A NOVEL ALGORITHM FOR SOLVING THE MULTI-OBJECTIVE ASSEMBLY LINE BALANCING PROBLEM

A Thesis
Submitted to the Faculty of Graduate Studies and Research
In Partial Fulfillment of the Requirements
For the Degree of

Master of Applied Science
in
Industrial Systems Engineering
University of Regina

by
Sayed Kaes Maruf Hossain
Regina, Saskatchewan
December 2016

© 2016: S. K. M. Hossain
Sayed Kaes Maruf Hossain, candidate for the degree of Master of Applied Science in Industrial Systems Engineering, has presented a thesis titled, *A Novel Algorithm for Solving the Multi-objective Assembly Line Balancing Problem*, in an oral examination held on September 2, 2016. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

- **External Examiner:** Dr. Malek Mouhoub, Department of Computer Science
- **Supervisor:** Dr. Mohamed Ismail, Industrial Systems Engineering
- **Committee Member:** *Dr. Ezeddin Shirif, Petroleum Systems Engineering* (not present at defense)
- **Committee Member:** Dr. Amr Henni, Industrial Systems Engineering
- **Chair of Defense:** Dr. Irfan Al-Anbagi, Electronic Systems Engineering

*Not present at defense*
Abstract

The assembly line balancing (ALB) problems address the issue of assigning work elements and workers into different workstations to optimize a predetermined set of objective parameters such as the number of workstations, the cycle time and the production efficiency. Most of these combinatorial optimization problems possess NP-hard computational complexity. This rationalizes the growing interest of researchers to use metaheuristics to solve ALB problems. The problems studied in literature over the years are often limited by simplifying assumptions, which brings in a challenge to the models developed in academia while applying to real life problems. In this thesis, we have intensively reviewed the characteristics of different types of ALB problems and existing solution approaches. Based on that, a general structure of the ALB model is developed which can be extended to any ALB problems with a reasonable effort. In traditional evolutionary optimization algorithms, encoding and decoding of solutions appear as an integral part. We eliminated the computational efforts for encoding and decoding of the solutions, by introducing a unique representation technique using an extended graph notion. This new method also helps to implement real life considerations into the algorithm in more effective manner. A novel workstation oriented line balancing algorithm is proposed to solve the multi-objective ALB model which complies with the new representation technique. This algorithm can start solving the problems at an arbitrary location compared to the conventional method of sequential forward and backward assignment. The algorithm is allowed to search in the infeasible regions while trying to minimize the degree of violation. These characteristics make the algorithm less prone to
getting trapped. In addition to that, the algorithm is capable of finding potential good solutions where the user might agree to allow minor trade-off on feasibility criteria. To deal with the multi-objective aspects of the problem, Pareto fronts are generated using a robust sorting technique based on NSGA II. An interactive user interface is developed where the user is allowed to update the input parameters at any time, making it suitable for reconfigurable facilities. Finally, the effectiveness of the proposed method is verified using a series of well-known test problems from literature. The proposed novel algorithm has demonstrated promising results in all test problems. The proposed algorithm is designed to solve a representative multi-objective model from versatile types of ALB problems. Some extension of the same methodology can be used to solve many other generalized assembly line balancing problems with real life features such as – stochastic processing time, assignment restrictions, processing alternatives, U-shaped lines, mixed model lines, machine breakdown, and the like.
Acknowledgements

First of all, I would like to thank the Almighty for all of His blessings in my life. I would like to express my sincere gratitude to Dr. Mohamed Ismail for his continuous support, guidance and mentoring throughout my M.A.Sc. program. I am also grateful to my thesis committee members, Dr. Amr Henni and Dr. Ezeddin Shirif for their kind support and recommendations.

I would like to thank Faculty of Graduate Studies Research for providing me a number of scholarships and awards, and Faculty of Engineering and Applied Science for providing Teaching Assistantship positions several times during my program. Their support was a great help for me to focus on my research.

I am grateful to some fellow graduate students and Alumni of the University of Regina for providing me moral support and motivation during some difficult times, Dr. Kazi Sumon, Imteaz Bhuiyan and Rashed Chowdhury to name a few. Thanks to my wife Sajia Afrin Ema for her faith, patience and motivation during my difficult times. Finally, I would like to thank my family for their unconditional love and support throughout my life.
Dedication

To my family
Table of Contents

Abstract .................................................................................................................. i

Acknowledgements ............................................................................................... iii

Dedication .............................................................................................................. iv

Table of Contents .................................................................................................. v

List of Tables ......................................................................................................... ix

List of Figures ....................................................................................................... x

Nomenclatures ..................................................................................................... xi

CHAPTER ONE: Introduction ............................................................................... 1

1.1 Background .................................................................................................... 1

1.2 The Assembly Line Balancing Problem and its Scope ............................. 3

1.3 Classifications of Assembly Line Balancing Problems ....................... 5

1.3.1 SALBP and GALBP .................................................................................. 5

1.3.2 Based on precedence graph characteristics ........................................ 8

1.3.3 Based on station and line characteristics ............................................. 12

1.3.4 Based on objectives ................................................................................ 16

1.4 Problem Statement ...................................................................................... 17

1.5 Research Objectives .................................................................................... 17

1.6 Outline .......................................................................................................... 18
CHAPTER TWO: Review of Solution Approaches for Line Balancing Problems

2.1 Overview .................................................................................................................. 19
2.2 Priority rule-based approaches .............................................................................. 21
2.3 Branch and Bound Procedures .............................................................................. 22
2.4 Dynamic Programming .......................................................................................... 25
2.5 Genetic Algorithms ............................................................................................... 26
  2.5.1 GA Methods ........................................................................................................ 27
  2.5.2 Chromosome representation ............................................................................. 30
  2.5.3 Feasibility ............................................................................................................ 35
  2.5.4 Initial population ............................................................................................... 36
  2.5.5 Selection ............................................................................................................. 37
  2.5.6 Crossover operator ............................................................................................ 38
  2.5.7 Mutation operator .............................................................................................. 44
  2.5.8 Survival ............................................................................................................... 46
2.6 Tabu Search ............................................................................................................. 46
2.7 Simulated Annealing .............................................................................................. 47
2.8 Ant Colony Algorithms .......................................................................................... 49
2.9 Honey Bee Algorithms ........................................................................................... 51
2.10 Bacterial Foraging ............................................................................................... 53
CHAPTER THREE: Problem Formulation and the Multi-Objective Optimization Algorithm

