Permutation based Genetic Algorithm with Event-Scheduling/Time-Advance Algorithm as Decoder for a Flexible Job-shop Scheduling Problem

by

C. H. Hayson Ko

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

© C. H. Hayson Ko, December, 2017
Today, numerous research support the growing scheduling problems that exist globally in competitive businesses. Scheduling needs to become efficient in order to remain relevant against competitors. Simulations need to provide results in short periods of time so that adjustments can be made and unnecessary costs can be avoided. Scheduling problems have become larger in size and greater in complexity given the rising product variations and increase in variety for manufacturing equipment. Hence, there is a practical need for genetic algorithms solving scheduling problems to be fast and versatile. This thesis introduces an event-scheduling/time-advance algorithm for the decoder to reduce the load on the genetic algorithm with a smaller global search space. Consequently, convergence can be reached sooner and larger problems can be tackled easily. The structure of this heuristic algorithm allows metrics to be easily implemented in order to give the user performance measures on the scheduling problem.

Keywords: scheduling, genetic algorithm, flowshop, jobshop, flexible, event scheduling, time advance, decoder, performance, scalable
Dedicated to my mother and my father.
ACKNOWLEDGEMENTS

I would like to thank Professor Fantahun M. Defersha for spending countless weeks introducing the concept of modelling manufacturing operations management problems. While challenges continuously appeared every semester, his wisdom and guidance steered me clear from major failures and significantly reduced my learning curve. His everlasting patience whilst teaching and introducing me new concepts has helped me progress thoroughly throughout my masters thesis research.

Thank-you NSERC, Integrated Production and Manpower Scheduling and MERSEN Canada Toronto Inc. for providing funding. This research would not have been possible without the financial support.

I would like to thank my parents for always providing the support and encouragement whenever doubts arose.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF SYMBOLS</td>
<td>xii</td>
</tr>
<tr>
<td>LIST OF ACRONYMS</td>
<td>xiv</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Existing Facility Layouts</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Multi-Row Layout</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 Flowshop Layout</td>
<td>4</td>
</tr>
<tr>
<td>1.1.3 Cellular Layout</td>
<td>5</td>
</tr>
<tr>
<td>1.1.4 Jobshop Layout</td>
<td>6</td>
</tr>
<tr>
<td>1.1.5 Dynamic Layout</td>
<td>8</td>
</tr>
<tr>
<td>1.2 Simulation and Mathematical Modelling</td>
<td>9</td>
</tr>
<tr>
<td>1.3 Organization of Thesis</td>
<td>11</td>
</tr>
<tr>
<td>2 Literature Review</td>
<td>12</td>
</tr>
<tr>
<td>2.1 Introduction to Scheduling</td>
<td>12</td>
</tr>
<tr>
<td>2.2 Genetic Algorithm Methodology</td>
<td>13</td>
</tr>
<tr>
<td>2.3 Genetic Algorithms Applied to Facility Layout Problems</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Flowshop Problem</td>
<td>19</td>
</tr>
<tr>
<td>2.3.2 Jobshop Problem</td>
<td>20</td>
</tr>
<tr>
<td>2.3.3 Direction of Improvement</td>
<td>22</td>
</tr>
<tr>
<td>2.4 Current Math Modelling Practices</td>
<td>24</td>
</tr>
<tr>
<td>2.5 Addressing Dynamic Facility Layout Scheduling</td>
<td>24</td>
</tr>
<tr>
<td>2.6 Purpose of Thesis</td>
<td>26</td>
</tr>
</tbody>
</table>
# Mathematical Model

3.1 Rudimentary Model

3.1.1 Problem Description and Assumptions

3.1.2 MILP Model

3.2 FJSP-LS Model

3.2.1 Problem Description

3.2.2 MILP Model

# The Proposed Algorithm

4.1 Introduction

4.2 Traditional Encoded Sequence Genetic Algorithm

4.2.1 Solution Representation

4.2.2 Initial Population

4.2.3 Fitness Evaluation

4.2.4 Selection Operator

4.2.5 Crossover Operators

4.2.6 Mutation Operator

4.3 Event-Scheduling/Time-Advance Genetic Algorithm

4.3.1 Solution Representation

4.3.2 Initial Population

4.3.3 Fitness Evaluation

4.3.4 Selection Operator

4.3.5 Crossover Operator

4.3.6 Mutation Operator

# Numerical Examples

5.1 Initial Population Generation Performance

5.2 Problem Size and Genetic Algorithm Computational Performance
6 Research Summary

6.1 Discussion and Conclusion ........................................ 74
6.2 Future Research ..................................................... 75

Bibliography ................................................................ 77
### LIST OF FIGURES

1.1 Multi-row facility layout adopted from (Hungerlnder and Anjos, 2015). 3
1.2 Multi-row facility layout adopted from (Hungerlnder and Anjos, 2015). Gantry robots can be used to handle materials in this layout 4
1.3 In a flowshop, parts flow strictly in the order of the machines placed down the line. 5
1.4 Appropriate layout to be used depending on product numbers (Sayed and Lash, 2014). 6
1.5 In the jobshop layout, parts can come in and use any eligible machine to process its operations before completion. 7
1.6 Dynamic layout for a 6-department (departments indicated by the numbers), 3 period problem arranged in a 2 by 3 facility layout adopted from (Ripon et al., 2010). 9

3.1 Notice how each run is assigned to only one job. Here the completion time of all the runs is 1300 seconds. 28
3.2 Example of how variables $r$, $j$, $p$, $S_{j,p}$, and $T_{j}$ are used. 30

4.1 The RHS of a TRADGA chromosome considering lot streaming is adopted from Defersha and Chen (2012a). 43
4.2 The full TRADGA chromosome considering lot streaming is adopted from Defersha and Chen (2012a). 44
4.3 SPC-1 and SPC-2 adopted from Defersha and Chen (2012a). 51
4.4 OMAC adopted from Defersha and Chen (2012a). 53
4.5 JLOSC and SLOSC adopted from Defersha and Chen (2012a). 54
4.6 An overview of the TAESGA logic. 57
4.7 Solution representation for TAESGA. Figure is adopted from Defersha and Chen (2012b). 58
4.8 Illustration of the LHS-Segment crossover operators assuming a crossover point between $\alpha_{1,3}$ and $\alpha_{2,3}$. Only one crossover operators will be applied on a given pair of parents. Figure is adopted from Defersha and Chen (2012b).

4.9 Illustration of SPOX-1: (a) Sublots to the left of the crossover point are inherited from the direct parent. (b) Then, the missing sublots are copied in their relative order of the other parent. Figure is adopted from Defersha and Chen (2012b).

4.10 Illustration of SPSSOX: (a) First, the common sublots in both parents are copied over to the children; (b) Then, sublots to the left of the crossover point are inherited from the direct parent. (c) Finally, the missing sublots are copied in their relative order of the other parent. Figure is adopted from Defersha and Chen (2012b).

5.1 The distribution of makespan for 1500 randomly generated solution in TAESGA vs TRADGA.

5.2 The RHS-Segment of a randomly permutated chromosome that may exist in the TRADGA. The first row denotes the location of the gene. The second row denotes the job, operation, and machine encoded into the gene.

5.3 The RHS-Segment of a randomly permutated chromosome that may exist in the TAESGA. The first row denotes the location of the gene. The second row denotes the job encoded into the gene.

5.4 Finding the order in which the operations are completed based on the TRADGA chromosome in Figure 5.2.

5.5 Decoding the TAESGA chromosome based on the TAESGA chromosome in Figure 5.3.

5.6 Computational performance of small problems.

5.7 Computational performance of large problems.

5.8 Computational performance of medium problems.
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Completion times for the jobs in Figure 3.1</td>
<td>30</td>
</tr>
<tr>
<td>3.2</td>
<td>Completion times for the runs in Figure 3.1</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Average setup time, processing time and machine release date. Data is</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>adopted from Defersha and Chen (2010b)</td>
<td></td>
</tr>
<tr>
<td>4.2</td>
<td>Initial assignments considering processing time (machine workload updates</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>dates in bold). Table is adopted from Defersha and Chen (2010b)</td>
<td></td>
</tr>
<tr>
<td>4.3</td>
<td>Initial assignments considering processing time, average setup time and</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>machine release date (machine workload updates in bold). Table is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adopted from Defersha and Chen (2010b)</td>
<td></td>
</tr>
<tr>
<td>4.4</td>
<td>Remaining work calculation and operations sequence for an initial solution</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>based on the operation-machine assignment given in Table 4.3. Table is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>adopted from Defersha and Chen (2010b)</td>
<td></td>
</tr>
</tbody>
</table>
# LIST OF SYMBOLS

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta$</td>
<td>Change in how poor the adjacent solution is compared to current solution for SA</td>
</tr>
<tr>
<td>$t$</td>
<td>Temperature parameter for SA</td>
</tr>
<tr>
<td>$J$</td>
<td>Maximum number of jobs</td>
</tr>
<tr>
<td>$R$</td>
<td>Maximum number of runs</td>
</tr>
<tr>
<td>$j$</td>
<td>Job index</td>
</tr>
<tr>
<td>$p$</td>
<td>Job index. Note that the $j^{th}$ job follows $p$</td>
</tr>
<tr>
<td>$r$</td>
<td>Run index</td>
</tr>
<tr>
<td>$S_{j,p}$</td>
<td>Setup time for job $j$ that follows $p$</td>
</tr>
<tr>
<td>$S_{j,0}$</td>
<td>Initial setup time for job $j$</td>
</tr>
<tr>
<td>$T_j$</td>
<td>Processing time for job $j$</td>
</tr>
<tr>
<td>$c_j$</td>
<td>Completion time for job $j$</td>
</tr>
<tr>
<td>$\hat{c}_r$</td>
<td>Completion time for the $r^{th}$ run</td>
</tr>
<tr>
<td>$c_{\text{max}}$</td>
<td>Makespan of the schedule. It is also equivalent to the completion time of the last run processed in the system.</td>
</tr>
<tr>
<td>$x_{r,j}$</td>
<td>Binary variable equal to 1 if the $r^{th}$ run is the $j^{th}$ job, otherwise equal to 0</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>Large positive number</td>
</tr>
<tr>
<td>$M$</td>
<td>Maximum number of machines</td>
</tr>
<tr>
<td>$m$</td>
<td>Machine index</td>
</tr>
<tr>
<td>$D_m$</td>
<td>Machine release date</td>
</tr>
<tr>
<td>$B_j$</td>
<td>Number of parts for job $j$</td>
</tr>
<tr>
<td>$S_j$</td>
<td>Maximum number of sublots for job $j$</td>
</tr>
</tbody>
</table>
\( s \)  
Sublot index

\( o \)  
Operation index

\( O_j \)  
Maximum number of operations for job \( j \)

\( s_j \)  
Size of sublot \( s \) for job \( j \)

\( T_{o,j,m} \)  
Processing time of operation \( o \) of job \( j \) when assigned to machine

\( S_{o,j,m,o',j'} \)  
Setup time for operation \( o \) of job \( j \), where \( o' \) and \( j' \) denote the previous operation for the previous job processed on the same machine

\( S^*_{o,j,m} \)  
Setup time for the first operation \( o \) of job \( j \) assigned to machine \( m \)

\( L_{o,j} \)  
Lag time for operation \( o \) of job \( j \)

\( A_{o,j} \)  
Binary data equal to 1 if setup for operation \( o \) of job \( j \) is attached (non-anticipatory), or 0 if the setup is detached (anticipatory)

\( R_m \)  
Maximum number of runs for machine \( m \)

\( u \)  
Run index

\( P_{o,j,m} \)  
Binary data equal to 1 if operation \( o \) of job \( j \) can be processed on machine \( m \), otherwise equal to 0

\( c_{o,s,j,m} \)  
Completion time of operation \( o \) of sublot \( s \) of job \( j \) assigned to machine \( m \).

\( \tilde{c}_{r,m} \)  
Completion time of the \( r \)th run on machine \( m \)

\( x_{r,m,o,s,j} \)  
Binary variable equal to 1 if the \( r \)th run on machine \( m \) is the \( o \)th operation of any sublot for the \( j \)th job, otherwise equal to 0

\( y_{r,m,o,j} \)  
Binary variable equal to 1 if the \( r \)th run on machine \( m \) is the \( o \)th operation of sublot \( s \) for the \( j \)th job, otherwise equal to 0

\( \gamma_{s,j} \)  
Binary variable equal to 1 if the size of the sublot \( s \) for a particular job \( j \) is non-zero (i.e. \( b_{s,j} \geq 1 \)), otherwise equal to 0

\( z_{r,m} \)  
Binary variable equal to 1 if the \( r \)th potential run on machine \( m \) has been assigned to an operation, otherwise equal to 0
\( l \)  
Gene location index

\( L \)  
Maximum number of genes for a particular chromosome

\( \alpha_{s,j} \)  
The percentage of which the job should be allocated to that particular subplot

\( t_{o,j,m} \)  
Future completion time

\( \rho \)  
Probability of the crossover operator being applied

\( \sigma \)  
Probability of the mutation operator being applied

\( \theta \)  
Value used to increase or decrease the value of a gene

\( \theta_{\text{max}} \)  
Predetermined parameter with a value that lies in \([0,1]\)

