. C. and Jaiprakash, P., 2007. Flow shop lot streaming. Springer,

Sivanandam, S. and Deepa, S., 2007. Introduction to genetic algorithms. Springer,

Smith, W. E., 1956. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, **3 (1-2)**, 59–66.

Stecke, K. E. and Raman, N., 1995. Fms planning decisions, operating flexibilities, and system performance. *Engineering Management, IEEE Transactions on*, **42 (1)**, 82–90.

Talbi, E.-G., 2002. A taxonomy of hybrid metaheuristics. *Journal of heuristics*, **8 (5)**, 541–564.

Tseng, C. T. and Liao, C. J., 2008. A discrete particle swarm optimization for lot-streaming flowshop scheduling problem. *European Journal of Operational Research*, **191**, 360–373.

van den Akker, J. M., Hoogeveen, J. A., and van de Velde, S. L., 1999. Parallel machine scheduling by column generation. *Operations Research*, **47 (6)**, 862–872.

Ventura, J. A. and Yoon, S.-H., 2012. A new genetic algorithm for lot-streaming flow shop scheduling with limited capacity buffers. *Journal of Intelligent Manufacturing*, , 1–12.

Wagner, H. M., 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, **6 (2)**, 131–140.

Wang, Y. M., Yin, H. L., and Da Qin, K., 2013. A novel genetic algorithm for flexible job shop scheduling problems with machine disruptions. *The International Journal of Advanced Manufacturing Technology*, , 1–10.

Wong, T. and Ngan, S.-C., 2013. A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. *Applied Soft Computing*, **13 (3)**, 1391–1399.

Xia, W. and Wu, Z., 2005. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, **48 (2)**, 409–425.

Yen, J., Randolph, D., Liao, J. C., and Lee, B., 1995. A hybrid approach to modeling metabolic systems using genetic algorithm and simplex method. In: Artificial Intelligence for Applications, 1995. Proceedings., 11th Conference on. IEEE, pp. 277–283.

Yi, H., Duan, Q., and Liao, T., 2012. Three improved hybrid metaheuristic algorithms for engineering design optimization. *Applied Soft Computing*, .

Yoon, S.-H. and Ventura, J. A., 2002. An application of genetic algorithms to lot-streaming flow shop scheduling. *IIE Transactions*, **34 (9)**, 779–787.

Zhang, G., Gao, L., and Shi, Y., 2011. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, **38 (4)**, 3563–3573.

Zipkin, P. H., 1986. Models for design and control of stochastic, multi-item batch production systems. *Operations Research*, **34 (1)**, 91–104.