3.1 Outline of the Model .................................................................................................................. 56
3.2 Nomenclature ............................................................................................................................. 56
3.3 Objectives .................................................................................................................................. 57
  3.3.1 $O_1$: Minimizing Number of Workstations ................................................................. 58
  3.3.2 $O_2$: Minimizing Cycle Time ......................................................................................... 59
  3.3.3 $O_3$: Maximizing Line Efficiency .................................................................................... 59
  3.3.4 $O_4$: Minimizing Line Smoothness Index ....................................................................... 60
  3.3.5 $O_5$: Minimize infeasibility index in terms of cycle time ........................................... 60
  3.3.6 $O_6$: Minimize infeasibility index in terms of number of station .................................. 61
3.4 Multi-Objective Optimization Algorithm .................................................................................. 61
  3.4.1 Fast Non dominated Sorting ............................................................................................ 62
  3.4.2 Density estimation and crowded comparison approach ................................................. 63
  3.4.3 Main loop ......................................................................................................................... 65
  3.4.4 Constraints handling ........................................................................................................ 66

CHAPTER FOUR: Solution Algorithm .............................................................................................. 68

4.1 Introduction to the Solution Algorithm ..................................................................................... 68
4.2 Solution Representation ............................................................................................................. 68
4.3 Initialization Algorithms ........................................................................................................... 69
4.4 Improvement Algorithms .......................................................... 71
4.5 Master Algorithm ........................................................................ 71
CHAPTER FIVE: Software Developed and Results ................................ 73
  5.1 Introduction to Software Interface and Benchmark Problems .......... 73
  5.2 Input data .................................................................................. 73
  5.3 Output Form ............................................................................. 76
  5.4 Benchmarking and Results .......................................................... 78
CHAPTER SIX: Conclusions and Future Research ............................ 79
References ....................................................................................... 81
List of Tables

Table 1: SALB Assumptions ........................................................................................................... 6
Table 2: A map for the types of SALB problems ............................................................................. 7
Table 3: Computational complexity of SALB problems ................................................................. 7
Table 4: Overview of solution approaches for ALB problems ..................................................... 20
Table 5: List of priority rules .......................................................................................................... 21
Table 6: A fast non-dominated sorting procedure [118] .............................................................. 63
Table 7: Main loop of NSGA II [118] .......................................................................................... 65
Table 8: Constraint handling approach in NSGA II for single objective problems .............. 66
Table 9: Initialization algorithm # 1 (SALBP type - 1) ............................................................... 70
Table 10: Initialization Algorithm # 2 (SALBP type - 2) ............................................................ 71
Table 11: Master algorithm based on NSGA II ............................................................................. 72
Table 12: Results for benchmark problems .................................................................................. 78
List of Figures

Figure 1: A broad classification of Assembly lines ............................................................. 2
Figure 2: Performance of a typical unbalanced assembly line ........................................... 5
Figure 3: Classification of ALBP based on precedence graph characteristics ............... 11
Figure 4: Classification of ALBP based on station and line characteristics .................. 16
Figure 5: Classification of ALBP based on objective ......................................................... 17
Figure 6: An example precedence graph (from Mansoor-11 tasks problem) [69] ............ 31
Figure 7: Density estimation and crowded distance calculation ..................................... 64
Figure 8: The novel individual assembly line graph G (V, E, W) ..................................... 69
Figure 9: Precedence diagram data without task description ............................................. 74
Figure 10: Precedence diagram data with task description ................................................. 74
Figure 11: Production rate and cycle time calculation form ............................................. 75
Figure 12: Objectives selection form ............................................................................... 76
Figure 13: Assembly line balancing hosted in FactDesign Environment [Ismail, 2016] ... 77
## Nomenclatures

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALBP</td>
<td>Assembly Line Balancing</td>
</tr>
<tr>
<td>SALBP</td>
<td>Simple Assembly Line Balancing Problem</td>
</tr>
<tr>
<td>GALBP</td>
<td>Generalized Assembly Line Balancing Problem</td>
</tr>
<tr>
<td>G</td>
<td>Graph</td>
</tr>
<tr>
<td>V</td>
<td>Vertex</td>
</tr>
<tr>
<td>E</td>
<td>Edge</td>
</tr>
<tr>
<td>W</td>
<td>Workstation</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithms</td>
</tr>
<tr>
<td>B&amp;B</td>
<td>Branch and Bound</td>
</tr>
<tr>
<td>( t_i )</td>
<td>Processing time of task ( i ), ( i = {1, ..., n} )</td>
</tr>
<tr>
<td>( S_s )</td>
<td>Set of tasks assigned to station ( s ), ( s = {1, ..., m} )</td>
</tr>
<tr>
<td>( T_s )</td>
<td>Station time of station ( s ), ( (T_s = \sum_{i \in S_s} t_i) )</td>
</tr>
<tr>
<td>( n )</td>
<td>Total number of tasks</td>
</tr>
<tr>
<td>( m )</td>
<td>Theoretical total number of stations</td>
</tr>
<tr>
<td>( m_R )</td>
<td>Realized total number of stations</td>
</tr>
<tr>
<td>( C_T )</td>
<td>Theoretical cycle time ( (C_T = \sum_{i=1}^{n} t_i/m) )</td>
</tr>
<tr>
<td>( C_R )</td>
<td>Realized cycle time</td>
</tr>
<tr>
<td>( X )</td>
<td>A line balance solution</td>
</tr>
<tr>
<td>( O_t(X) )</td>
<td>( t )-th objective function value of solution ( X ), ( t = {1, ..., 6} )</td>
</tr>
<tr>
<td>( d_p )</td>
<td>Number of solutions dominate ( p )</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>$D_p$</td>
<td>A set of solutions which are dominated by $p$</td>
</tr>
<tr>
<td>$&lt;_d$</td>
<td>Crowded comparison operator</td>
</tr>
<tr>
<td>$N$</td>
<td>Set of tasks to be assigned, $i \in N$</td>
</tr>
<tr>
<td>$M$</td>
<td>Set of workstation under consideration</td>
</tr>
<tr>
<td>$CX$</td>
<td>Constraints</td>
</tr>
<tr>
<td>$F_r$</td>
<td>$r$-ranked Front</td>
</tr>
<tr>
<td>$F_1$</td>
<td>Pareto Front</td>
</tr>
</tbody>
</table>
CHAPTER ONE:  

Introduction  

1.1 Background  

The underlying idea of assembly line systems is the concept of division of labor i.e. dividing a total work into smaller work elements to obtain better productivity. Having the same worker repetitively working on a similar set of operations contributes to the efficiency of the worker. A critical requirement for having an effective an assembly line is to have the mating parts and fasteners within a predetermined close tolerance, i.e. ensuring interchangeability of parts. American automotive industrialist Henry Ford and his colleagues are often credited with developing the first well-designed assembly line in the industry for magneto flywheel production in Highland Parks, Michigan, in 1913, leading to a fourfold increase in productivity. That idea was inspired by the meat packing operations from slaughtered stock in Chicago, Illinois, where the slaughtered stock is moved from one worker to the next by conveyors allowing each worker to concentrate on their part of work [1].