\( \text{rand()} \)  
Number generated randomly that lies in \([0,1]\)
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGV</td>
<td>Automated/Automatic Guided Vehicle</td>
</tr>
<tr>
<td>ARC</td>
<td>Assignment Choice Rule</td>
</tr>
<tr>
<td>BB</td>
<td>Branch and Bound</td>
</tr>
<tr>
<td>FLP</td>
<td>Facility Layout Problem</td>
</tr>
<tr>
<td>FJSP-LS</td>
<td>Flexible Job Shop with Lot Streaming</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>HGA</td>
<td>Hybrid Genetic Algorithm</td>
</tr>
<tr>
<td>IOAM</td>
<td>Intelligent Operations Assignment Mutation</td>
</tr>
<tr>
<td>JLOSC</td>
<td>Job Level Operations Sequence Crossover</td>
</tr>
<tr>
<td>JRQ</td>
<td>Job Release Queue</td>
</tr>
<tr>
<td>LHS</td>
<td>Left Hand Side</td>
</tr>
<tr>
<td>LHSSX</td>
<td>LHS-Segment Crossover</td>
</tr>
<tr>
<td>LP</td>
<td>Linear Programming</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed-Integer Linear Programming</td>
</tr>
<tr>
<td>MLT</td>
<td>Manufacturing Lead Time</td>
</tr>
<tr>
<td>MOR</td>
<td>Most Operations Remaining</td>
</tr>
<tr>
<td>MWR</td>
<td>Most Work Remaining</td>
</tr>
<tr>
<td>OMAC</td>
<td>Operation-to-Machine Assignment Crossover</td>
</tr>
<tr>
<td>OSSM</td>
<td>Operations Sequence Shift Mutation</td>
</tr>
<tr>
<td>PGA</td>
<td>Parallel Genetic Algorithm</td>
</tr>
<tr>
<td>RHS</td>
<td>Right Hand Side</td>
</tr>
<tr>
<td>RHSSSX</td>
<td>RHS-Segment Swap Crossover</td>
</tr>
<tr>
<td>ROAM</td>
<td>Random Operation Assignment Mutation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>RSJ</td>
<td>Randomly Select a Job</td>
</tr>
<tr>
<td>SA</td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td>SLOSC</td>
<td>Sublot Level Operations Sequence Crossover</td>
</tr>
<tr>
<td>SOShM</td>
<td>Sublot Order Shift Mutator</td>
</tr>
<tr>
<td>SOSwM</td>
<td>Sublot Order Swap Mutator</td>
</tr>
<tr>
<td>SPC</td>
<td>Single Point Crossover</td>
</tr>
<tr>
<td>SPOX</td>
<td>Single Point Order Crossover-2</td>
</tr>
<tr>
<td>SPSBOX</td>
<td>Single Point Similar Block Order Crossover</td>
</tr>
<tr>
<td>SPSSOX</td>
<td>Single Point Similar Sublot Order Crossover</td>
</tr>
<tr>
<td>SSStM</td>
<td>Sublot Step Mutation</td>
</tr>
<tr>
<td>SSwM</td>
<td>Sublot Swap Mutation</td>
</tr>
<tr>
<td>SSD</td>
<td>Sublot Size Degenerator</td>
</tr>
<tr>
<td>TAES</td>
<td>Event-Scheduling/Time-Advance</td>
</tr>
<tr>
<td>TAESGA</td>
<td>Event-Scheduling/Time-Advance Genetic Algorithm</td>
</tr>
<tr>
<td>TPOX</td>
<td>Two-Point Order Crossover</td>
</tr>
<tr>
<td>TRADGA</td>
<td>Traditional Encoded Sequence Genetic Algorithm</td>
</tr>
<tr>
<td>TPSSOX</td>
<td>Two-Point Similar Sublot Order Crossover</td>
</tr>
<tr>
<td>TPSBOX</td>
<td>Two-Point Similar Block Order Crossover</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Assembly lines have been used throughout history to manufacture or assemble a product together. Traditionally, parts typically physically move from workstation to workstation until the final product is produced. As technology progressed, more layouts were discovered with an objective to optimize the flow of the assembly plant in order to decrease cost, number of workstations, and final completion time (Corominas et al., 2011). For instance, manufacturing plants started to leave the final product stationary instead of moving the workstations, tools, and parts towards the final product. This layout is referred to as fixed-position layout. Fixed-position layout is beneficial if the part is not mass produced and contained many different variations on the final product. Moreover, if moving the part is not practical (i.e. due to mass and volume), fixed-position layout can prove to be a practical layout (Lin and Liao, 2012).

With the introduction of new products into the market, a wide variety of layouts became available for businesses to choose from. The classical assembly line is the flowshop. In a flowshop, the part is simply moved down the line for completion. Clearly, this is not the most optimal layout to choose for every single application. Moreover, with the introduction of new manufacturing machines
capable of performing a wide variety of functions, a part’s route becomes flexible since it can choose to be processed by the variety of workstations eligible to complete the task. Today, variations in the final product can be addressed by implementing flexible machines (Wilson, 2013) that can perform a variety of processes instead of purchasing new, dedicated machines that require recalibration every time variations are introduced.

Hence, efficient and optimized manufacturing plants play a critical role in producing products for today’s successful businesses. Optimizing for low cost and fast completion time is the trend present in today’s market. Balancing is the act of assigning tasks to workstations in an efficient manner Corominas et al. (2011). Schedulers and controllers balance workstations constantly during new product introduction and machine breakdowns. Schedulers must complete this task in a fast, efficient, and accurate manner. Having an optimized algorithm that delivers a good makespan improves speed and accuracy of the schedule. Businesses that fail to optimize their manufacturing plants through either balancing the workstations or scheduling the processes at the correct machine will become irrelevant amongst the global competition.

1.1. Existing Facility Layouts

The following sections introduce existing layouts of the current facility layout problems in the industry. Different layouts dictate different rules for the heuristic scheduling algorithm. Moreover, algorithms exist both for scheduling jobs and searching for the most optimal facility layout. It would be beneficial for algorithms to integrate both the scheduling of jobs and optimization of the facility layout. However, this is missing in current literature. Integration of multiple factors that ultimately affect the scheduling of manufacturing plants is further discussed in Section 6.2.
1.1.1. Multi-Row Layout

A multi-row facility layout consists of equal or unequal lengths of stations laid out in horizontal rows that are equally spaced. These facility layouts are not limited to manufacturing plants; they exist in libraries when cataloguing books or warehouse storage. Automated/automatic guided vehicles (AGVs) can also be used to decrease material handling time. In the long term, AGVs typically prove to be a more economical option for material handling. The multi-row facility layout has been used as early as the 1960s for computer wiring, scheduling, and facility planning (Hungerlnder and Anjos, 2015). An example of a multi-row layout is shown in Figure 1.1.

![Multi-row facility layout](image)

Figure 1.1: Multi-row facility layout adopted from (Hungerlnder and Anjos, 2015)

A similar layout to the multi-row facility layout is the single row facility layout. The layout is limited to a single row (Kothari and Ghosh, 2013). The objective in this layout problem typically is to minimize the material handling cost (Ravi Kumar et al., 1995). This can be achieved by decreasing the weighted sum of the distances between similar stations beside one another.
Chapter 1. Introduction

Figure 1.2: Multi-row facility layout adopted from (Hungerlnder and Anjos, 2015). Gantry robots can be used to handle materials in this layout

1.1.2. Flowshop Layout

A flowshop layout has machines in a line where the job follows each machine in the same order. It is assumed that each job needs to be processed on each of the machines. This means that the same sequence will be used for the remaining machines down the assembly line (Fernandez-Viagas and Framinan, 2017). Henry Ford popularized the flowshop layout, as it was the first assembly line introduced for the car manufacturing industry in 1914 (Torrero, 1977). For a pure flowshop, the following assumptions are applied. When the setup time depends on its previous process (sequence dependent setup), the setup times are considered negligible. For setup times that do not depend on its previous process (sequence independent setup), the job is assumed to be non-anticipatory – meaning the setup time is attached to the job itself. To be specific, the machine cannot start the setup until the job requiring the setup has completed its previous operation and is available for processing. Machines are also assumed to be available at time zero. Typically, the flowshop facility layout problem becomes an optimization problem.
that focuses on minimizing the makespan (the overall completion time of all the jobs), consideration of the homogeneity of job completion times, or consideration of the total tardiness and earliness.

Figure 1.3: In a flowshop, parts flow strictly in the order of the machines placed down the line

1.1.3. Cellular Layout

A cellular facility layout is when machines with different functions are grouped together, placed in different parts of the manufacturing plant in close proximity to one another (Javadi et al., 2013). The group of machines form a cell. These machines are grouped together so that parts with similar sequences can be manufactured in this cell. Parts may not be identical but require a similar sequence or set of machines to process the required operations. These parts are grouped into part families. Cellular layout can also be coined as group technology. The name is derived from the layout requiring parts to be grouped together based on the similar workstations (or technology) required for processing (Hassan, 1995).

Using a cellular layout can reduce setup time, lot sizes, work-in-process, and a list of other quantitative factors listed in Javadi et al. (2013), which typically leads to a decrease in overall makespan before optimizing the schedule is
required. This leads to improved productivity and reduced material handling costs (Hassan, 1995). However, these benefits depend on the type of product being manufactured. Although many factors determine whether or not a cellular facility layout is necessary, a simple factor before extensive studies are conducted is the production quantity. Typically, cellular layouts are suitable for a medium amount of variation in the product and a medium production quantity. A graph is shown in Figure 1.4 to clearly show which type of layouts are preferred based on variety in the part and production quantity.

![Graph showing appropriate layout to be used depending on product numbers](Sayed and Lash, 2014)

1.1.4. **Jobshop Layout**

In the jobshop layout, machines are strategically placed on the floor while parts enter into the jobshop and are processed by their eligible machine Mati et al. (2011). The flexible jobshop problem assumes there are $n$ jobs to be completed on a set of $m$ eligible machines. The jobshop facility layout is also a form of cellular
manufacturing. Introducing different machines that are eligible to process many operations of different jobs improves flexibility. This tends to lead to reduced setup times, as there may be a variety of setup times available for the scheduler to pick from. As such, material handling time and cost become bottlenecks in this facility layout optimization problem Kamoun et al. (1999). In the scheduling problem for this layout, the optimization problem typically leads to minimization of either cost or makespan. A diagram of the jobshop layout is shown in Figure 1.5.

![Diagram of Jobshop Layout](#)

Figure 1.5: In the jobshop layout, parts can come in and use any eligible machine to process its operations before completion

The jobshop layout can typically model flowshop models. This is due to the fact that parts are able to flow into any machine or workstation and be completed in any order. In a flowshop, there is a specific order that all parts must follow before completion. Hence, if the order was imposed onto the jobshop model, then the jobshop model can handle the flowshop problem. This means that most GAs able to solve jobshop problems are able to solve flowshop problems if they are
setup to follow the certain order existing in the flowshop model. An example of this is in Defersha and Chen (2010b). Note that this may not be the most efficient approach to obtaining a good solution or makespan.

1.1.5. Dynamic Layout

The aforementioned layouts are considered static layouts, where it is assumed the layout does not change over time. However, as time moves forward and new changes and products are introduced to the manufacturing plant, change is inevitable and facility layouts need to accommodate this. Much of the research existing today focuses on optimizing static layouts, which serve only as a good starting point for optimizing real-world problems that exist in the industry (Ripon et al., 2010).

Dynamic facility layouts are physical layouts that change over time due to the change in flow of processes that need to occur when a new change or product is introduced in the assembly line. Hence, the number of possible solutions considered is significantly increased compared to the static facility layouts. The genetic algorithm (GA) has been chosen to tackle these problems due to its ability to quickly generate a solution compared to other algorithms. In Ripon et al. (2010), the approach was to solve and optimize the static facility layout problem for one particular period in order to determine the layout to be used. Then, for the next period (where product change or introduction is added), the GA determines whether it is optimal to change the layout of the plant or leave it as it is in an attempt to decrease the distances between the facilities (thereby decreasing material handling cost). For a problem consisting of 6 departments spanning 3 periods, an optimal layout generated by Ripon et al.’s GA may be as shown in Figure 1.6.
Chapter 1. Introduction

Figure 1.6: Dynamic layout for a 6-department (departments indicated by the numbers), 3 period problem arranged in a 2 by 3 facility layout adopted from (Ripon et al., 2010)

1.2. Simulation and Mathematical Modelling

Simple facility layouts with rudimentary or small-sized problems can be solved and verified with a Mixed-Integer Linear Programming (MILP) problem. The optimization problem can be solved with a Branch and Bound (BB) algorithm. However, as these models become more complex due to the increase in problem size and complexity of facility layout, the MILP BB method is no longer viable due to its slow computation time. Hence, the use of GAs is necessary to solve medium-sized problems. Real world problems or large-sized problems even pose a computational challenge for the GAs that exist in current literature. This is not due to lack of computational power; rather, due to the design of the framework of existing GAs. This thesis seeks to lay a framework that reduces the load in the global search by decreasing the size of the chromosome in order to allow a quicker convergence to a good solution. The creation of models and the running of simulations should only be proceeded if they can be completed in a timely manner. They must be completed where the duration of time spent researching, developing, and executing the simulation is worth the time and money lost stopping production of the newly introduced product (Banks et al., 2004). Hence, timely results in a dynamic environment are key to decreasing the negative impacts associated with downtime.

GAs are not limited to scheduling, they have been used in designing an
optimal layout of all the machines in a facility. The facility layout design process has matured since the 1970s with multiple facility layout options available; hence, introducing many facility layout optimization problems for today’s research. The GA has been the typical route to solving these complex optimization problems. The algorithm and modelling was introduced by Holland in 1975 (Azadivar and Wang, 2000). Moreover, GAs should not be limited to optimizing certain aspects in a manufacturing plant separately. An ideal GA should encompass the layout, scheduling, inventory, metrics, and dynamic environments in order to provide a result that optimizes multiple factors. This is left for future research, discussed in 6.2. While the BB method for solving MILP problems can provide exact results, GAs search for the most optimal solution that may not necessarily be the exact solution. The procedure of GAs will be elaborated upon in Section 4. Moreover, depending on how the GA is built, GAs can be stuck in the local minima instead of the global minima.

When solving facility layout problems, random numbers are typically introduced when determining the initial permutation of sequences or chromosomes. Unexperienced users of GAs will worry that GAs are unable to produce the same answer each time the algorithm is run. To clarify these misconceptions, (and for the purpose of this thesis,) the initial starting point at which the random numbers are chosen, called the seed number or initial seed point, can and will be kept constant throughout the simulation study. A list of random numbers to use is set before the study is conducted. The random numbers to be used is based on the initial index at which the random numbers are chosen from the list. An in-depth tutorial is available in the textbook by Banks et al.
1.3. Organization of Thesis

The outline of the thesis as follows. In Chapter 2, we will present a literature review covering the following topics: introduction to scheduling, the current GA methodology, the current GAs that exist for the facility layout design problem, the current GAs that exist for the scheduling problem, current math modelling practices, dynamic environments, and how this thesis is positioned in current research and literature. In Chapter 3, a mathematical model will be presented which will be used for the validity of the two GAs presented in this thesis. Moreover, a rudimentary model will be presented to further the readers’ understanding of how a mathematical model for facility layout and scheduling is derived. In Chapter 4, two GAs will be analyzed and their performance will be compared in Chapter 5. The first GA analyzed is adopted from Defersha and Chen (2012a), which presents a flexible flowshop model and considers lot streaming. The second GA is newly introduced, which uses a event-scheduling/time-advanced (TAES) as the decoder for the GA. Their results and conclusions will be summarized in Chapter 6.
Chapter 2

Literature Review

2.1. Introduction to Scheduling

Optimizing the facility layout problem is a crucial step in surviving the global competition. The typical result from optimizing the facility layout problem is 10-30% material transportation cost savings (Kuab et al., 2011). Good facility layouts are able to address 20-50% of the operating cost that is typically related to material transportation and facility design (Kuab et al., 2011). Aside from optimizing the physical facility layout, optimizing the schedule at which jobs are dispatched into the available resources or machines for processing is also important in facing today’s competition. Numerous research articles provide extensive studies for both optimization methods manufacturing plants choose to invest in.