The design of an assembly line starts with breaking down a total work into sensibly most basic operations, typically named as work elements or tasks. These work elements are assigned to specific workstations. The workload of a workstation and nature of the operations dictates the number of people required at each station. Apparently, this is vital to have the workload well distributed across the workstations. Otherwise, a workstation where the assigned works are done far quicker than adjacent workstations would lead a workstation to block the downstream workstation and starve from the upstream
workstation. Although sometimes buffers are used to mitigate such circumstances, it is still a huge concern to have a ‘balanced’ assembly line to minimize the work in process inventory and maximize production efficiency.

Based on physical attributes and nature of operation the assembly lines can be of different types as shown in Figure 1. The classification presented here is in a broader context, a detailed hierarchical classification of assembly line balancing problem with their particular characteristics is addressed in the following section.

![Figure 1: A broad classification of Assembly lines](image)

With more emphasis on sustainability and recycling, the disassembly lines have got more attention only in recent days compared to the traditional assembly lines. In terms of the level of automation, an assembly line can be fully automated, semi-automated or manual. Use of higher degree of automation is often decided based on capital investment required and availability of technology.

If the assembly line under consideration is being balanced for the first time during its initial setup, the balancing process becomes simpler; However, if there have been already a setup and a further rebalancing is required, consideration for any incurring changes have to be addressed.
1.2 The Assembly Line Balancing Problem and its Scope

The assembly line balancing problem (ALBP) addresses the issue of assigning work elements and workers into different workstations to maximize the production efficiency using a minimum number of workstations and/or minimum cycle time. A cycle time is defined as the maximum station time of all workstations or the time elapsed between two consecutive dispatches from a workstation. In a perfectly balanced assembly line all the workstations will have same station times, in other words, the workloads are equally distributed among all stations. The cycle time includes both of the service time at a station and the repositioning time following the operations at that station.

The inputs to the assembly line balancing problems are [2]:

(a) Work breakdown into elements and their precedence relationship

(b) Time for each work element

(c) Number of workstations or cycle time

The precedence relationship constraints the work elements to be sequenced in required orders and often represented in the form of precedence network diagrams. The processing times for each work elements or tasks are usually obtained from time studies which can be deterministic or probabilistic.

The objective of assembly line balancing problem can be expressed in a typical form as following [1]:
\[
\text{Minimize } \left( m \cdot C_R - \sum_{i=1}^{n} t_i \right)
\]

Or,

\[
\text{Minimize } \left( \sum_{s=1}^{m} (C_R - T_s) \right)
\]

Subject to:

(a) The sum of all work elements at a station must be equal to or less than the cycle time. i.e. \( T_s \leq C_R \)

(b) All of the precedence constraints must be complied with

Another key parameter commonly used for measuring the performance of an assembly line is balancing efficiency. The balancing efficiency is defined as following:

\[
\text{Balancing Efficiency, } E_b
\]

\[
= \frac{\text{Sum of times to complete each work elements}}{\text{Number of workstations} \times \text{Cycle time}} \times 100\%
\]

\[
= \frac{\sum_{i=1}^{n} t_i}{m \cdot C_R} \times 100\%
\]

From the above-mentioned relation, this is clear that to obtain a required production rate using a target balancing efficiency, the total number of workstations would be inversely related to the cycle time.

Another measure is also in practice named as balance delay, which is the exact opposite of the balancing efficiency and defined as following:
Balance delay, \( d = \frac{m \cdot C_R - \sum_{i=1}^{n} t_i}{m \cdot C_R} = 1 - E_b \)  

Figure 2 demonstrates an overview of the idea of typical assembly line balancing problem with some arbitrary data as following:

![Performance of a typical unbalanced assembly line](image)

**Figure 2: Performance of a typical unbalanced assembly line**

### 1.3 Classifications of Assembly Line Balancing Problems

In an effort to ease the communication between the researchers and practitioners, a classification scheme is proposed by Boysen et al. [3] which is frequently being quoted in the literature published afterward. The discussion of this section would be centered on their classification scheme; however, with some extensions and additional comments.

#### 1.3.1 SALBP and GALBP

The most basic and classical scheme for classification of assembly line balancing (ALB) problem divides them into two main categories i.e. (a) Simple assembly line balancing (SALB) problem, and (b) General assembly line balancing (GALB) problem. The SALB
category of the ALB problems is the simplest and most studied category. As it reveals by the name itself, the SALB models are based on many assumptions which may not essentially reflect real life industrial situations. The SALB problems are identified with some limiting assumptions [4-6] as listed in Table 1.

Table 1: SALB Assumptions

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>A single homogenous product will be produced at mass scale.</td>
<td>6.</td>
<td>No assignment restrictions other than precedence constraints are not considered.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>No processing alternatives will be considered</td>
<td>7.</td>
<td>A task cannot be split and assigned to more than one station.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>The production line will be paced.</td>
<td>8.</td>
<td>All stations are equally equipped with the machines and workers.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>Only serial line models are considered with no parallel operation.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>The task times are deterministic.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The GALB category of ALB problems deals with the problems other than SALB. Many GALB problems are extended versions of SALB problem cases and models. GALB problems may accommodate a large variety of problems relaxing some of the assumptions to address multi models cases, zoning constraints, the existence of parallel stations and more [4]. Such cases are fully listed and further discussed in the following sections.

Based on the optimization objective the SALB problems are classified into four further types [7]:

6
(a) SALBP type – 1: In this type of problem a predetermined cycle time is given, and the optimization goal is to minimize the number of workstations.

(b) SALBP type – 2: If the number of stations is given and the objective is to minimize cycle time the problem would belong to type SALBP – 2.

(c) SALBP type – E: In this type of problems both of the numbers of stations and cycle times are variables. The objective of the problem is to maximize the line efficiency.

(d) SALBP type – F: In SALB type F problems both of the number of stations and cycle times are given and the objective of the problem is to find a feasible balance.

<table>
<thead>
<tr>
<th>Type of SALB</th>
<th>Given</th>
<th>Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALBP type – 1</td>
<td>Cycle time</td>
<td>Min (Number of stations)</td>
</tr>
<tr>
<td>SALBP type – 2</td>
<td>Number of stations</td>
<td>Min (Cycle time)</td>
</tr>
<tr>
<td>SALBP type – E</td>
<td>Cycle time &amp; Number of stations</td>
<td>Max (Balancing efficiency)</td>
</tr>
<tr>
<td>SALBP type – F</td>
<td>Cycle time &amp; Number of stations</td>
<td>Feasible balance</td>
</tr>
</tbody>
</table>

As this is evident that these problems belong to a combinatorial category, the computational complexity of different types are listed below [6, 8]:

<table>
<thead>
<tr>
<th>Type of SALB</th>
<th>Computational complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>SALBP type – 1</td>
<td>NP-hard</td>
</tr>
<tr>
<td>SALBP type – 2</td>
<td>NP-hard</td>
</tr>
<tr>
<td>SALBP type – E</td>
<td>NP-hard</td>
</tr>
<tr>
<td>SALBP type – F</td>
<td>NP-complete</td>
</tr>
</tbody>
</table>
1.3.2 Based on precedence graph characteristics

A precedence graph is a great tool to represent the sequential constraints and relationships among different tasks in an assembly line in the form of a diagram. The tasks are usually represented in the form of vertices, and a set of edges demonstrates their inter-relationships. As more advanced features important characteristics like processing alternatives and processing times of tasks can be represented using weights assigned to vertices and edges.

1.3.2.1 Product specific precedence graph

Single model: Using dedicated assembly line for a single model constitutes a relatively simpler problem. This option is a better fit for mass production of consumer products which do not require any customization.

Mixed model: The mixed-model assembly line balancing problem category addresses the assembly lines where different models are assembled on the same line provided the models are similar enough that the setup times between two models are negligible. Different models are assembled in an arbitrary sequence. In most studies regarding mixed model assembly line problems, a mixed model line is transformed into a single model line. There two common methods used for the transformation. In the first method, multiple precedence graphs for different models are combined to generate a new single precedence graph that is capable of addressing all the required processes for all different models under consideration during a particular shift [9]. Another common method is to determine average task processing times for tasks that belong to more than one model.

Multi-model: In many cases, a single line can be used for assembly of multiple models which are considerably decisive and similar in terms of processes. However, the degree of
homogeneity of the precedence requirements varies which it makes sense to use the same assembly line. In such cases, the assembly is carried out in batches. In between two batches, there are set up requirements in the line raising demand for additional time and resources. In addition to balancing problem, this type of assembly line comes with an additional requirement for sequencing and batch/lot sizing in order to obtain an effective optimal solution [9, 10]. Although, Boysen et al. mentioned both of them are not part of the long term or medium term configuration decisions [3].

1.3.2.2 Processing times

Deterministic processing times: Despite the fact that the variability will be present in a manual assembly line, deterministic task times is used for many models provided such compromises are allowed and addressed in the assumptions of the model.

Stochastic processing times: The stochastic model is based on the assumption that the task times will follow a predetermined probabilistic distribution function. Examples of such models include models suggested by Sarin et al. [11] Xiao et al. [12].

Dynamic processing times: Some of the known issues such as learning curves can be addressed by means of incorporating the function for their learning curve [13].

1.3.2.3 Sequence dependency of task times

Direct dependency: A direct dependency of task times on sequence of tasks occurs when set up is required in between two immediate tasks [14].
Indirect dependency: An indirect dependency of task times on sequence of tasks occurs when a status of the workpiece achieved by certain operation affects the task times for any subsequent operation at the same or different station.

1.3.2.4 Assignment restrictions

Task-related constraints (zoning constraints): A linked subset of tasks may be required to be assigned in immediate sequence to use non-duplicable resources or use same environmental condition and the like. This constraint is incorporated into the model as maximum distances. Incompatible tasks may be constrained by introducing minimum distances [5].

Station related constraints: If some tasks require specialized resources which are not easily movable or duplicable, these tasks must be assigned to corresponding stations only [3, 15].

Position related constraints: In cases where certain tasks require being executed from certain sides, tasks can be grouped based on corresponding sides [16].

Operator related constraints: For some tasks advanced and specialized skill may be required and that has to be performed by a certain worker. The task usually determines the requirement of skill level for a station with the highest skill requirement in that station.

1.3.2.5 Processing alternatives

Processing alternatives allow the engineers to consider rerouting of the operations, thus changing the precedence order. This kind of problem simultaneously works on alternate processing options reordering the subgraphs and assignment of tasks to workstations. For
example, such a model is formulated by Pinto et al. [17]. Three kinds of processing alternatives are listed based on their effects by Boysen et al. [3]. The first kind results in deviation of processing times and costs only. In this kind of problems, a trade-off is made between the time and cost. The second kind affects relationships between tasks in addition to the times and costs. Thus, if an alternative placement of a task is executed it changes precedence relations of some other task(s) as well; however, they must not violate any imposed restriction. The third kind makes a complete part of the assembly process interchangeable. So, depending on the model requirement, the whole subgraphs are substitutable and can be replaced as the scenario dictates (e.g. tasks for installing parts of a module vs. installing equivalent preassembled module).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure3.png}
\caption{Classification of ALBP based on precedence graph characteristics}
\end{figure}
1.3.3 Based on station and line characteristics

1.3.3.1 Movement of workpieces

Paced: A paced assembly line is controlled by a predetermined cycle time. All stations must execute their assigned tasks within this cycle time. Often a timed conveyor or equivalent mechanisms are used to implement a paced assembly line. Based on the nature of cycle time restriction a classification is presented here.

(a) Strict requirement: Each model must strictly comply with the cycle time.

(b) Average requirement: The cycle time restricts the work content per cycle for each station.

(c) Probabilistic requirement: The cycle time restrictions are required to comply with a given probabilistic distribution (stochastic task time).

(d) Same global cycle time: A common global cycle time is to be respected.

(e) Diverging cycle times: In cases where local delays can occur may put forward a requirement for locally diverging cycle times [18].

Unpaced: The unpaced assembly lines are not constrained by a predetermined cycle time. The workpiece stays at certain workstation until all required operations are carried out and then put forward to the next station satisfying any further constraints.

Unpaced asynchronous line: In an unpaced asynchronous assembly line each workstation delivers the workpiece to the next station once work at that station is completed. It is common to use buffers in this kind of lines [19]. Recently a concept
of Intelligent Navigation of Assembly Stations (INoAS) is proposed based on real information, guidance and re-queuing of tasks [20].

(a) Unpaced synchronous line: In this type of lines, all the workstations have to wait for an uncertain time until all the stations are not done with their operations to achieve the simultaneous transfer of workpieces or materials [19].

1.3.3.2 Line Layout

Serial lines: Although this layout is very common due to the simplicity and compatibility with flow line production, it may lead to poor job enrichment. In the case of failures or changing demand rates, the sensitivity and flexibility of this type of lines are quite rigid.

U-shaped lines: U-shaped lines are capable of incorporating features of relatively modern concepts such as Just in time (JIT) and Group technology. The workstations are laid out in a U-shape, and a worker may work at two facing segments of the line interchangeably for a pair of tasks. This layout also helps operators to get involved in different parts of the production process enhancing their skills, better visibility of the whole assembly process, better communication among workers, sharing worker or machine capabilities in case of congestion. The flexibility helps to re-balance the line in cases (e.g. JIT) where demands for different parts dynamically changes [5, 21, 22]. As understandable, most of the literature addresses problems with single U-shaped lines but recently lines with multiple U-shaped segments are also considered [23, 24].