Scheduling is defined as the act of assigning jobs to the existing manufacturing resources available to execute the product outlined by the process plan. The process plan determines the eligible machines and sequence in order to manufacture the job (Zhang and Wong, 2015). The need for scheduling jobs in manufacturing plants has risen as today’s products depend on the manufacturing plants they are being produced in. In order for these businesses to survive, they
must have optimal plants in order to remain relevant in today’s global competition. Many companies, such as Dell and Toyota, are leaders in the time-based competition of optimizing makespan in order to quickly produce products for consumers (Chang and Chiu, 2005a). If more products are produced, the cost of manufacturing can be offset by the sale of the product. This leads to an increase in profit. Decreasing the manufacturing lead time (MLT) can lead to a reduction in makespan. Techniques to decrease the MLT include introducing just-in-time manufacturing, lot streaming, facility layout rearrangement, and optimal scheduling. The dynamic environment needs to be accounted for in order to address any setbacks that arise. Hence, rescheduling is a common practice many manufacturing plants in order to reduce the negative side effects of the setbacks.

2.2. Genetic Algorithm Methodology

Classical methods such as MILP BB prove to be difficult for solving NP-Hard problems. Hence, many researchers opt to use GAs to tackle scheduling problems with high computational complexity as it is the most promising method (Jia et al., 2011). Moreover, the MILP BB models typically serve as a verification tool to ensure the accuracy of the GA model. The accuracy of a model can be determined by comparing its result with either other linear programming (LP) methods, such as MILP, or another verified GA suitable to solve the same problem. This has been done numerous times in Defersha and Chen (2012a), Defersha and Chen (2012b), and Defersha and Chen (2010b). This thesis seeks to do the same by verifying the newly proposed algorithm in Section 4.3 with a MILP model discussed in Section 3.1.2 and a former GA discussed in Section 4.2. GAs were initially introduced by Professor John Holland in the 1960s and further developed in 1975 (Azadivar and Wang, 2000) by his students in University of Michigan (Mitchell, 1995). The initial purpose of developing the GA
was to explore how nature, adaptation, and evolution can play a role in computational problems. It is part of the family of evolutionary-inspired algorithms in the field of metaheuristics (Doerner et al., 2007). The GA was then taken to optimize scheduling problems, where the fitness of a chromosome was typically set to the objective function. Typical objective functions in early research included minimizing cost or makespan. Current and future research attempts to optimize multiple variables, trying to address all factors that a scheduler typically encounters.

GAs usually struggle in finding a good solution. The ability to search for a optimal and acceptable solution using the genetic algorithm requires a fine balance between the two parameters: exploitation and exploration (Zhong and Huang, 2009). Using only exploitation means that the search is only searching for the most optimal solution. Using only exploration means that the search is searching everywhere in the solution space, which may not necessarily lead to an optimal solution. The time to convergence may be extremely slow. A balance between the two parameters allows for the solution to converge, while not being stuck at a local optimum due to too much exploitation. Fine tuning parameters becomes an art since different problems with different sizes call for different adjustments to each parameter. Hence, many runs of the GA with different parameter settings are required before a successful and acceptable solution can be reached.

Similar behaviour can be noted when adjusting the population size for each iteration of the GA (Zhong and Huang, 2009). A population size too small is similar to high exploitation. Consequently, the search has the possibility of becoming stuck in the local optimum. A population size too large is similar to high exploration. The search becomes too random; hence, converging on a solution can become difficult for the algorithm.

The length of the code is also another aspect that needs to be fine tuned correctly from the design stage of the GA. The length refers to the resolution
of the gene (Zhong and Huang, 2009). Increase in chromosome size lead to an increase in search area. GAs search long place-values in genes inefficiently; thus, there needs to be a balance between search efficiency and precision of the result (Zhong and Huang, 2009). This thesis seeks to develop a better GA that significantly decreases the resolution of the gene and length of chromosome by using a TAES simulation as the decoder of the GA (Banks et al., 2004). This systematic approach is hypothesized to be more efficient in scheduling jobs; however, at the cost of precision of the result. The result and answer to this hypothesis is stated in Section 6. It must be noted the precision of the result is slightly less than the old GA. However, the difference is negligible when applied to real world applications. Moreover, the speed of which the results are delivered is significantly faster. This offsets any results that are less accurate since it can quickly reschedule in the case of a setback (i.e. machine breakdown).

Due to the nature of GAs, they are strong in searching the space globally for the optimum fitness chromosome. It only focuses on modifying the chromosomes with high fitness, which results in an efficient but general and global search (Zhong and Huang, 2009). As such, GAs typically struggle when it comes to narrowing down the search results so that a local optimum is obtained. Other methods can be implemented in alongside the GA in order to improve the local search. When other methods are implemented alongside the GA, the GA is termed Hybrid Genetic Algorithm (HGA). Many GA users today will create a HGA to improve the exploitation capacity once the global search of the GA has discovered a high performance or high fitness search area (Tang and Pan, 2015). In Zhong and Huang (2009), simulated annealing (SA) is suggested to be used alongside the GA to improve the local search. Incorporation of any additional optimization procedure (usually as an extra procedural step of the GA) is what makes a GA hybrid, as demonstrated in Elattar (2015), Tseng and Lin (2010), and Tang and Pan (2015). Similar to other GA users, this thesis uses the TAES
algorithm alongside the GA (in the decoder) attempting to outperform the traditional GA. The Event-Scheduling/Time-Advance Genetic Algorithm (TAESGA) aims to develop a scalable model that addresses longer problems in size and complexity, while leaving room to add additional metrics in future research.

2.3. Genetic Algorithms Applied to Facility Layout Problems

SA is commonly used and combined to solve the facility layout problem (FLP), as demonstrated in Kuab et al. (2011). The FLP is slightly different from the scheduling optimization problem as it deals with optimizing the physical layout while considering the area of a plant. SA is a stochastic search procedure that repeatedly takes adjacent solutions to check if it is better than the currently solution. If so, it accepts it as the new best solution. If not, a probability of $\exp(-\Delta/t)$ is assigned to the adjacent solution where it is $\Delta$ worse than the current solution and $t$ is the temperature parameter (Portal et al., 2016). SA is based on the concept of annealing in metallurgy, a thermodynamic process that deals with how liquids freeze and crystallize or how metals cool and anneal to reduce defects (Kosmas and Vlachos, 2012). A random walk search is performed on higher temperatures and a stochastic local search is performed on lower temperatures. A sufficiently slow cooling cycle where the algorithm starts with the high temperature and cools down to a lower temperature allows for the SA to converge to an optimal solution.

Since GAs are weak in local search and strong in global search, Kuab et al. (2011) opted to use SA to handle the local search. SA can be used to find the global extremum even if it is hidden in poorer local extrema (Kosmas and Vlachos, 2012); however, in this case, SA is only being used to find the optimal solution since GA’s inefficiency lies in performing the search in a local search space. The combination of both allows for a faster search to the optimal solution.
in comparison to only using GA. Better solutions can be obtained if the only responsibility of the GA is to perform the global search, thereby preventing it from being stuck in a local minima.

In Kuab et al. (2011), parallelism was also applied to the algorithm. To parallelize a GA means to perform both decoding and evolution (including iterations of the genetic operation) in parallel. This is termed parallel genetic algorithm (PGA). PGAs improve both the quality of the solution and the computational performance (Qi et al., 2016). Today, it is common for parallelism to be applied to GAs. However, there are other computational performance issues that still exist in PGAs and GAs.

Wang et al. pointed out the performance issues due to poor parameter selection in metaheuristics (SA, GA, and the tabu search technique). Researchers used a two-stage GA in order to tackle the issue of lack of robustness in parameter setup. As stated before, fine tuning parameters is an art since parameter setup differs depending on the problem being solved by the GA. Improper parameter setup can easily blind the GA into being trapped in a local minima, which may not be the most optimal solution (global minima). Hence, setting up the correct parameters depending on the problem being solved becomes an optimization problem that another GA can solve. Wang et al. (2008) provides an example of how modern research combines different GAs together to approach a problem by combining the preliminary GA that finds the best parameter combination with the secondary GA that solves the scheduling problem. The performance of searching for the best optimal schedule was greatly improved when the (secondary) scheduling GA parameters were optimal before starting the (preliminary) scheduling GA. Current research demonstrates the ability to take many optimization tools and combine them through software programming in order to accomplish searching for a solution for complex scheduling problem. This is not limited to combining GAs together. This thesis is aligned with current research in that it too seeks to
optimize current GA search practices so that computation time is decreased, the prospect of being stuck in a local minima is decreased, and the overall algorithm is scalable when industry problems increase (not only in size but in complexity).

In Hasan et al. (2011), the NP-hard scheduling problem also experienced the same issues many other scheduling problems have experienced: inefficient local search or converged solution stuck in a local optimum. Hence, Hasan et al. introduce a new technique called Shifted Grip-Reduction to improve the local search performance. This heuristic rule finds any gaps that exist between two consecutive jobs on a particular machine. If a job can be inserted into the gap without violating any precedence constraints, it will improve the overall fitness and makespan. Machine unavailability and machine breakdown is also incorporated into the scheduling GA. The GA outperformed existing metaheuristics by comparing its performance through benchmark problems. Current research continuously expands existing algorithms to include complex situations (such as staff inavailability or machine breakdown) that arise in the real-world industry. This pertains to this thesis since it is realized that GAs must be expandable to easily incorporate the surprises an industry encounters when operations are running in real-time. Hence, the development of this new, expandable, and scalable TAESGA has a place in today’s research when tackling larger problems with pertinent issues that are unaddressed in literature.

Today, scheduling is not limited to assigning the jobs to available resources at a certain time, but the scheduling of staff is also considered in order to reduce cost while still meeting demand requirements and considering different type of staff members (i.e. day vs. night shift workers). In Cai and Li (2000), researchers develop a GA that has multiple objectives (multi-objective) that need to be optimized. Instead of typically focusing on only one objective (i.e. minimizing cost or makespan), Cai and Li focused on minimizing cost, maximizing staff surplus, and minimizing variation of staff surplus. While there is a growing amount of
research focusing on developing algorithms for different types of scheduling problems based on different layouts, optimization of facility layouts, and reduction in computational time while increasing performance of existing algorithms, there is a limited amount of research with multi-objective optimization for the aforementioned algorithms. Yet, multi-objective problems are important to address since real world problems have multiple factors that ultimately influence the schedule and inventory control. Hence, future work of this existing research will include multi-objective optimization further explained in Section 6.2.

2.3.1. Flowshop Problem

In Defersha and Chen (2012b), the research builds upon existing research of the pure flowshop problem by adding lot streaming. In the scheduling context, hybrid is defined as the inclusion of both single and parallel stage/stations in the problem. Flexible refers to the ability of certain jobs to be processed by more than one eligible machine. There was a gap in literature where lot streaming and hybrid flowshop were previously never combined in mathematical models.

Lot streaming was first introduced and termed in Reiter (1966). Its prime objective upon inception is to address jobs with long operation processing times followed by short operation processing times. In this particular scenario, finished jobs in a single batch are held up by other jobs in the batch waiting to be finished due to its long processing times. As a result, the completed jobs cannot move onto the next machine; even if the next machine has a short processing time. It is advantages to split the batch of jobs into smaller sublots. As long as the process is completed, the scheduler is able to ”stream the lot ahead” to the next shorter operation. This would decrease the makespan in production plants. However, the additional setup costs incurred could offset the time advantage gained.

This model in citedefersha2012 adds lot streaming, which laid foundation for many other researchers to build upon (further explained in Section 2.3.3).
Moreover, this model is specific and designed for solving flowshop problems. It is unable to expand and solve other scheduling problems or layouts that exist in the real world. Hence, Defersha and Chen developed the jobshop GA. In today’s literature, it is common to have a dedicated GA for each particular scheduling layout problem. Each GA can then be tailored to be efficient for that particular problem. However, this poses an issue when companies switch their layouts in the dynamic real world environment. Instead of creating many different GAs, this thesis will lay a generalized foundation to solve most scheduling problems in the TAESGA.

2.3.2. Jobshop Problem

The jobshop problem is still relatively new within the past decade. One of the pure jobshop scheduling problems tackled in research was Penn and Raviv (2009). Here, GA was not used to solve the problem. Instead, an algorithm was developed to dispatch the jobs in an efficient and logical manner following a rule that would produce the lowest makespan. Similarly, in this thesis, a dispatch rule is used specifically for the decoder. This is to efficiently dispatch the jobs to the eligible machines so that the computational load on the GA is reduced, since local search is not a strength of GA in comparison to a broad, global search. Moreover, the research objective in Penn and Raviv (2009) was to maximize revenue over the planning horizon instead of minimizing the makespan. Using an dispatch algorithm may not be the most computationally efficient method in optimizing the problem; research has since moved on and improved by relying on GAs to solve the optimization problem.

In Azardoost and Jamshidi (2010), GA was used to search for an optimal solution to the jobshop problem. For the probabilities applied on each GA operator, a linear regression equation with the lowest total square error is used to determine the specific percentage of every probability. As stated above, fine tuning the
parameters is critical for obtaining a good solution (Wang et al., 2008). Linear regression is a regression is used to determine if there is a relationship amongst the variables. The dependent variable is a linear combination of parameters as coefficients. Minimal square error occurs when the linear regression equation of the line intersects with the optimum combination of parameters. Hence, the specific parameter probabilities are obtained in this manner. This properly sets up the GA. Moreover, the problem tackled expands upon the pure jobshop problem as a flexible jobshop problem, meaning that jobs had more than one eligible machine to process the operations.

Around the same time, in Defersha and Chen (2010b), a mathematical model and GA was developed to solve the flexible jobshop problem. Several factors have been taken into consideration for the model, such as sequence dependent setup times, attached or detached setup, machine release/availability date/time. Computational performance was also measured as simulation models needed to be efficient enough to solve real-world problems in a timely manner (Banks et al., 2004). While the former model focused on parameter setups, this model focused on expanding the jobshop mode to incorporate real world complexities.

Similar to many of the existing scheduling problems mentioned above, researchers typically take existing GAs and improve their local search. In Pandian et al. (2012), they improved the GA’s search by implementing jumping genes operation in the local search process. The optimization objective is to minimize AGV flow time and minimize makespan. Jumping genes are operators that take two randomly selected genes and switch them with the two randomly selected genes in the secondary chromosome. Further detail of procedure of the jumping gene operator can be found in Pandian et al. (2012). This operator is similar to the assignment operator used in Defersha and Chen (2010b). However, no child chromosomes are formed when using the jumping genes operator. this literature demonstrates that GAs can be improved by having proper operators that
Chapter 2. Literature Review

give a better result or deliver a lower computational time. Today’s literature constantly aims to improve computational efficiency. This can address dynamic environments, discussed in Section 2.5.