1.3.3.3 Parallelization

Parallel lines: Having multiple assembly lines in parallel allows a quick response to a rapid fluctuation in demand or a sudden machine breakdown since the number of lines in
operation can be controlled. This certainly comes with a trade-off for the higher capital investment. For assembly lines with parallelization options, the number of lines can be predetermined or can be part of the problem itself. A reflective production concept is discussed by Ellegård et al. which addresses social and economic advantages of such lines [25]. Further aspects of reflective production concept are discussed, and an evaluation of it in comparison to lean production under the context of mass production is presented by Freyssenet et al. [26]. Recently, assembly line problems with multiple parallel line options have been extended to the studies regarding multiple U line problems [27].

Depending on whether one or both side of the line is utilized the lines can be classified as single sided or two sided. In two-sided lines, workers at two ends are usually given analogous or mirrored jobs. Use of two-sided lines can reduce capital cost, the overall length of the line, material handling cost, the cost of tooling, worker movement and more [28, 29].

Parallel workstations: Parallel stations (may not be physically parallel) in an assembly line allows to have multiple stations with same equipment which are capable of carrying out identical tasks at any individual station. Using multiple parallel options provides a mechanism for responding to demand fluctuations and minimizing the cycle time [30, 31].

Parallelized task: The assembly line can take advantage of a task being processed at different stations at the same time. If there is any flexibility of dividing a task with largest processing time, it may result in significant reduction in the cycle time [30, 31].
**Parallel workpiece:** For large products or parts where designs are symmetric, a mirrored assignment of the tasks to both sides can result in better productivity which is essentially only a special case of two-sided lines. In this case, measures have to be taken to avoid interferences among workers [28, 29]. A group wise task assignment method is suggested by Kim et al. with emphasis on maximizing work-relatedness and work slackness for two-sided lines for large products [32].

1.3.3.4 **Resource assignment**

*Equipment selection problems:* In the case of assembly line design problems simultaneous decision problems are solved for line balancing and selecting equipment for each station. The set of available equipment is usually predetermined [33].

*Equipment configuration problems:* Unlike the equipment selection problems, the same equipment can be re-configured according to the assigned task to the station.

1.3.3.5 **Additional aspects of line configuration**

Depending on the attributes in an assembly line some additional features may be considered during modeling such as the use of buffers or feeder lines. The model may also be developed to count for material handling issues and unproductive time in the line. There could be one or more of these features in the same model since they are selective features.
1.3.4 Based on objectives

The classification of Assembly line balancing problems based on their objectives is relatively straightforward. Among the list below the ALBP with objectives for minimizing the number of stations, minimizing the cycle time, maximizing the line efficiency and the feasible solution is analogous to SALB Type – 1, Type – 2, Type – E and Type – F as discussed in the SALBP section. Although having the objective same, the model might be incorporating some extensions beyond the SALBP limiting assumptions. Rest of the problems are also fairly self-explanatory. Hence, a list is provided and the types are not further elaborated.
1.4 Problem Statement

In this thesis, a novel approach to solving the multi-objective assembly line balancing problem is presented. The problem is represented as an extended directed graph $G(V, E, W)$ where any vertex can be a start-up location for allocating workstations. The algorithm developed will use the Non-Dominated Sorting Genetic Algorithm (NSGAII) to address the multi-objective aspect of the problem. A new extensible framework is proposed to handle more generalized and more practical version of the problem.

1.5 Research Objectives

1. Develop a novel algorithm to solve the Generalized Assembly Line Balancing Problem (GALBP).

2. Address the multi-objective aspect by incorporating the NSGAII.

3. Develop an extensible framework to address several versions of the problem.
4. Develop a user-friendly software tool for system users.

1.6 Outline

This thesis will be presented in six chapter that can be summarized as follows:

Chapter 1 introduces the problem and demonstrates several related aspects, defines the problem statement, and lists the research objectives.

Chapter 2 review the several solution approaches developed in the literature developed to address this problem.

Chapter 3 introduces the problem formulation and some principles related to multi-objective optimization.

Chapter 4 introduces the solution representation and the solution algorithm.

Chapter 5 demonstrates the developed software tool and several benchmark problems solutions.

Chapter 6 concludes the thesis, demonstrate its contributions, and discusses future research.
CHAPTER TWO:

Review of Solution Approaches for Line Balancing Problems

2.1 Overview

A wide range of approaches has been developed over the past years to solve ALB problems. At the early stages, the concentration was on SALB problems only and has been well studied in the literature. Mathematical programming models like Binary programming (BP), Integer programming (IP) and Mixed Integer Programming (MIP) have been developed, and algorithms like Branch and Bound procedures were used to solve them. At the same time, heuristic algorithms were also being developed (e.g. COMSOAL[34]) to optimize certain objective functions. Due to the computational complexity of ALB problems, use of meta-heuristics seems to be more effective, especially when solving large problem instances. Starting in the early 90s, meta-heuristics like Simulated Annealing (SA) and Genetic Algorithms (GA), etc. have been gaining popularity. Some recent work has been done on the use of Ant colony optimization (ACO) algorithm, Honey bee optimization algorithms (HBA) and Bacterial foraging techniques and the like. In some of the approaches, Hybrid algorithms are implemented taking advantage of multiple algorithms. With the advent of meta-heuristic approaches, the scholars have the privilege to explore for more generalized approaches for solving ALB problems more effectively and efficiently. This chapter gives a brief overview of different solution approaches used for solving ALB problems. However, an intensive review of Genetic Algorithms for solving ALB problems is provided.
<table>
<thead>
<tr>
<th>Category</th>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exact Solution Approaches</strong></td>
<td>Dynamic Programming</td>
</tr>
<tr>
<td></td>
<td>Branch and Bound Method</td>
</tr>
<tr>
<td><strong>Heuristic Approaches</strong></td>
<td>Priority rule based</td>
</tr>
<tr>
<td></td>
<td>Incomplete enumeration</td>
</tr>
<tr>
<td></td>
<td>Other Search Methods</td>
</tr>
<tr>
<td><strong>Metaheuristic Approaches</strong></td>
<td>Genetic Algorithms</td>
</tr>
<tr>
<td></td>
<td>Simulated Annealing</td>
</tr>
<tr>
<td></td>
<td>Swarm Optimization</td>
</tr>
<tr>
<td></td>
<td>Ant colony Optimization</td>
</tr>
</tbody>
</table>

Most of the algorithms develop to date have incorporated either of the following assignment schemes: task oriented assignment or station oriented assignment. In task oriented method, a single task is selected and an available station is searched and assigned to the earliest station found. On the other hand, in station oriented method, from a set of available tasks, station load is developed until there is no more space in the station. Once the station is full and can’t accommodate any more task, it is closed, and a new station is created [35]. Irrespective of construction scheme, all of the methods uses string-based representation in order to generate solutions. Further detail on representation schemes is discussed in 2.5.2.
2.2 Priority rule-based approaches

Priority rule based heuristics are very familiar to solve ALB problems. Although they often appear to be highly exhaustive, they were widely used as the means to construct solutions in other exhaustive, heuristic and meta-heuristic algorithms. Typically, a prioritized list of tasks is developed, and the stations are assigned according to their position on the list respecting the precedence constraints. Such kind of rules include [35]:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Descending task time</td>
</tr>
<tr>
<td>2.</td>
<td>Descending positional weight</td>
</tr>
<tr>
<td>3.</td>
<td>Descending number of followers</td>
</tr>
<tr>
<td>4.</td>
<td>Descending number of immediate followers</td>
</tr>
<tr>
<td>5.</td>
<td>Ascending earliest station</td>
</tr>
<tr>
<td>6.</td>
<td>Ascending latest station</td>
</tr>
<tr>
<td>7.</td>
<td>Ascending slack</td>
</tr>
<tr>
<td>8.</td>
<td>Descending Average positional weight</td>
</tr>
<tr>
<td>9.</td>
<td>Ascending average positional weight</td>
</tr>
<tr>
<td>10.</td>
<td>Ascending average latest station</td>
</tr>
<tr>
<td>11.</td>
<td>Descending task time divided by latest station</td>
</tr>
<tr>
<td>12.</td>
<td>Descending number of followers divided by slack</td>
</tr>
<tr>
<td>13.</td>
<td>Descending total number of following arcs</td>
</tr>
<tr>
<td>14.</td>
<td>Descending cumulated number of following arcs</td>
</tr>
<tr>
<td>15.</td>
<td>Descending cumulated positional weight</td>
</tr>
<tr>
<td>16.</td>
<td>Ascending task number</td>
</tr>
</tbody>
</table>

Table 5: List of priority rules

Scholl et al. mentioned several ways to construct solutions using priority rule-based approaches [6]. Traditionally, the unidirectional solutions are constructed using priority rules. In the flexible bidirectional construction, an assignment is considered in both forward
and backward directions. In station oriented procedure the task is selected based on priority rule and assignment to both of the earliest and latest possible station is considered. A backward search can be obtained by using same priority rules for a reversed graph. In task-oriented methods, both of the forward and backward available tasks are considered, and the one with the highest priority is chosen. Some priority rule (e.g. rule 12) can be updated dynamically when the only partial solution is generated [36, 37]. The priority rules can also be selected stochastically or over a number of iterations [37, 38].

2.3 Branch and Bound Procedures

Branch and Bound procedures are often used to solve Integer Programming (IP) or Mixed Integer Programming (MIP) models based models of ALB problems. One major step of Branch and Bound procedure for both of type – 1 and type – 2 ALB problem is deciding on how to determine the Lower Bound (LB). In the case of type – 1 problems the Lower Bound is the lowest possible number of workstations and for type – 2 problems the Lower Bound is the lowest possible cycle time.

The research in this area looks into exploring different search strategies, analyzing different construction schemes, improving the lower or upper bound criteria, the possibility of using dominance rules and reduction rules and more [6, 39, 40].

The search strategies for Branch and Bound procedures discussed in the literature include depth-first search (DFS), depth-first search with complete node development (DFSC), minimal lower boundary (MLB) strategy and local lower bound method (LLBM).
As the name implies, the depth-first search method develops a whole single branch at a time. Sometimes, a node can be fathomed and development of the rest of the branch is not required. Not looking into alternative branches at the same time gives an advantage on memory. After reaching the leaf node, the branch is traced back to the node where first alternative branch is found. Eventually, that alternative branch is developed. Thus each node is developed and evaluated completely (i.e. for all of the following branches), before going back to an ancestor node. A common way of implementing depth-first search method is the laser search technique. In laser search, for a current node, only one following node is generated at a time [41]. An alternative approach named as depth-first search with complete node development allows to develop all following alternative nodes of the current node and use some prioritization criterion to sort them. According to the priority, branches on the nodes (one node at a time) are further developed. The remaining nodes are stored on a list of candidates. When the search return to an ancestor node by back tracing, the next node is selected from candidate list based on priority and branch is developed [42].

The local lower bound method incorporates a relatively different approach trying to take advantages of the other methods. By approaching a node, the following potential branches are grouped into two categories: category I and category II. Category I includes following subtrees where the local lower bound is the same as the current node. Category II includes following subtrees where the local lower bound has to be increased with respect to the current node. Following nodes having local lower bound same as the global upper bound are immediately fathomed and are not branched any further. Subtrees belonging to category I are branched first, and subtrees belonging to category II are saved for branching later if needed. If a following subtree has a local lower bound less than the upper bound ( an
indication of improvement) at that node level, all category II is dropped and the algorithm stops when enumerating category I is completed. If there is no subtree with the same local lower bound value as the current node, the lower bound values are increased by 1 and category II is subtrees are enumerated. Otherwise, the node is fathomed [43]. The simplest lower bound is the theoretical minimum number of station given by the sum of all work elements divided by cycle time. Pape has listed seven local bound criteria from a range of literature for SALB type – 1 problems [40]. Lower bounds for some other ALB problems are outlined by Scholl et al. [6].

Further to that, Dominance Rules and Reduction techniques may be used to improve the algorithm efficiency and reduce computation efforts. The idea of dominance rule takes advantage of evaluating a partial solution before computations for residual problems take place. If a partial solution is proven to not ending in a better solution, it can be dropped at that point before a complete solution is generated. Two incomplete solutions are compared provided a predetermined dominance relationship between them exists, the dominant partial solution is further explored and the other one is dropped. There is a range of dominance rules which include Jackson’s maximum load rule [44], Jackson’s dominance rule extended by Scholl and Klein [43], Feasible set dominance rule [39] and Station ordering rules [45] etc. The reduction rules are used to reduce computational effort modifying the problem data. For example, a task with processing time very close to the cycle time may mean no other tasks can be assigned to the station accommodating that task. In order to let the algorithm avoid such unnecessary effort, the task processing time can be incremented to the cycle time [41]. Some more reduction rules are prefixing [39], additional precedence constraints and task conjoining rule [46], etc. Recently, Morrison et
al. suggested a Branch, Bound and Remember (BB&R) method where the algorithm remembers all previously generated subproblems and uses this information in case a similar condition appears again [47]. Talbot et al. have described the idea of a ‘network cut’ to eliminate considerations of assignment of tasks which may not lead to improved line balances [48].

2.4 Dynamic Programming

Dynamic programming has been considered as a popular tool for multi-stage decision-making problems, particularly for graph-based problems. In this approach, a problem is divided into a set of subproblems which are solved sequentially, and at the end, the original problem is solved by a compilation of solutions for subproblems.