In Defersha and Chen (2012a), lot streaming was added to the jobshop scheduling model. Majority of the research for lot streaming was on the flowshop problem, found in Defersha and Chen (2010a), Martin (2009), Marimuthu and Ponnambalam (2005), Kim and Jeong (2009), and Gmez-Gasquet et al. (2013). Hence, developing a lot streaming model for the jobshop problem was necessary to further advance scheduling research. It can be argued that the jobshop model is more important than the flowshop model.

It is important to notice that jobshop models can be used to solve flowshop scheduling problems. The sequence of machines needs to be readjusted such that it is in sequential order, like flowshop setups are. However, utilizing the models in Defersha and Chen (2010b) and Defersha and Chen (2012a) to solve flowshop scheduling problems may not be efficient even if possible. This is due to the fact that the search developed in Defersha and Chen (2012b) is tailored specifically for flowshop models. This thesis seeks to redevelop the specialized GA into a generalized GA with the TAES as the algorithm of the decoder so that it can tackle both flowshop and jobshop problems in a computationally efficient manner.

2.3.3. Direction of Improvement

Since the inception of the researched done on the flowshop and jobshop problem, numerous researchers have extended and advance today’s literature in many different directions – highlighted in the next section. Demir and Ileyen expands upon Defersha’s research by considering overlapping in operations. This means that the constraint for a particular job where the previous operation needs to be completed before starting the next operation is relaxed. A transfer batch or sublot can be sent to the next required operation while the remainder of the
portion of the job is finished processing. The research also proposed a mathematical model to solve small problems before creating a GA to solve larger, real-world problems. Evidently, computational time remains a factor to consider in scheduling research since Demir and Ileyen created a GA with lower computational time than GAs lacking the consideration of overlapping operations. Other researchers chose to implement the consideration of cost and demand in their papers. Moslemipour further considered the time value of money and the random fluctuation of demands in its proposed model. Therefore, inventory control should be considered in future research, further discussed in Section 6.2. In developing new GAs to tackle new scheduling problems, operators in existing research are shared when tackling the new scheduling problems as exemplified in Asefi et al. (2014). This thesis builds upon existing operators in Defersha and Chen (2012b) in order to build a GA with lower computational time and better scalability and expansibility. This is further detailed in Section 4.2.5.

Further expanding upon Defersha and Chen’s research, Mortezaei and Zulkifli redeveloped the mathematical MILP model to account for machine downtime due to preventative maintenance performed on machines. Shafigh et al. also expanded on Defersha’s research by tackling dynamic distributed layouts while considering the layout configuration and production planning. The direction of that research focused on expanding the complexity of current models when simulating real-world, industry problems that are currently present. However, other researchers have focused on refining existing research either by decreasing computation time or finding near-optimal solutions (Nouri et al., 2017). Moreover, Fadaei (2013) explored how different sublot sizes affected the makespan of scheduling problems. Their research concluded consistent sublot sizes and using iterations with a mix of both the parent and offspring pool in the population provided the lowest makespan. However, this research assumed that machines are always available, which is not realistic in real-world industry problems. It
was further suggested that in Chang and Chiu (2005b) that realistic sublots are never equal sized in real-world situations and are generally, most-likely not equal sized when searching for the most optimal solution. In short, today’s scheduling literature has developed and broadened from the simple goal of optimizing a scheduling problem. It is still relatively new area of research due to the amount of area in literature that can still be improved upon.

2.4. Current Math Modelling Practices

Majority of research on modelling scheduling problems typically follow two common notations. The first style of notation is sequence-position variable based model. The second style of notation is a precedence variable based model. In Bayat Movahed (2014), the thesis compares two models using the two different styles of notation. The thesis provides a strong thorough explanation of the differences for the interested reader.

In the sequence-position variable based model, the computational load is less than a precedence variable based model. This is due to the fact that all possible combinations must be expressed in the precedence variable based model. Whereas, in the sequence-position variable based model, the number of runs can be set to a lower amount than the theoretical maximum number of runs. Hence, this decreases the computational load because the size of the model is smaller.

2.5. Addressing Dynamic Facility Layout Scheduling

Existing research on dynamic scheduling is limited. Most scheduling algorithms assume that the environment is not changing. Disturbances such as a machine breakdown are not modelled (Cheng, 2009). These changes are dynamic and need
to be accounted for in real-world scenarios. Rescheduling is required in order to minimize the increased makespan due to sudden changes Liu et al. (2015). Real time scheduling has been introduced in order to tackle the dynamic environment scheduling problems live in. Typically, researchers have chosen not to factor in the random machine breakdown. To be specific, if a particular/specfic machine is known to breakdown, this can be factored; otherwise, it is difficult to know which random machine at any given point of time will break down. Thus, researchers have chosen to optimize their evolutionary search methods, in order to quickly reach a satisfactory solution that typically lowers makespan or reduces cost. In Rossi and Dini (2000), researchers developed a GA orientated for the dynamic environment schedulers and controllers face in a flexible manufacturing system. The GA was able to solve less complex problems. However, the literature noted that further research was needed for larger, more complex problems (i.e. real-world problems). This demonstrates the limited application of its GA due to the slow computational times associated with larger problems.

Scheduling and rescheduling also impacts the inventory control of a business. Hence a model integrating the two aspects is vital for reducing costs in a business. There is limited literature on the integration of the two. In Ni et al. (2013), the integrated model was able to reduce warehouse inventory by 20% and increased revenue significantly when accounting for inventory planning. Due to the limited literature, there is room for future research in having inventory planning integrated in the GA. This thesis seeks to create a framework that allows for many other metrics and factors to be integrated into the GA in the future.

None of the aforementioned research in the dynamic environment was able to tackle on larger scheduling problems that exist in the real world. The limited literature allows this thesis and future research to advance current literature by providing faster GAs that minimize makespan with satisfactory solutions with short computational time.
2.6. Purpose of Thesis

Using the existing research in Defersha and Chen (2012a), the paper presents a GA that uses traditional methods to solve the jobshop scheduling problem. This GA is called the Traditional Encoded Sequence Genetic Algorithm (TRADGA). Although the TRADGA can solve both jobshop and flowshop problems, using TRADGA to solve flowshop problems in comparison to jobshop problems is inefficient. Moreover, the GA is inefficient for large problems due to the size of the chromosomes. Since it encodes the operation and machine in addition to the job and sublot, the chromosome carries a lot more detail and GA operators can form many different sequences. This means that most, if not all, solutions are searched and explored. However, this can be computationally exhausting for larger problems if all solutions are searched and fitness is calculated for each chromosome.

In this thesis, a modified GA is developed with a new decoder. The decoder uses a TAES algorithm that uses logic to assign the eligible machines per flexible operation (routing problem). Tackling the routing problem in this manner allows for less idling time. This new decoder also allows the GA to exclude the operation and machine from the chromosome, since the heuristic assumes that the only the first operation of a particular sublot for a job (dictated by the gene) is input into the system and stays in the system until the job has been fully completed. This translates to the real-world since jobs are not taken from a warehouse and transported into the plant until necessary, thereby saving material handling costs. Hence, the smaller chromosome allows for larger problems to be simulated with reasonably good solutions. Of course, smaller problems simulated by the former GA will possibly reach a better solution since the search space is explored in much greater depth in comparison to the proposed GA. Yet computational efficiency is in the proposed GA should be greater than current research available today.
Chapter 3

Mathematical Model

Having a mathematical model serves as a reference to ensure GA is accurate. Small problems can be solved for both in the GA and using a BB method to solve the MILP in order to compare numerical results. BB can be used to solve smaller problems. A small problem solvable by both the MILP and GA models will be used to test the validity of the models. Both solutions should be similar, with little to no variation.

A job represents one particular, unique part that must be machined by numerous machines. A station or stage (terms used interchangeably) is typically used to describe a group of machines that perform the same task. For example, the rudimentary model in Section 3.1 has only one machine and one stage. That machine exists in stage one. The term, run, is used to describe the processing of one particular job.

Sequence dependent setup times exist commonly in many manufacturing plants. Depending on the job that has just been processed, the following job will have a unique setup time based on its processed, preceding job. Sequence dependent setup times naturally exist in manufacturing plants. A job that has finished drilling with the same tool may have another job following it such that it too requires the same tool on the drilling machine. The setup time for a job
would naturally be less if its preceding job utilized the same required tool. Note that time represents the total amount of seconds elapsed in the system. It is not to be confused with duration of processing time. Time in mathematical models is typically measured with an integer value with a unit specified in the beginning stages of creating the model. The aforementioned terminology can be clarified once reviewing Fig 3.1.

![Machine 1 diagram](image)

Figure 3.1: Notice how each run is assigned to only one job. Here the completion time of all the runs is 1300 seconds

### 3.1. Rudimentary Model

In this section, we present the problem description of a rudimentary model and the proposed MILP optimization model for the single machine, sequence dependent setup problem addressed in this paper. The rudimentary model is derived from an existing model in Defersha and Chen (2012b).

In a LP optimization model, there is an objective function with constraints. The objective of optimization is to maximize or minimize your objective function, subject to the constraints imposed for the problem. In this paper, we must minimize the total time required to complete all runs (makespan). Once the objective function and its constraints are made, a solution can be calculated using numerous optimization techniques available today. The purpose of this Section is to derive a model that may be used in any programmable optimization software package.
3.1.1. Problem Description and Assumptions

Consider a small manufacturing plant consisting of one single machine. There is only one stage that contains this single machine. Hence, all jobs must be assigned to be processed at this given stage. For each job, there is a sequence dependent setup time on the single machine. The problem is to determine the sequence of each job to be processed by the single stage. The number of runs is equivalent to the number of jobs.

3.1.2. MILP Model

Let $J$ represent the maximum number of jobs. It has been stated that the maximum number of jobs must equal the maximum number of runs, denoted by $R$. Hence, we have

$$J = R. \quad (3.1)$$

This is considered an input constraint. The constraint is enforced initially by the user during the setup of the problem.

Each job will be indexed by $j$ or $p$. Thus, we have $j = 1, 2, ..., J$ or $p = 1, 2, ..., J$ jobs. There will be $r = 1, 2, \ldots, 3$ runs. The requirement of using two different variables will be exposed later in this paper. The variables $j$ and $p$ act as indices that indicate the job number required for a particular term or equation. Moreover, job $j$ must follow job $p$ in sequence. To clarify, job $p$ always occurs first sequentially; after that job $j$ must be performed. An example will follow shortly to show how $j$ and $p$ are used.

Before the example can be exposed, we must introduce one more variable. Let $S_{j,p}$ be the sequence dependent setup time for a job $j$ that follows $p$. Let $S_{j,0}$ be the initial setup time if job $j$ is the first job the machine processes.

Suppose we have a total of $J = 3$ jobs that must be completed. Then, there must be a total of $R = 3$ jobs since the number of jobs must equal to the number...
Table 3.1: Completion times for the jobs in Figure 3.1

<table>
<thead>
<tr>
<th>Completion time of job 3</th>
<th>300 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion time of job 1</td>
<td>750 s</td>
</tr>
<tr>
<td>Completion time of job 4</td>
<td>1050 s</td>
</tr>
<tr>
<td>Completion time of job 2</td>
<td>1300 s</td>
</tr>
</tbody>
</table>

of runs. If the optimal sequence or processing order was calculated to be job 1, followed by job 3, and finally followed by job 2, then the total sequence time can be represented algebraically as

\[ S_{1,0} + S_{3,1} + S_{2,3}. \] (3.2)

This example exposes why two variables, \( j \) and \( p \), must be used in this mathematical model.

Let \( T_j \) represent the processing time for job \( j \). Processing time is not sequence dependent for this problem. To prevent confusion for the reader, Fig. 3.2 clearly illustrates how the variables \( j \) and \( p \) will be used in this paper.

The variables introduced above are assumed to be known and inputted into the MILP model before solving. Note that \( j \), \( p \), \( J \), \( R \), \( T_j \), and \( S_{j,p} \) are indexes and input data.

Let \( c_j \) be the completion time for job \( j \). Referring to Figure 3.1, the following table illustrates the completion time variable and values.
Table 3.2: Completion times for the runs in Figure 3.1

<table>
<thead>
<tr>
<th>Completion time of run 1</th>
<th>300 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion time of run 2</td>
<td>750 s</td>
</tr>
<tr>
<td>Completion time of run 3</td>
<td>1050 s</td>
</tr>
<tr>
<td>Completion time of run 4</td>
<td>1300 s</td>
</tr>
</tbody>
</table>

Let $c_r$ be the completion time for the $r^{th}$ run. In Figure 3.1, the following table illustrates the completion time variable and values.

A constraint will be presented later on for the relationship between the completion time for job $j$ and the completion time for run $r$.

It can be concluded that the maximum completion time, $c_{\text{max}}$, is equivalent to the last run processed in the system. The makespan of a schedule is a term used to define the maximum completion time of the entire manufacturing system. It is obvious that

$$c_{\text{max}} \geq c_j$$

(3.3)

for all jobs (denoted by $j$). Notice in Fig. 3.1 and 3.2 the only instance or possibility for $c_{\text{max}}$ to be equivalent to $c_j$ is when $j$ is the last job. In Fig. 3.1, job 2 was the last run. Hence, $c_2$ was equal to 1300 s, which was the $c_{\text{max}}$ in that figure. In Fig. 3.2, job 2 was also the last run. Hence, $c_2$ was equal to 1100 s, which was $c_{\text{max}}$ for Fig. 3.2.

We must use a binary variable to let the model know which job it is calculating. Its use will become clearer once its constraints are introduced. Let $x_{r,j}$ be a binary variable equal to 1 if the $r^{th}$ run is the $j^{th}$ job, otherwise it is equal to 0.

In this problem, each job can only be completed once. All jobs must be assigned to the single machine. We can express this constraint with the following equation:

$$\sum_{i=1}^{r} x_{r,j} = 1$$

(3.4)

for all jobs, $j$. Furthermore, one run can only be assigned to one job. This is
constrained by
\[ \sum_{j=1}^{n} x_{r,j} = 1 \]  
(3.5)
for all runs, \( r \). Indicies \( i \) and \( j \) begin at 1.

Let \( \Omega \) represent a huge positive number, such as \( \infty \). Notice that in this model, the completion time of the run is equivalent to the completion time of its corresponding job. This can be constrained by
\[ c_r \geq c_j + \Omega x_{r,j} - \Omega \]  
(3.6)
and
\[ c_r \leq c_j - \Omega x_{r,j} + \Omega \]  
(3.7)
for all runs \( j \) and all jobs \( j \). It is required (when programming in any optimization software package) to split this constraint into two equations that mathematically state \( c_r \) must be equivalent to \( c_j \) for the corresponding run, \( r \), and the corresponding job, \( j \).