All feasible sets of solutions to a subproblem are labeled as different states on the graph. These states contain information about partial solutions. A state space is defined as a set of all possible states. For example, in the context of SALBP type – 1 problems, a feasible set of task assignment at certain stage may be considered as a state. The enumeration can start from a stage zero, where the very first station is to be developed and a stage by stage forward recursion method is used. At each stage, all feasible sets of task assignments are generated according to the cycle time requirement and precedence constraints [44, 49]. This implies that a node on the graph can be considered as a state which indicates a workstation load (a feasible set of tasks which are assigned to that station) and the quality of the solution can be shown on the arcs in terms of idle time. Each of the arcs would lead to a feasible solution and the optimal (shortest arc) one is selected at each stage. This way the SALBP problems can be transformed into a shortest path problem [50].
Again, the number of states would be increasing exponentially with increasing number of tasks for the problem under consideration [51]. There are a number of methods suggested in literature to reduce the size of the graph and not to enumerate relatively worse solutions. Some of these approaches consider the use of bounds that takes advantage of the idea that, it may not be required to further enumerate a solution as soon as the solution is already proven to be same as or worse than the current best solution. Which means a partial solution to a subproblem can be stopped even before it is completed. Please refer to the previous section on branch and bound procedures for example of criteria to set bounds. A bounded dynamic programming approach also uses heuristics methods to reduce the state space in addition to the bounds as suggested by Bautista et al. [52].

2.5 Genetic Algorithms

There has been a review of the current applications of genetic algorithms in assembly line balancing by Tasan et al. [53]. They have outlined the use of different GA features and operators in detail particular to the line balancing problems. However, the review identifies the type of assembly lines discussed based on whether they are SALB or GALB, the number of models and their objectives in their problem specifications. The review was published in 2007 and there has been further work in the area in the following years. Tasan et al. suggest an admirable structural framework for reviewing the application of genetic algorithms in assembly line balancing. We adopt their structural framework and extend their review with a more detailed taxonomy of assembly line balancing problems being addressed (mentioned in the previous section) and to the very recent contributions. There are three parts of this structural framework such as: (1) Problem specifications: addresses type of assembly line balancing studied and their features, (2) GA specifications: addresses
information on GA methods, chromosome representation, feasibility issues, initialization, selection and survival schemes, genetic operators (crossover, mutation), fitness function, termination criteria and (3) Performance specifications: data sets used for experiment, methods compared with, computation time and implementation language [53].

Since we have already discussed the classification of assembly line balancing problems in more detail based on Boysen et al. [3] compared to the considerations in this structural framework, our discussion in this chapter will be limited to the features of different genetic algorithm approaches for ALBP based on GA specifications only.

2.5.1 GA Methods

Different methods are suggested in order to realize genetic algorithms for assembly line balancing problems including Typical Genetic Algorithm (GA), Grouping Genetic Algorithm (GGA), Parallel Genetic Algorithm (PGA), Hybrid Genetic Algorithm (HGA), Greedy Randomized Weighted Adaptive Search (GRWASP), Genetic Algorithm with dynamic partitioning and GENITOR like modified GA.

2.5.1.1 Typical Genetic Algorithm (GA)

A typical GA would be more analogous to the standard genetic algorithm [54]. Common steps in a typical standard genetic algorithm are shown as following:

\begin{itemize}
  \item \textit{Step 1:} Start \((t = 0)\)
  \item \textit{Step 2:} Generate initial population, \(P(t)\)
  \item \textit{Step 3:} Evaluate fitness of all individuals of \(P(t)\)
  \item \textit{Step 4:} Increment in \(t\) \((t = t + 1)\)
  \item \textit{Step 5:} Select parents for crossover
  \item \textit{Step 6:} Crossover
\end{itemize}
Step 7: Mutation
Step 8: Evaluate fitness and determine which offspring will survive
Step 9: New P(t)
Step 10: Check for termination criteria
Repeat (3) to (9) while termination criteria is not true
Step 11: End

2.5.1.2 Grouping Genetic Algorithm (GGA)

The Grouping Genetic Algorithm are similar to the standard genetic algorithms but very special considerations are made in this genetic algorithm. Each station is considered to contain a group of feasible tasks. For genetic operations, they are used as a group unlike the tradition approach of treating them one by one [55]. Further details are provided in the upcoming sections.

2.5.1.3 Parallel Genetic Algorithm (PGA)

Most of the literature address only serial implementation of GA for ALBP whereas, the Parallel Genetic Algorithms can enhance the performance significantly taking advantage of the multi-core processing architecture. The most common types of parallel GA models are - distributed GA and cellular GA; which are also called course grain PGA and smooth grain PGA respectively. In the case of distributed GA individual subpopulation is assigned to each available processors. Genetic operations are carried out in each processor simultaneously using conventional techniques. Occasional migration may be allowed where individuals from a population on a processor travel to a population on another processor. They may travel to some particular processors only (e.g. to immediate neighbors only) or may travel to any of processor randomly. The method and rate of this migration have to be determined for the most efficient outcome. In the case of the cellular PGA, individuals are only allowed to interact within their neighborhood. It normally deals with
one conceptual population and each of the processors usually holds one or two individuals. The good solutions appear in different areas of the solution space and eventually spread through the search space. These structured algorithms are found to be beneficial for some serial implementations as well. Another type of parallel GA model is the master-slave model, where the main algorithm is run on one processor and the fitness evaluation process is run on individual processors for individuals in the whole population [56, 57].

Anderson et al. proposed a parallel genetic algorithm based on cellular GA. However, they had a motivation to minimize the communication cost. Hence, they have suggested instead of picking two individuals each time for crossover, one of the offspring are kept and only the other one is called based on ring 8 neighborhood search [58].

2.5.1.4 Hybrid Genetic Algorithm (HGA)

A hybrid genetic algorithm can be formed combining the genetic operations with task assignment heuristics or other search techniques such as – hill climbing technique, branch and bound, simulated annealing [59], tabu search [60], ant colony optimization [61], swarm optimization [62], honey bee algorithm [63] etc.

Bautista et al. suggested Greedy Randomized Adaptive Search Procedure (GRASP) and Greedy Randomized Adaptive Weighted Search Procedure (GRWASP) where such concepts are utilized. In these methods heuristic rules are used to generate the solutions and the genetic algorithm is used for local neighborhood search. There is a range of greedy heuristics based on assignment rules such as – largest candidate, rank positional weight and more. A combination of these rules may result in a better solution, however, deciding their order and combination is not very straightforward [64].
Fathi et al. and Rashid et al. have summarized overall soft computing efforts for assembly line balancing problems [65, 66].

2.5.1.5 *Genetic Algorithm with Dynamic Partitioning*

Genetic algorithm with dynamic partitioning takes an effort to preserve part of a feasible solution (chromosome) complying with a predetermined dynamic partitioning coefficient and performing genetic operations only on rest of the chromosome which evidently reduces computational effort as well [67].

2.5.1.6 *GENITOR like modified GA*

A GENITOR like modified GA is suggested by Rubinovitz et al. The suggested algorithm contains following steps: (1) Initial population is generated by randomly constructed solutions (2) Two solutions randomly selected for crossover and Mutation (3) Decoding/Evaluation (4) Selection for next generation based on elitism, after selection if there is presence of multiples of the same solution, redundant candidates are removed. This leads to decreased number of population (5) Termination after the desired number of operations. The decoding and evaluation part have some improvement operator that seeks local improvements [68].