Referring to Fig. 3.1, it is evident that no job can start before time at 0. This is constrained by
\[ c_1 - T_j - S_{j,0} - \Omega x_{1,j} + \Omega \geq 0 \]  
(3.8)
for all jobs, \( j \).

Also, no job can start before its previous job has completed. This is constrained by
\[ c_r - T_j - S_{j,p} - \Omega(x_{r-1,p} + x_{r,j}) + 2\Omega \]  
(3.9)
for all runs, \( r \), and all jobs, \( j \) and \( p \).

Equations 3.3 through 3.9 are the constraints that exist in the MILP model. The objective for this particular problem is to minimize the setup times subject to these constraints. Determining the minimum makespan of the schedule will also determine the sequence that gives the minimum setup times. Note that \( Z \) is
Chapter 3. Mathematical Model

standard notation used in most (if not all) optimization problems denoting the objective function. Hence, the objective function for this optimization model is to minimize

\[ Z = c_{\text{max}} \]  \hspace{1cm} (3.10)

subject to the constraints presented in Equations 3.3 through 3.9.

Above, we have clearly defined what the makespan, \( c_{\text{max}} \) is equivalent to. More importantly, we have accounted for the setup time depending on its sequence through the constraints introduced earlier. Also, we have successfully restricted each job to be assigned only once in the constraints. Entering this model into any optimization solver will be able to determine the solution for any large amount of jobs, \( J \), that match the problem description of this paper. Of course, once the problem becomes too large to solve, the MILP model can no longer be used. Instead, the GA must be used in order to shorten computational time.

3.2. FJSP-LS Model

The mathematical model as a foundation for the two GAs discussed in this thesis is adopted from Defersha and Chen (2012a). It considers a flexible job shop problem with lot streaming (FJSP-LS). The FJSP-LS model contains a MILP model. This mathematical model can then be used to verify the validity of the GAs discussed in this thesis.

3.2.1. Problem Description

Consider a job shop consisting of \( M \) machines. These machines are grouped together based on similarities. For instance, all drilling machines will be placed beside one another. The machines are indexed by \( m = 1, 2, \ldots, M \). In this problem, the job shop may still be processing jobs from a previous schedule. Hence, not all machines are available at time or date zero. The machines that
are busy will be free at machine release date $D_m$. The number of jobs in the system are indexed by $j = 1, 2, \ldots, J$. In this model, the definition of job $J$ is slightly modified: it represents a batch $B_j$ of unique parts that must be processed instead of a single unique part as discussed in Section 3.1. The batch is split into sublots, where the total number of sublots is $S_j$. The sublots are indexed by $s = 1, 2, \ldots, S_j$. Each job can have more than one operation, which is indexed by $o = 1, 2, \ldots, O_j$, where $O_j$ is the maximum number of operations of a particular job $j$. The continuous variable $b_{s,j}$ represents the size of the sublot $s$ of job $j$. The processing time $T_{o,j,m}$ now concerns the operation $o$ of job $j$ when assigned to a particular machine $m$.

Let $S_{o,j,m,o',j'}$ be the setup time where $o'$ and $j'$ denote the previous operation for the previous job processed on the same machine $m$. Let $S^*_{o,j,m}$ be the setup time for the first operation $o$ for a job $j$ assigned to machine $m$. Here, the setup time is based on the previous operation. This model uses the precedence-variable based model. A sequence-position model was used in the rudimentary model. Differences between the two models are shown in Bayat Movahed (2014).

The setup time for an operation can be attached or detached. Attached means that the operation is non-anticipatory. For instance, if the setup requires the job to physically be at the machine before the setup can occur, the setup is attached (non-anticipatory). Detached means that the operation is anticipatory. For the same sublot, this allows for the setup time to overlap the previous operation’s processing time. Let $A_{o,j} = 1$ denote if a particular operation $o$ for a job $j$ is attached and let $A_{o,j} = 0$ denote if a particular operation $o$ for a job $j$ is detached. There may also be a lag time $L_{o,j}$ for a particular operation $o$ for a job $j$. The lag time occurs after the setup has been completed but before the operation has begun processing. For instance, a panel waiting to be painted may require its previous coat to dry even though the machine has already finished setting up. Lag time can occur for both attached and detached setups.
Let $R_m$ be the maximum number of runs for a particular machine $m$. The runs can be indexed by $r$ or $u = 1, 2, \ldots, R_m$. Each run $r$ is limited to being assigned to one sublot. The runs ultimately determine the sequence of operations to be performed on a particular machine $m$. For instance, the first run on machine $1$ ($m = 1$ and $r = 1$) can be assigned to process operation $1$ of job $5$ for sublot $1$. Since this layout is flexible, it is important to denote what machines are eligible to process certain operations. Let $P_{o,j,m} = 1$ to denote a machine $m$ is eligible to process operation $o$ of job $j$; otherwise let $P_{o,j,m} = 0$. Large positive number $\Omega$ is reused again in this model.

Aside from the input and indexes introduced above, the following are the remaining continuous variables used in this model. The makespan of the schedule $c_{max}$ is reused again in this model to signify the completion time of the last job in the model. New completion time variables $c_{o,s,j,m}$ and $\hat{c}_{r,m}$ are introduced to keep track of the completion times of each operation. Refer to Defersha and Chen (2012a) for a detailed explanation of the completion time variables.

The following are the binary integer variables used in this model. The binary integer variables are used to allow the model to know which job it is calculating. The binary integer variables are: $x_{r,m,o,s,j}$, $y_{r,m,o,j}$, and $z_{r,m}$. They are equal to $1$ if the runs, machines, jobs, sublots, and operations are assigned or linked together, otherwise equal to $0$. The binary integer variable $\gamma_{s,j}$ is used to indicate whether the sublot size is positive (equal to $1$) or nonpositive (equal to $0$). Refer to Defersha and Chen (2012a) for a detailed explanation of the binary integer variables.

The goal of the problem is to find the size of the sublots for each job, the assignment of the eligible machines for flexible jobs, the sequence of operations on each machine, the start and end time of each operation on each machine. The objective of the MILP problem is to minimize the makespan. Hence, the objective function for this optimization model is to minimize
Chapter 3. Mathematical Model

\[ Z = c_{\text{max}} \]  

subject to the constraints presented in the next section.

3.2.2. MILP Model

The MILP mathematical model presented below is adopted from Defersha and Chen (2012a). The model is constrained by machine capacity, precedence, logical relations among variables, and binary constraints. The model has been presented below for convenience of the reader as the proposed TAESGA improves upon the existing mathematical model. The definitions of the contraints written by Defersha and Chen (2012a) have also been provided so that the complete model is available to the reader.

Minimize:

\[ Z = c_{\text{max}} \]  \( (3.12) \)

Subject to:

The makespan in this model is defined by

\[ c_{\text{max}} \geq c_{o,s,j,m} ; \quad \forall (o, s, j, m) \]  \( (3.13) \)

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. This is enforced by

\[ \tilde{c}_{r,m} \geq c_{o,s,j,m} + \Omega \cdot x_{r,m,o,s,j} - \Omega ; \quad \forall (r, m, o, s, j) \]  \( (3.14) \)

and

\[ \tilde{c}_{r,m} \leq c_{o,s,j,m} - \Omega \cdot x_{r,m,o,s,j} + \Omega ; \quad \forall (r, m, o, s, j) \]  \( (3.15) \)

The staring time of the setup for the first run \( (r = 1) \) of machine \( m \) is given by \( \tilde{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S^*_{o,j,m} \) if the \( o^{th} \) operation of sublot \( s \) of job \( j \) is assigned to
Chapter 3. Mathematical Model

this first run. This starting time cannot be less than the release date of machine $D_m$. Hence,

$$\hat{c}_{1,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m} + \Omega \cdot x_{1,m,o,s,j} + \Omega \geq D_m ; \quad \forall (m, o, s, j) \quad (3.16)$$

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. This is constrained by

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

$$\forall (r, m, o, s, j, o', j') | (r > 1) \quad (3.17)$$

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 $r' - 1$ of machine $m'$ plus lag time $L_{o,j}$. The constraint

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

$$\forall (m, r', m', o, s, j) | \{(1, m) \neq (r', m') \} \land (o > 1) \} \quad (3.18)$$

is applied if first run of machine $m$ is assigned to operation $o$ of subplot $s$ of job $j$ and run $r'$ of machine $m'$ is assigned to operation $o - 1$ of this same subplot.

Similar to Equation 3.18, a constraint must be applied 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 $m$. Thus, we have

$$\hat{c}_{r,m} - b_{s,j} \cdot T_{o,j,m} - S_{o,j,m} - A_{o,j} - \Omega \cdot (y_{r-1,m,o',j'} + x_{r,m,o,s,j} + x_{r',m',o-1,s,j}) + 3\Omega \geq \hat{c}_{r',m'} + L_{o,j} ; \forall (r, m, r', m', o, s, j, o', j') | \{(r > 1) \land (o > 1) \land (r, m) \neq (r', m') \} \land (o, j) \neq (o', j') \} \quad (3.19)$$

For 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. This is enforced by

$$y_{r,m,o,j} \leq P_{o,m} ; \quad \forall (r, m, o, j) \quad (3.20)$$
The logical relation between the binary variable \( y_{r,m,o,j} \) and \( x_{r,m,o,s,j} \) is enforced by
\[
y_{r,m,o,j} = \sum_{s=1}^{S_j} x_{r,m,o,s,j}; \quad \forall (r, m, o, j) \quad (3.21)
\]

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. Therefore, we have
\[
\sum_{m=1}^{M} \sum_{r=1}^{R_m} x_{r,m,o,s,j} = \gamma_{s,j}; \quad \forall (o, s, j) \quad (3.22)
\]

However, if the size of this sublot is zero, it should not be assigned to any production run. Hence, we have
\[
b_{s,j} \leq B_j \cdot \gamma_{s,j}; \quad \forall (s, j) \quad (3.23)
\]

If the sublot size \( b_{s,j} = 0 \), the binary variable \( \gamma_{s,j} \) is forced to take the value 0. This is constrained by
\[
\gamma_{s,j} \leq b_{s,j}; \quad \forall (s, j) \quad (3.24)
\]

The sum of the sizes of the sublots of job \( j \) equals the batch size of this job. Thus,
\[
\sum_{s=1}^{S_j} b_{s,j} = B_j; \quad \forall (j) \quad (3.25)
\]

Each production run of a given machine can be assigned to at most one operation:
\[
\sum_{j=1}^{J} \sum_{s=1}^{S_j} \sum_{o=1}^{O_j} x_{r,m,o,s,j} = z_{r,m}; \quad \forall (r, m) \quad (3.26)
\]

and production run \( r + 1 \) can be assigned to an operation if and only if run \( r \) of that machine is already assigned:
\[
z_{r+1,m} \leq z_{r,m}; \quad \forall (r, m) \quad (3.27)
\]

The constraints given in Eqs. (3.28) and (3.29) are used 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. (3.18) and (3.19). If an operation $o$ of sublot $s$ 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 $m$. This is enforced by

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} ; \forall(r, r', m, o, o', s, j) \{(o' > o) \land (r' < r)\}. \quad (3.28)$$

Similarly to the above constraint, 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. Hence, we have

$$x_{r',m,o',s,j} \leq 1 - x_{r,m,o,s,j} ; \forall(r, r', m, o, o', s, j) \{(o' < o) \land (r' > r)\} \quad (3.29)$$

Equations 3.28 and 3.29 constraints are not redundant as the constraints in Equations 3.18 and 3.19 have already imposed these relationships. They are added to the model to improve the speed of the BB procedure.

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 by

$$x_{r,m,o,s,j}, y_{r,m,o,j}, \gamma_{s,j} \text{ and } z_{r,m} \text{ are binary} \quad (3.30)$$
Chapter 4

The Proposed Algorithm

4.1. Introduction

The two proposed GAs are introduced in this section. Terminology must be introduced for the reader unfamiliar with GAs. In this section, a system is defined as a group of machines with its own machine queue waiting to append or process any job in the machine queue once the machine is free. GAs require many chromosomes to carry the information of the optimization problem. The chromosomes together form the search space of the GA. The solution representation is encoded into each of the genes of the chromosome. For the purposes of scheduling, the genes will dictate the sequence of operations. The combination of these genes form the chromosome. The initial population represents all the chromosomes that must be generated before the GA begins its search. Each of the chromosomes can be decoded so that the makespan is calculated. The chromosome with the lowest makespan is considered to have the high fitness within the pool of chromosomes (search space). Operators are then used to manipulate the chromosomes with high fitness so that the search result can give a better solution through each iteration. Operators are applied with probabilities. These parameters must be fine tuned per problem in order to achieve the best results (Zhong and Huang, 2009).
As stated before in Section 2.2, GAs can struggle in the local search after the global search is complete. Majority of the scheduling literature face a problem where the search is stuck in a local optimum. The second proposed GA in this section seeks to combat that problem by laying a new framework explained in Section 4.3.

The classical job shop problem is NP hard. Hence, GA is typically employed to search for the best solution. Many GAs already exist as described in Chapter 2. There are two proposed algorithms to solve this model. The first algorithm, presented in Defersha and Chen (2012a), has been studied in scheduling for many years now. It is necessary to explore the Traditional Encoded Sequence Genetic Algorithm (TRADGA) in this thesis in order to better develop an understanding of how the (second) new algorithm works.

TRADGA performs a search of all possible solutions in order to find an optimal solution. The thorough search is made possible since each chromosome contains every operation of the job. With each gene carrying information of each operation, the genes can be operated on to give all possible solutions. Because this considers all possible solutions, the computation time is increased. Moreover, it is more likely for the GA to be stuck in a local optimum. When systems become more complex and larger in size, then the computation time must be considered before starting the simulation Banks et al. (2004). One way to combat the long computation time is to use a permutation based genetic algorithm. This means that the search no longer considers all the possible solutions, but an approximately optimal solution is reached.

The proposed TAESGA uses the TAES algorithm as the decoder. The only information added to the gene is the first job (and sublot if lot streaming is considered). Machines are omitted from the encoded sequence since the machine responsible for processing is determined through the decoder. The remainder
of the operations are omitted from the encoded sequence/solution representa-
tion/chromosome. The chromosome then becomes a dispatch sequence for which
the jobs enter into the queues of the machines (waiting to be processed when ma-
chine is free). Less genes are required to form the chromosome. Hence, smaller
chromosomes lead to a faster search to an acceptable solution. For a particular
problem, the search space is considerably smaller in comparison to the search
space in the TRADGA. This means that the TRADGA is more suited to tackle
small sized problems while the TAESGA is more suited to tackle large size prob-
lems. The Event-Scheduling/Time-Advance Genetic Algorithm (TAESGA) is
more scalable than TRADGA.

4.2. Traditional Encoded Sequence Genetic Al-
gorithm

The TRADGA searches for a solution to the NP-hard problem while encoding the
machine and operations as part of the solution representation. The GA elements
are elaborated upon in the sections below.

4.2.1. Solution Representation

The solution representation is encoded into a chromosome. The job number is
encoded into the gene. Then, the sequence can be determined by the sequence of
the genes. In addition, the TRADGA encodes the assigned machine and operation
number. This increases the chromosome size, while providing a more thorough
search at the expense of computation time. If lot streaming is introduced, the
chromosome can be augmented to include the sublot size and number of sublots
of each job. Figure 4.1 shows the sequence and assignments encoded into the
chromosome. Here, the sublot index is included in the gene to introduce the
reader later on how lot streaming is considered in an augmented chromosome. Each gene represents a sublot of a job for particular operation, assigned to a particular machine. All the genes form a sequence that the operations must be completed in. The locations \( l \) of each gene are shown in the first row of the chromosome. Here, the sequence of the genes is equal to the sequence of operations in the system. For instance, looking only at the genes that contain information relating to machine \( m = 4 \) at locations \( l = 5, 6, 17, 19 \), we see that the sequence for the machine is \((\text{job 2, sublot 2, operation 1}) \rightarrow (\text{job 3, sublot 2, operation 2}) \rightarrow (\text{job 1, sublot 1, operation 3}) \rightarrow (\text{job 1, sublot 3, operation 3})\).

![Figure 4.1: The RHS of a TRADGA chromosome considering lot streaming is adopted from Defersha and Chen (2012a)](image)

If the model does not consider lot streaming, the chromosome in Figure 4.1 would exclude the sublot index \( s \). If lot streaming is introduced, the sublot index is included as part of the gene. In Figure 4.1, the sublot index is part of the gene since it represents the right hand side (RHS) portion of an augmented chromosome that considers lot streaming. Figure 4.2 shows the full, augmented chromosome when lot streaming is considered. The chromosome is augmented from the original shown in Figure 4.1 to include the size of each sublot and the number of sublots per job. This is all encoded onto the left hand side of the chromosome. Evidently, the left hand side (LHS) of the chromosome shows that jobs 1 and 3 have 3 sublots whereas job 2 has only 2 sublots.

The genes in the LHS segment of this chromosome contain the percentage \( \alpha_{s,j} \) of which the job should be allocated to that particular sublot. It has a random value from the inclusive interval: [0 to 1]. The sublot size of each sublot
Chapter 4.

The Proposed Algorithm

Figure 4.2: The full TRADGA chromosome considering lot streaming is adopted from Defersha and Chen (2012a)

can be calculated using equation 4.1. Some sublots can have a size of zero. This allows the number sublots per job to vary.

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

(4.1)

\(\alpha_{s,j}\) takes a value from 0 to 1

The initial population is then formed from all the random chromosomes generated in the beginning of a problem. The values in the chromosome’s genes are randomly permutated.

4.2.2. Initial Population

The initial population is formed by the chromosomes. Each gene is formed by \((j,s,o,m)\) which is a quadruple. For a particular job \(j\) and sublot \(s\), each quadruple lies to the right of its previous quadruple \((j,s,o',m')\) where the previous operation is \(o'\) and previous assigned machine is \(m'\). The initial population cannot be formed until the operations for each job have been assigned a machine. Before the chromosome can be formed and encoded, the operations of each job must be assigned to a specific machine. Machine assignment rules must be developed for a flexible job shop problem (FJSP), with or without lot streaming. Note that this
The Proposed Algorithm

may not be the most optimal chromosome; however, the initial population must be formed (with randomly assigned machines to jobs) before the search can begin. There are two assignment rules for this particular algorithm. When generating the initial population, the assignment rules account for the processing times and workload of the machines. The workload of the machines is the total duration in which the machine is operating. The assignment rules will be introduced through a small example problem of 4 machines and 3 jobs. Each of these jobs can be processed by any of the 4 machines. The data is given in Table 4.1. In Table 4.1: Average setup time, processing time and machine release date. Data is adopted from Defersha and Chen (2010b)

<table>
<thead>
<tr>
<th>Job</th>
<th>Operation</th>
<th>Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>J O</td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>1</td>
<td>(10, 20)</td>
<td>(10, 50)</td>
</tr>
<tr>
<td>2</td>
<td>(10, 40)</td>
<td>(15, 60)</td>
</tr>
<tr>
<td>3</td>
<td>(15, 90)</td>
<td>(10, 70)</td>
</tr>
<tr>
<td>2</td>
<td>(25, 70)</td>
<td>(20, 60)</td>
</tr>
<tr>
<td>3</td>
<td>(10, 40)</td>
<td>(10, 80)</td>
</tr>
<tr>
<td>3</td>
<td>(15, 90)</td>
<td>(20, 50)</td>
</tr>
<tr>
<td>2</td>
<td>(10, 80)</td>
<td>(15, 60)</td>
</tr>
<tr>
<td>3</td>
<td>(15, 30)</td>
<td>(15, 50)</td>
</tr>
<tr>
<td>Machine Release Date</td>
<td>70</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.1, the average processing time, setup time, and machine release date are considered. The average processing time is used because sequence dependent setup times exist. Since the sequence has not been determined during this phase of the algorithm, the creation and initialization of the initial population considers the average setup time for each operation of a job.

The first assignment rule is called assignment-rule-1. Workload is calculated by summing all assigned processing times to the machine. Throughout each
assignment, the machine with the minimum workload is selected in an attempt to equally balance machine assignments for every flexible operation. An example of assignment-rule-1 is shown in Table 4.2.

Table 4.2: Initial assignments considering processing time (machine workload updates in bold). Table is adopted from Defersha and Chen (2010b)

<table>
<thead>
<tr>
<th>J</th>
<th>O</th>
<th>Initial Data</th>
<th>1st Assignment</th>
<th>2nd Assignment</th>
<th>...</th>
<th>Final Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>50 80 30 60</td>
<td>50 80 [30] 60</td>
<td>50 80 30 60</td>
<td>50 80 [30] 60</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>30 30 80 50</td>
<td>30 30 [110] 50</td>
<td>30 30 110 50</td>
<td>30 30 80 50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>50 70 40 60</td>
<td>50 70 [70] 60</td>
<td>80 70 70 60</td>
<td>50 70 40 [60]</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>60 40 50 80</td>
<td>60 40 [80] 80</td>
<td>60 40 80 80</td>
<td>60 [40] 50 80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>70 90 40 50</td>
<td>70 90 [70] 50</td>
<td>100 90 70 50</td>
<td>70 90 [40] 50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>30 20 10 50</td>
<td>30 20 [40] 50</td>
<td>60 20 40 50</td>
<td>30 20 10 50</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>40 40 80 60</td>
<td>40 40 [110] 60</td>
<td>70 40 110 60</td>
<td>40 [40] 80 60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20 90 20 70</td>
<td>20 90 [50] 70</td>
<td>50 90 50 70</td>
<td>20 90 20 70</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The order of the jobs and operations in the first two columns of Table 4.2 are randomly permutated when forming the initial population. Here the random order is job 3, then job 2, and then job 1. Using assignment-rule-1, the assignment of the machine used to process the job is determined through the minimum workload. Table 4.2 only accounts for processing time. Defersha and Chen’s paper also accounts for setup time and release date. This is shown in Table 4.3. Assignment-rule-1 is best used when there are no setup times and machines are available at time zero (when the machine release date is zero).

The second assignment rule is called assignment-rule-2. It does not require
Chapter 4. The Proposed Algorithm

Table 4.3: Initial assignments considering processing time, average setup time and machine release date (machine workload updates in bold). Table is adopted from Defersha and Chen (2010b)

<table>
<thead>
<tr>
<th>J</th>
<th>O</th>
<th>Initial Data</th>
<th>1st Assignment</th>
<th>2nd Assignment</th>
<th>···</th>
<th>Final Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>M4</td>
<td>M1</td>
<td>M3</td>
<td>M2</td>
<td>M4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>195</td>
<td>160</td>
<td>145</td>
<td>75</td>
<td>195</td>
</tr>
<tr>
<td>2</td>
<td>175</td>
<td>115</td>
<td>190</td>
<td>65</td>
<td>175</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>180</td>
<td>115</td>
<td>120</td>
<td>70</td>
<td>180</td>
<td>115</td>
</tr>
<tr>
<td>2</td>
<td>190</td>
<td>120</td>
<td>200</td>
<td>70</td>
<td>190</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>165</td>
<td>175</td>
<td>125</td>
<td>90</td>
<td>165</td>
<td>175</td>
</tr>
<tr>
<td>1</td>
<td>190</td>
<td>150</td>
<td>145</td>
<td>70</td>
<td>190</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>120</td>
<td>155</td>
<td>95</td>
<td>200</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>210</td>
<td>175</td>
<td>145</td>
<td>60</td>
<td>210</td>
<td>175</td>
</tr>
</tbody>
</table>

the random permutation of jobs initially unlike assignment-rule-1. Instead, the rule analyzes the entire workload matrix and searches for the global minimum workload. This is why the random permutation of jobs is not necessary. Both assignment rules are used when forming the initial population. A parameter called Assignment Rule Choice (ARC), where $0 \leq ARC \leq 1$, gives the percentage of chromosomes formed using assignment-rule-1. The remainder of the chromosomes are formed using assignment-rule-2.

Now that assignment rules are in place to determine which machine should be used for a particular flexible operation, the sequence of the genes in the chromosomes still needs to be determined before forming the initial population. There are three dispatching rules. Dispatching concerns the order in which the operation of the jobs are input into the system. The first rule is to randomly select a job (RSJ). Here, the jobs are randomly selected while respecting the precedence constraints of each operation. The second rule is called most work remaining (MWR). The average setup time and processing are totaled in calculating the
remaining work of each operation. While the precedence constraints are still respected, the operation with the most remaining work is selected to go ahead of other operations. Based on the solution given in Table 4.3, Table 4.4 calculates each operations' remaining work. Then, the operation sequence is determined using the MWR rule.

Table 4.4: Remaining work calculation and operations sequence for an initial solution based on the operation-machine assignment given in Table 4.3. Table is adopted from Defersha and Chen (2010b)

<table>
<thead>
<tr>
<th>Work Remaining (WR) Calculation</th>
<th>Operation Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Machine</td>
</tr>
<tr>
<td><strong>J</strong> 1 <strong>O</strong> 1 (10 + 30) + (10 + 40) + (10 + 70) = 170</td>
<td>M1</td>
</tr>
<tr>
<td>2 (10 + 40) + (10 + 70) = 130</td>
<td>M2</td>
</tr>
<tr>
<td>3 (10 + 70) = 80</td>
<td>M3</td>
</tr>
<tr>
<td>2 1 (20 + 40) + (10 + 80) + (15 + 40) = 205</td>
<td>M4</td>
</tr>
<tr>
<td>2 (10 + 80) + (15 + 40) = 145</td>
<td></td>
</tr>
<tr>
<td>3 (15 + 40) = 55</td>
<td></td>
</tr>
<tr>
<td>3 1 (15 + 60) + (15 + 30) = 120</td>
<td></td>
</tr>
<tr>
<td>2 (15 + 30) = 45</td>
<td></td>
</tr>
</tbody>
</table>

The last rule is called most number of operations remaining (MOR). Here, the sequence of the operations depends on which job has the most number of operations remaining. If an appropriate machine is free, the operation to be processed for a particular job will have the most number of operations remaining. The initial set of chromosomes can be formed with the aforementioned rules; thereby forming the initial population.

4.2.3. **Fitness Evaluation**

When the chromosomes are encoded, they need to be decoded in order to calculate the makespan. The makespan will be considered as the fitness of the chromosome. Chromosomes with better fitness will have lower makespan. The fitness evaluation
considers the sequence-dependent setup time, attached or detached setup, time lag between certain operations, and machine release dates. Since the dispatch sequence is already given in the genes of a chromosome, the decoding process is relatively straightforward in comparison to the TAESGA. The pseudocode for the fitness evaluation of the TRADGA provided below is adopted from Defersha and Chen (2010b).

**Step 1.** Set \( l = 1 \)

**Step 2.** Set the values of indices \( j, o \) and \( m \) as obtained from the gene at location \( l \) of the chromosome under consideration.

**Step 3.** Calculate the completion time \( c_{o,j,m} \)

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

**Step 4.** If \( l \) is less than the total number of operations, increase its value by 1 and go to Step 2; otherwise go to Step 5.
Step 5. Calculate the makespan of the schedule as $c_{max} = \max\{c_{o,j,m}; \forall (o, j, m)\}$ and set the fitness of the individual under consideration to $c_{max}$.

4.2.4. Selection Operator

The selection operator selects which chromosomes should be reproduced for the next iteration of the GA search. A $k$ amount of individual chromosomes are selected randomly. The chromosome with the best fitness (lowest makespan) wins. This method is called a $k$-way tournament selection. This method ensures that chromosomes with higher fitness have a higher probability of being selected. The $k$ can be adjusted. A higher $k$ results in a higher chance of repeated chromosomes with high fitness being used to fill the next generation of chromosomes/individuals. A lower $k$ results in a more diverse range of chromosomes that can be used to fill up the next generation pool.

This process is continued until the newly selected chromosomes forms the next generation of individuals such that number of individuals is equal to the population size. Selection operator occurs first, before crossover and mutation are applied. Once the pool is fully formed, individual chromosomes are randomly paired together as parents in order to form the new child chromosome. This is how evolution occurs in a GA.

4.2.5. Crossover Operators

The crossover operators used in this GA are adopted from Defersha and Chen (2012a):

- Single Point Crossover-1 (SPC-1)
- Single Point Crossover-2 (SPC-2)
- Operation-to-Machine Assignment Crossover (OMAC)
• Job Level Operations Sequence Crossover (JLOSC)

• Sublot Level Operations Sequence Crossover (SLOSC)

The operators are applied with a probability $\rho$ in order to prevent random and sudden changes to the parent genes when forming the child gene. This is necessary or else the offspring chromosome may result in having genetic makeup worse than their parent (Defersha and Chen, 2012b).

For SPC-1 and SPC-2, an arbitrary crossover point is chosen on the LHS of the chromosome. They focus on dealing with the lot streaming portion of the chromosome. For SPC-1, the left side of the chosen point is exchanged with the other parent chromosome. For SPC-2, the right side of the chosen point is exchanged with the other parent chromosome. SPC-1 and SPC-2 are operators applied with probabilities $\rho_1$ and $\rho_2$ respectively. These operators are shown visually in Figure 4.3.

![Figure 4.3](image-url)

(a) SPC-1  
(b) SPC-2

Figure 4.3: SPC-1 and SPC-2 adopted from Defersha and Chen (2012a)

OMAC, JLOSC, and SLOSC focus on dealing with the RHS portion of the
chromosome. There are two types of operators used in the RHS of the chromosome: assignment and sequencing. Assignment deals with changing the assignment of a machine in a particular location. This type of operator creates two child chromosomes. Genes from the two parents are selected arbitrarily. One of the subset of operations is changed to take on the assignment from the second parent. The remaining sequence of the chromosome is left unmodified. OMAC is the assignment operator used in this algorithm. These steps are shown visually in detail in Figure 4.4. OMAC is applied with probability $\rho_3$. The OMAC steps are listed below in detail.

**Step 1.** Arbitrarily select operations from parent 1

**Step 2.** Keep the remaining unmodified sequence of operations from parent 1 and copy onto the child chromosome

**Step 3.** Assignment properties from parent 2 are copied onto the child chromosome

**Step 4.** Create child 2 by beginning to select arbitrary operations from parent 2 and continue the above process

The crossover sequencing operators change the sequence of operations in the parent chromosomes, while leaving the assignment of the machines in a gene unmodified. JLOSC and SLOSC are both sequencing operators. They are applied with probabilities $\rho_4$ and $\rho_5$ respectively. Sequencing operators create only one child from two parent chromosomes. These steps are shown visually in detail in Figure 4.5. The JLOSC steps are listed below in detail.

- Choose an arbitrary operation from parent 1

- For the chosen operation, keep the remaining unmodified operations of all the sublots of the job and copy onto the child chromosome
For the remaining jobs, retain and copy the same sequence from the second parent onto the child.

SLOSC is similar to JLOSC. Instead of keeping the unmodified operations of all the sublots, only the information from the sublot of the selected operation is kept and copied to the child.

The precedence constraints are always respected in the sequencing operators. For a given job $j$ and sublot $s$, the gene is always located after the gene with its previous operation already sequenced. When evolving a new population with this rule, new child chromosomes will not violate the precedence constraints.

4.2.6. Mutation Operator

Crossover operators do not change the genetics of the existing chromosomes. They only alter the assignment and sequencing of the chromosome. Mutation operators are able to introduce new genetic material into the pool by altering the information contained in one of the genes. Mutation operators are applied to
child chromosomes after the crossover operators have been performed. Assignment and sequence operators also exist amongst the mutation operators. The mutation operators used in this GA are adopted from Defersha and Chen (2012a):

- Sublot Step Mutation (SStM)
- Sublot Swap Mutation (SSwM)
- Sublot Size degenerator (SSD)
- Random Operation Assignment Mutation (ROAM)
- Intelligent Operations Assignment Mutation (IOAM)
- Operations Sequence Shift Mutation (OSSM)

They are applied with small probability \( \sigma \) with the exception of SSD. The SSD operator is non-probabilistic.

Assignment mutation operators change the assignment of the machine for the flexible machine without changing the sequencing of the chromosome. SStM,
SSwM, and SSD are mutation operators that modifies the LHS of the chromosome. SSTM and SSwM are applied with small probability $\sigma_1$ and $\sigma_2$. SSD is a nonprobabilistic operator. SSTM steps the value of the gene up or down by $\theta$ when the operator is applied to a gene. When stepping the value up, in order to ensure the value of $\alpha_{s,j}$ does not exceed 1, the value is calculated as $\alpha_{s,j} = \min\{1, \alpha_{s,j} + \theta\}$. When stepping the value down, in order to ensure the value of $\alpha_{s,j}$ does not become negative, the value is calculated as $\alpha_{s,j} = \max\{0, \alpha_{s,j} - \theta\}$.

The value of $\theta$ is determined as $\theta = \theta_{\text{max}} \times \text{rand}()$, where $\theta_{\text{max}}$ is a predetermined parameter with a value that lies in $[0,1]$ and $\text{rand}()$ is a number generated randomly that lies in $[0,1]$. For a particular job $j$, SSwM swaps the values between the two genes $\alpha_{s,j}$ and $\alpha_{s',j}$. SSD is used to set the sublot size $\alpha_{s,j} = 0$ if the sublot size percentage $\alpha_{s,j}/\sum_{s=1}^{S} \alpha_{s,j}$ is less than a particular degeneration limit $d$. This is required since extremely small sublot sizes may not be realistic due to the nature of the setup in a plant. The degeneration limit is set close to 0.

ROAM, IOAM, and OSSM are concerned with modifying the RHS-segment of a child chromosome. They are applied with small probability $\sigma_3$, $\sigma_4$, $\sigma_4$ respectively. For a flexible operation, ROAM takes the assignment of a machine in a particular gene and changes its assignment to another qualified machine. IOAM finds an operation assigned to a machine that has maximum workload and attempts to assign it to a machine with minimum workload, if that particular machine is qualified. OSSM takes a gene and changes its location within the RHS-segment of the chromosome, while respecting the precedence constraints.

4.3. Event-Scheduling/Time-Advance Genetic Algorithm

While TRADGA encodes the machine and operations (in addition to the jobs and sublots) as part of the solution representation, TAESGA encodes only the
jobs (and sublots) in each of the genes of the chromosome. The chromosome still remains a dispatch sequence. However, the chromosome in TAESGA is significantly smaller and only contains the first operation of each job. Jobs are waiting to be inputted into the system once there is any appropriate machine to process the first operation of each job. The priority of the jobs are determined in order based on the location of the chromosome. Location \( l = 1 \) gets the first priority, while location \( l = 2 \) gets the second priority. This is continued to the end of the gene. The decoder is then responsible for determining the entire sequence of the system. Then, the makespan/fitness can be calculated.

Due to the nature of this procedure, this eliminates the need to assign appropriate machines to flexible jobs when generating the initial population. The release date of machines can simply be accounted for. The operation will go to the machine that has the smallest setup and processing time combined, if and only if the machine is free. The sequence of operations are also eliminated from the chromosome because the solution encoding is only concerned with the first operation of the job. Other operations cannot be considered until the first operation is inputted into the system. The overall steps of the TAESGA is shown in Figure 4.6. The Job Release Queue (JRQ) is a set of alternative job dispatching sequences, to be iteratively improved and supplied by the metaheuristic algorithm. It is the solution representation or pool of chromosomes of the TAESGA. It is then followed by a decoder and genetic operators. The GA elements are elaborated upon in the sections below.

### 4.3.1. Solution Representation

This GA uses a time-advanced discrete-event simulation as its decoder. Hence, the solution encoding only needs to contain the dispatch sequence in which the jobs should be released into the system. Without lot streaming, the solution representation would simply have the jobs in the genes of the chromosome. This
would be the dispatch sequence and order in which the first operation of each job should be released into the system. With lot streaming, the chromosome is augmented to include a LHS and RHS. The genes only contain the sublot sizes in the LHS and the job, sublot \((s,j)\) in the RHS of the chromosome.

The solution representation used for this GA has already been developed for a flowshop model in Defersha and Chen (2012b). The nature of the flowshop model and this GA coincidentally use the same solution representation. Recall that flowshop is only a special scenario of a jobshop. In a flowshop, all jobs are processed using the same order of machines. Hence, the flow is always going in a one-way direction. For example, all jobs will always go to machine 1, machine 2, and lastly machine 3. It becomes obvious why encoding the operations into the solution representation for a flowshop is not needed; rather, it is redundant. Since the TAESGA does not need the operations encoded into the solution representation, it is evident why the flowshop GA solution representation can be reused for the TAESGA. Figure 4.7 shows the solution representation used to solve this GA.

The sublot size can be calculated similarly using Eq. 4.1.
### 4.3.2. Initial Population

Since the decoder handles the assignment of the machines for flexible jobs, there is no need to encode the machines into the solution representation. The discrete-event system simulation that occurs during the decoding process determines which machine is free to process the first operation of a particular job. The initial population can be formed by permutating random numbers for the genes of the chromosomes.

### 4.3.3. Fitness Evaluation

This new fitness evaluation is similar to the fitness evaluation in the old algorithm. Since eligible machines are known, any free machine with the lowest processing time will be selected to process an operation. The end time of the operation is calculated as \( c_{j,o} \). To get the starting time of an operation, subtract its processing time from its \( c_{j,o} \). Knowing the start and end time of an operation, we can develop a schedule of when a machine is being used (since the machine is already preassigned to the operation in the given data).

This can be done by using discrete-event system simulation. Once the chromosome is formed from encoding (solution representation), the dispatch sequence can be obtained from the genes. The dispatch sequence is read from left to right or location \( l = 1 \) to \( l = L \), where \( L \) signifies the end of the chromosome. The sequence of jobs to be dispatched is simulated and the machines responsible for

---

<table>
<thead>
<tr>
<th>LHS-Segment</th>
<th>RHS-Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j=1 )</td>
<td>( 1,1 ) ( 2,1 ) ( 3,1 ) ( 4,1 ) ( 5,1 ) ( 6,1 ) ( 7,1 ) ( 8,1 ) ( 9,1 ) ( 10,1 )</td>
</tr>
<tr>
<td>( \alpha_{1,1} ) ( \alpha_{1,2} ) ( \alpha_{1,3} ) ( \alpha_{1,4} ) ( \alpha_{1,5} ) ( \alpha_{1,6} ) ( \alpha_{1,7} ) ( \alpha_{1,8} ) ( \alpha_{1,9} ) ( \alpha_{1,10} )</td>
<td></td>
</tr>
<tr>
<td>( (1,1) ) ( (1,2) ) ( (2,1) ) ( (2,2) ) ( (2,3) ) ( (3,1) ) ( (3,2) ) ( (3,3) ) ( (1,2) ) ( (1,3) )</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.7: Solution representation for TAESGA. Figure is adopted from Defersha and Chen (2012b)
processing the jobs are checked to see whether or not they are free to process these jobs. A clock time is used to keep track of the simulation.

Notice in Figure 4.6, a job in queue in front of a particular machine (resource) \( m \) is identified as \((o, j, t_{o,j,m})\) which represents operation \( o \) of job \( j \) and its future completion time \( t_{o,j,m} \). The corresponding future job completion event is listed in a calendar as \((o, j, m, t_{o,j,m})\) in which all other job completion events are listed in chronological order. The schedule generator (running in a simulated time) provides schedules and performance measure for each potential job-release-sequence. The decoder begins by initializing the system: setting the clock time to zero \((t_{\text{current}} = 0)\) and obtaining the JRQ from a chromosome in the solution population. The decoder then follows the steps listed below.

**Step 1.** Is there an Operation-Completion Event \((o - j - m - t_{o,j,m})\) at the top of the event list to occur at the current time (i.e. \( t_{o,j,m} = t_{\text{current}} \))? If yes go to Step 2; Otherwise go to Step 3

**Step 2.** Remove Event \((o - j - m - t_{o,j,m})\) from the top of the event list of the calendar.
Remove Operation \((o - j - t_{o,j,m})\) from the front of job queue of machine \( m \).

If operation \( o \) of job \( j \) is not the last operation, assign operation \( o + 1 \) of this job to the alternative machine \( m' \) that can complete this operation at earliest time. Set the completion time to \( t_{o+1,j,m'} \).

Append Operation \([(o+1) - j - t_{o+1,j,m'})\) at the tail of operations queue of machine \( m \) and insert Event \([(o+1) - j - m - t_{o+1,j,m'})\) in the calendar. Go to Step 1

**Step 3.** Are there machine(s) that are currently (at \( t = t_{\text{current}} \)) idle. If yes, go to Step 4. Otherwise, go to Step 7
Step 4. If there is a job in the JRQ, go to Step 5. Otherwise, go to Step 7

Step 5. Starting from the front the JRQ, look for a job whose first operation can be assigned to one of these idling machines. If there is such a job $j$, remove this job from JRQ and go to Step 6. Otherwise, go to Step 7

Step 6. Among the idling machines that can process the first operation of job $j$, find the one that can finish this operation at the earliest time. Set this machine to machine $m'$ and the completion time is $t_{1,j',m''}$. Append operation $(1 - j - t_{1,j',m''})$ to the empty queue of machine $m$. Insert event $(1 - j - m - t_{1,j',m''})$ in the calendar list. Go to Step 3

Step 7. If the calendar list is not yet empty: advance the time to the time of the event at the top of the event list (set this event to $o - j - m - t_{o,j,m}$): $t_{\text{current}} = t_{o,j,m}$. Then, go to Step 2. Otherwise, go to Step 8

Step 8. The JRQ is completed and the current time is the makespan ($\text{Makespan} = t_{\text{current}}$). Start again from initializing the system until all JRQ are evaluated.

4.3.4. Selection Operator

The operators used in this GA can also be adopted from Defersha and Chen (2012b) since they share the same solution encoding. The selection operator used in this GA is the same as the TRADGA discussed. Chromosomes are put into a $k$-way tournament. Depending on the value of $k$, there can be a variation of chromosomes selected to form the new population pool. Same as before, a large value of $k$ will lead to lower variation of chromosomes in the new population pool. A small value of $k$ will lead to a higher variation of chromosomes in the new population pool. The chromosome with the highest fitness (lowest makespan)
will be selected out of the $k$ members in the tournament. The procedure continues until there are enough chromosomes to fill up the population pool.

### 4.3.5. Crossover Operator

The crossovers used in this GA are adopted from Defersha and Chen (2012b):

- LHS-Segment Crossover-1 (LHSSX-1)
- LHS-Segment Crossover-2 (LHSSX-2)
- RHS-Segment Swap Crossover (RHSSSX)
- Single Point Order Crossover-1 (SPOX-1)
- Single Point Order Crossover-2 (SPOX-2)
- Two-Point Order Crossover (TPOX)
- Single Point Similar Sublot Order Crossover (SPSSOX)
- Single Point Similar Block Order Crossover (SPSBOX)
- Two-Point Similar Sublot Order Crossover (TPSSOX)
- Two-Point Similar Block Order Crossover (TPSBOX)

The above operators are applied with probabilities $\rho_1, \rho_2, \ldots, \rho_{10}$ respectively.

LHSSX-1 and LHSSX-2 are similar to one another. They are applied with probabilities $\rho_1$ and $\rho_2$ respectively. They focus on modifying the lot streaming (LHS) portion of the chromosome. LHSSX-1 takes an arbitrary point in the LHS of the chromosome and exchanges the portion to the left of the arbitrarily selected point with the other chromosome. LHSSX-2 does the same but takes the portion to the right of the arbitrarily selected point. Figure 4.8 illustrates LHSSX-1 and LHSSX-2. RHSSSX simply takes two chromosomes and exchanges
## The Proposed Algorithm

### Chapter 4

#### Figure 4.8: Illustration of the LHS-Segment crossover operators assuming a crossover point between $\alpha_{1,3}$ and $\alpha_{2,3}$. Only one crossover operators will be applied on a given pair of parents. Figure is adopted from Defersha and Chen (2012b)

<table>
<thead>
<tr>
<th>LHS-Segment</th>
<th>LHS-Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j=1$</td>
<td>$j=1$</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$l=1$</td>
<td>$l=1$</td>
</tr>
<tr>
<td>$\alpha_{1,2}$</td>
<td>$\alpha_{1,2}$</td>
</tr>
<tr>
<td>$\alpha_{1,3}$</td>
<td>$\alpha_{1,3}$</td>
</tr>
<tr>
<td>$\alpha_{1,4}$</td>
<td>$\alpha_{1,4}$</td>
</tr>
<tr>
<td>$\alpha_{2,4}$</td>
<td>$\alpha_{2,4}$</td>
</tr>
</tbody>
</table>

(a) LHSSX-1

(b) LHSSX-2

SPOX-1 and SPOX-2 are similar. Both operators modify the RHS of the chromosome. Both require two parents and generate two child chromosomes. SPOX-1 takes the genes to the left of an arbitrarily selected point in the RHS-segment of the direct parent chromosome and copies them over to the child chromosome. The remaining missing genes are filled in using genetic material from the other parent. The sequence from the other parent is conserved when filling in the missing genes. Figure 4.9 illustrates SPOX-1. SPOX-2 operates using the same steps except it takes the genes to the right of an arbitrarily selected point in the RHS-segment of the direct parent chromosome.

TPOX is slightly similar to SPOX-1 and SPOX-2. Instead of using a single arbitrary point, two arbitrary points (boundary points) are selected. The sublots from the direct parent outside of these two boundary points are kept in the child chromosome, while the missing points in between the boundary points are filled in from the other parent. The sequence from the other parent is still conserved when the empty genes are filled in. TPOX uses two parent chromosomes to create two child chromosomes.

SPSSOX takes two parent chromosomes to form two child chromosomes.
Chapter 4.

The Proposed Algorithm

Figure 4.9: Illustration of SPOX-1: (a) Sublots to the left of the crossover point are inherited from the direct parent. (b) Then, the missing sublots are copied in their relative order of the other parent. Figure is adopted from Defersha and Chen (2012b)

The operator searches for genes that are identical in the same locations of the RHS-segment of the two parent chromosomes. These genes are then copied over to the two child chromosomes. An arbitrary point is selected and the genes to the left of this point are copied over to the child one and child two from parent one and parent two respectively. The remaining empty genes are filled with the missing genes from the other parent. The sequence from the other parent is still conserved when the empty genes are filled in. SPSSOX is illustrated in Figure 4.10.

SPSBOX is similar to SPSSOX. Instead of selecting the genes that are identical between the two parents in the same location, the genes must be grouped beside one another. The block of genes must consist of two or more genes before they can be copied over to the child chromosomes. In Figure 4.10 sublots (2,4) and (3,3) form a block because the two genes are beside one another.

TPSSOX and TPSBOX is similar to SPSSOX and SPSBOX respectively. For TPSSOX and TPSBOX, instead of copying genes to the left of the arbitrarily
Chapter 4. The Proposed Algorithm

Figure 4.10: Illustration of SPSSOX: (a) First, the common sublots in both parents are copied over to the childred; (b) Then, sublots to the left of the crossover point are inherited from the direct parent. (c) Finally, the missing sublots are copied in their relative order of the other parent. Figure is adopted from Defersha and Chen (2012b)
selected point, the genes to the right of the arbitrarily selected point are copied.

4.3.6. **Mutation Operator**

Mutation operators are applied to child chromosomes after the crossover operators have been applied. The mutation operators used in this GA are:

1. SStM
2. SSwM
3. SSD
4. Sublot Order Shift Mutator (SOShM)
5. Sublot Order Swap Mutator (SOSwM)

The mutation operators used on the LHS-segment (SStM, SSwM, and SSD) of the child chromosomes are identical to the mutation operators discussed in the previous GA. SStM is applied with small probability $\sigma_1$. SSwM is applied with small probability $\sigma_2$. SSD is a nonprobabilistic operator.

SOShM and SOSwM are mutation operators with small probabilities $\sigma_3$ and $\sigma_4$ respectively. These operators affect the RHS-segment of the child chromosomes. SOShM takes a gene from the RHS-segment and moves it to a new arbitrary location within the RHS-segment. SOSwM takes two genes in the RHS-segment and exchanges their locations. Precedent constraints are still respected when this mutation operator is applied.
Chapter 5

Numerical Examples

5.1. Initial Population Generation Performance

To test the performance of the two GAs, an initial population of 1500 chromosomes is generated for both the TRADGA and TAESGA. The fitness or makespan is analyzed to see how frequent good makespans are generated with little variation amongst the chromosomes. Evident in Figure 5.1, the TAESGA is able to outperform the TRADGA when generating the initial population. The makespans generated in the initial population of the TRADGA are approximately double in size in comparison to the TAESGA. There is a huge variation in the makespans of the TRADGA, which is undesired. Indicated in the graph, the TAESGA experiences a higher frequency of similar makespans. It can be concluded that the TAESGA outperforms the TRADGA when generating the initial population.

To explore the root cause of this discrepancy in performance, it can be hypothesized one of the potential causes could be due to the different solution encoding techniques. Recall in the TRADGA, the entire sequence is encoded into the chromosome, whereas the only the initial operation of each job is encoded into the chromosome for the TAESGA. This means that extremely inefficient chromosomes (where machines could potentially be idling for unnecessary periods of
Figure 5.1: The distribution of makespan for 1500 randomly generated solution in TAESGA vs TRADGA

time) could be generated in the initial population. This would ultimately lead to a poor makespan.

It can be hypothesized that using the TAES algorithm as a decoder is much more efficient in shortening the makespan because whenever machines are free, they are open to process the machine queue (whether the machine queue contains another operation waiting from a previously completed operation or an initial operation of a new job newly dispatched into the system). We can verify this hypothesis by performing a hand calculation on a randomly permuted chromosome that would exist in the TRADGA and TAESGA. Consider an example with 3 machines, 4 jobs, and 3 operations. Each operation can be performed on any of the three machines. Potential TRADGA and TAESGA chromosomes are generated. These chromosomes are shown in Figure 5.2 and 5.3.
Chapter 5. Numerical Examples

Figure 5.2: The RHS-Segment of a randomly permutated chromosome that may exist in the TRADGA. The first row denotes the location of the gene. The second row denotes the job, operation, and machine encoded into the gene.

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>L7</th>
<th>L8</th>
<th>L9</th>
<th>L10</th>
<th>L11</th>
<th>L12</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Gene Location: gene location
Job, Operation, Machine: job

Before finding the makespan of the chromosome for the TRADGA, it is important to find the sequence at which the jobs can be completed for each machine. The order in which the operations are completed is shown in Figure 5.4. Analyzing gene locations 1-4, it can be shown that there is a long queue for machine 2 and machine 1 is still in idle after 4 genes have been inputted into the system.

Figure 5.3: The RHS-Segment of a randomly permutated chromosome that may exist in the TAESGA. The first row denotes the location of the gene. The second row denotes the job encoded into the gene.

<table>
<thead>
<tr>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Gene Location: gene location
Job: job

<table>
<thead>
<tr>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,3</td>
<td>3,1</td>
<td>2,1</td>
</tr>
<tr>
<td>3,3</td>
<td>1,1</td>
<td>L2</td>
</tr>
<tr>
<td>2,3</td>
<td>2,2</td>
<td>L3</td>
</tr>
<tr>
<td>1,3</td>
<td>4,2</td>
<td>L4</td>
</tr>
<tr>
<td>1,3</td>
<td></td>
<td>L1</td>
</tr>
</tbody>
</table>

Figure 5.4: Finding the order in which the operations are completed based on the TRADGA chromosome in Figure 5.2.
It can be concluded that within the completion of these first few operations, machine 1 is left idling. Clearly, this is an inefficiency that would exist only in the TRADGA because the TAESGA uses a TAES algorithm where machines are only allowed to be idle if there is no queue. The TRADGA does not worry about the queue; rather, it generates the makespan using brute force and trial and error to see if the sequence is able to compute a small makespan.

\[
\begin{array}{ccc}
\text{M1} & \text{M2} & \text{M3} \\
2,1 & 3,1 & 1,1 \\
2,2 & 3,2 & 1,2 \\
2,3 & 3,3 & 1,3 \\
4,1 & 4,2 & 4,3 \\
\end{array}
\]

Figure 5.5: Decoding the TAESGA chromosome based on the TAESGA chromosome in Figure 5.3

Using the TAES algorithm to decode the TAESGA chromosome, it is seen that the machine is no longer left to idle as shown in Figure 5.5. The TAESGA forces all the machines to be used in this case, which allows for a more efficient distribution of operations when aiming for a lower makespan.

5.2. Problem Size and Genetic Algorithm Computational Performance

TRADGA performs better for smaller sized problems, whereas TAESGA performs better for larger sized problems. This is evident in Figures 5.6 and 5.7. The TAESGA struggles to keep up with the TRADGA for the smaller problems. This is due to the fact that TRADGA is searching for all possible solutions. Specifically, the chromosomes account for all the possible solutions. Hence, the
Figure 5.6: Computational performance of small problems
Figure 5.7: Computational performance of large problems

Chapter 5. Numerical Examples
search space of the TRADGA is broadened. Having all the possible solutions considered allows for the TRADGA to reach a better solution than the TAESGA. However, this can pose a problem when problems are increased in size.

As stated before, GAs struggle to perform well in the local search phase. This is further illustrated in the larger problems sets. When the problem gets larger, TAESGA performs better than TRADGA since the search space is smaller in the TAESGA compared to the TRADGA. It is easy for the GA to get stuck in a local minima. Therefore, in a large problem, it could take infinitely long for the TRADGA to reach a better solution than the TAESGA.

![Figure 5.8: Computational performance of medium problems](image)

In the medium sized problems (shown in Figure 5.8), the TRADGA is
unable to converge as quick as the TAESGA. However, the final solution quality is on par, if not better than the TAESGA for medium sized problems. Moreover, the solution quality of the TRADGA is only slightly better than the TRADGA by a small factor. It can be stated that if more computational time was available, the TRADGA would be able to converge at a better solution than the TAESGA.

If speed is a factor in simulating these problems, the TAESGA would be a better choice. More importantly, TAESGA demonstrates its ability to converge quickly within the first 20 minutes of running the simulation. Its application is further discussed in 6.1. If solution quality needs to be extremely high, then TRADGA would be a better choice for smaller to medium sized problems. The computational time in the time axis of the graphs shows that the proposed TAESGA is more computationally efficient than the TRADGA. The simulations were run on a computer with the following specifications: Intel Xeon E3-1225 v3 @3.20Ghz and DDR3 16GB RAM.
Chapter 6

Research Summary

6.1. Discussion and Conclusion

Most of the time in industry, solving scheduling problems needs to be done quickly. An hour may be reasonable, but developing a GA that can have a relatively good solution within seconds for the controller of the manufacturing operations would be advantageous in today’s time-based manufacturing competition. For smaller to medium sized problems, the solution quality delivered by the TRADGA is only slightly better than the TAESGA. However, if more time was available (i.e. an extra 12 hours), TRADGA would be able to deliver a better solution at a cost of time. Once an optimized schedule is in place, the schedule generator is to run in real time to become a supervisory controller and monitor the schedule and react for unexpected events.

More importantly, the scheduling problems that exist today in industry are large sized problems. Small problems are used as an exercise, while large sized problems become an issue for the TRADGA to tackle. Hence, the proposed TAESGA is introduced so that an acceptable solution for the large sized problems can be immediately acquired and adjustments can be made on-the-fly to improve makespan in the real-world.
In addition to the TRADGA developed for this thesis, the design of the code allows for metrics to be added. The TAESGA does not require the type of machine layout adopted in the system as an input. Recall in Figure 4.6, a resource \( m \) in the schedule generator can be defined either as a unit processing (e.g. lathe, milling), a batch processing (e.g. heat treatment) or a continuous processing (e.g. painting line) or a material handling resource or even as buffer storage. As such the algorithm can be expanded to handle a wide range of discreet and semi-discrete manufacturing systems with greater operational complexity. This allows for the code to scale in both problem size and programmable features (metrics).

6.2. Future Research

Majority of current research on scheduling problems cover single-objective optimization. Usually, either makespan or cost are minimized. There is limited research on multi-objective scheduling optimization, (i.e. Gkan May (2015)). Multi-objective optimization in scheduling is critical and more practical for companies as multiple factors (such as earliness, lateness, and tardiness) are needed to take into account during product launch. This is today’s requirement for remaining relevant in the current global competition. Moreover, certain metrics (such as work-in-process inventory, machine utilization, and work-load balancing) ultimately affecting the schedule should also be factored into the heuristic rather than being overlooked. The research presented in this thesis lays a foundation to be built upon, since the heuristic is generalized enough to solve most variations of scheduling problems, while still being efficient. Addition of these metrics as objective functions is possible due to the flexibility of the framework of the code. Hence, future research lies in multi-objective optimization for scheduling problems. To achieve this, the TAESGA must have database modules for integration with IT-infrastructure. The method will also incorporate modelling,
parallel computation, several metaheuristics, and artificial intelligence (AI) for self-parameter tuning.

The computational performance and efficiency has been stressed upon throughout this thesis. Speed is valued because speed allows industries to take this tool and implement them and receive immediate results. The immediate results allow for industries to constantly update the schedule when setbacks arise. Moreover, industries can use the tool to immediately plan the schedule for new product introduction. This addresses the issue dynamic plant layouts face. The constantly changing environment can be rescheduled with an algorithm that is able to provide immediate results. Hence, updates can be changed on-the-fly in real-time. The assumptions of a static environment can still hold for the algorithm; but due to the fast convergence while still providing good solution quality, machine failures, urgent job arrival, order cancellation, due date changes, staff absentee can be addressed with this GA tool. Future research can either turn this GA into a tool that fully integrates with existing manufacturing plants, making the tool either user friend and/or capable of fully controlling/supervising and updating the system automatically whenever changes/setbacks arise.

In the long term, fully integrating a system that is able to fully control the schedule by quickly adapting to the dynamic environment is where the direction of future research should go. Integration of more variables that inherently affect the schedule should also be included. For instance, operator/personnel scheduling, inventory planning, and preventative maintenance scheduling are all factors that can be implemented into the existing model. This will thereby expand the GA model into not only a scheduling tool, but a tool that manages an entire manufacturing plant.
Bibliography


Bibliography


Bibliography