2.5.2 *Chromosome representation*

As featured with unique requirements to implement genetic algorithms to assembly line balancing problems, different researchers have come out with a range of schemes to represent solutions in the form of the chromosome in such a way that they are suitable for genetic operations. An overview of such methods is given in the following discussion. In
order to better represent the contrast among different encoding schemes, the following precedence graph will be used in example representation of corresponding schemes.

![Figure 6: An example precedence graph (from Mansoor-11 tasks problem) [69]](image)

### 2.5.2.1 Task based

The most common representation scheme for ALBP is task based representation. In this scheme, each gene in the chromosome represents a particular task and they are arranged in order of their assignment complying with the feasibility criteria. Evidently, another heuristic will be needed in order to determine the corresponding station for each task except for the problems where each station can hold only one task. The length of the chromosomes will be constant since it will depend on the total number of tasks to be assigned [67, 70, 71].

For example in Figure 6, since we have 11 tasks the chromosome according to this method will have 11 genes. The genes are ordered in the chromosome based on their assignment preference set by the precedence requirements and the heuristic under consideration. Let’s
consider largest candidate rule with a target cycle time of 45 units for instance, and we have the following sequence in the chromosome:

| 3 | 2 | 1 | 4 | 5 | 7 | 6 | 9 | 8 | 10 | 11 |

In order to decode this chromosome, a small heuristic will be required. The first gene (a task) will belong to the first station, A. The following gene will belong to the same station as long as the criteria, $C_T>T_s$ is not violated. Otherwise, the gene will be assigned to the next station B.

2.5.2.2 Workstation based

Another popular scheme to represent candidates of assembly line balancing problems is workstation based representation. Unlike the task based representation, the value of the genes represents the workstation. In this method, however, a locus represents the corresponding task which belongs to that station (as labeled). For example, if the ith element is j then task i is assigned to workstation j [72].

| B | B | A | C | C | C | C | D | D | E |

Each cell (locus of chromosome) on this string represents following tasks respectively:

1 2 3 4 5 6 7 8 9 10 11

Again we will need a total of 11 loci on the chromosome to represent 11 tasks in a predetermined order which is shown in the lower string for clarity. The value of a gene at a certain locus (location) represents the accommodating station for the corresponding task. The first locus has value B which means Task 1 is assigned to station B, the third locus contains C which means Task 3 is assigned to station C and likewise.
2.5.2.3 *Embryonic*

The embryonic representation is again a variant of task-based representation. In this method, the length of the chromosome is not constant. The initial length is generated randomly with only a part of the tasks to be completed, and the full length of the chromosome is formed as the evolution progresses [73, 74]. For example, the development of the chromosome over generations may be demonstrated with few snapshots as follows:

<table>
<thead>
<tr>
<th>Observation 1:</th>
<th>1</th>
<th>4</th>
<th>5</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observation 2:</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Observation 3:</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

2.5.2.4 *Group-based*

The conventional encoding where one gene is used to represent one task is mentioned to be too object (equivalent to task) oriented while could perform better if they were rather group (equivalent to station) oriented as mentioned by Falkenauer et al. In line balancing problem we are more concerned with how well balanced the stations are, rather than which task is going to which station as long as the precedence is met and unless there are any special assignment restrictions. Falkenauer et al. suggests for a new encoding scheme where an object oriented part of the chromosome just carries information regarding which task is assigned to which station and doesn’t have any part in crossover or mutation. Separated by a colon (“:”), a group (station) oriented part holds the group wise information of the tasks in stations and only this part is considered during the use of crossover and mutation operator. On the demerits, this method has to deal with individuals with different chromosome lengths [55].
Tasks accommodated by corresponding stations:

<table>
<thead>
<tr>
<th>B</th>
<th>B</th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>D</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

In the above representation the part on the left side of “:” is the object oriented part keeps track of which task belongs to which station, however, doesn’t participate in the genetic operations. The other part is a list of stations which is used for genetic operations.

2.5.2.5  Task and station vector based

The chromosome is represented in terms of two vectors: (1) an integer vector consists of activity numbers sequenced in order of their execution and (2) a vector of pointers indicating the first activity for each station. The vector of pointers divides the vector of activities into the number of stations. This scheme requires a special procedure to check for the equality of solutions in order to determine redundancy caused by feasible permutation within a station [68].

Similar to task based representation, the tasks are first sorted according to their assignment preference but the first row above is a set of vectors which indexes the first task of each station which eliminates the need for running another heuristic to interpret which task is accommodated at which station.
2.5.2.6 **Heuristic based**

This is an indirect method of representation. None of the tasks or workstations are directly represented in the chromosome rather the chromosome consists of heuristics considered for assignments. First, a vector of rules and procedures are formed and the tasks are divided into subsets of tasks, then a rule vector is assigned for each of the subsets. The chromosome length is usually same to the number of tasks to be assigned where each locus represents a task. The genes carry information on the heuristics to be used for a particular task assignment. Any of the heuristics listed in 2.2 can be used. In addition, heuristics suggested by Battacharjee and Sahu, Kilbridge and Wester etc. may also be used [64, 75].

Let’s assume a set of five heuristics: H1, H2, H3, H4 and H5 are under consideration. For the example in Figure 6, five subsets of tasks are arbitrarily generated and a heuristic is assigned H1 (2,3), H2 (1,5,10), H3 (7,8), H4 (11) and H5 (4,6,9). If the locus is considered to represent each task, the chromosome would be represented as following:

<table>
<thead>
<tr>
<th></th>
<th>H2</th>
<th>H1</th>
<th>H1</th>
<th>H5</th>
<th>H2</th>
<th>H5</th>
<th>H3</th>
<th>H3</th>
<th>H5</th>
<th>H2</th>
<th>H4</th>
</tr>
</thead>
</table>

2.5.3 **Feasibility**

Maintaining feasibility of the candidates during genetic operations is a key issue of implementing GA for assembly line balancing problems. Usually, the feasibility is determined based on two factors, i.e. conformance to the precedence requirements and ensuring a task is assigned only once in a solution.
2.5.3.1 *Forced feasibility*

In this method, no infeasible candidate is allowed in the population. The most common method is to consider a subset of those tasks only for which there are no more unassigned predecessors and the elements of that subset are reloaded during each assignment [71, 76, 77].

2.5.3.2 *Repairing*

In the repairing method again there is no infeasible candidate that appears in the population but unlike the forced feasibility candidates are allowed to form and then their infeasible portions are passed through a repairing mechanism in order to make them feasible. A common method of repair is to reorder the infeasible tasks such that the candidate converts into a feasible solution. This reordering is achieved by the use of heuristics [72, 78].

2.5.3.3 *Penalty function*

Some researchers preferred to allow infeasible solutions in the population to take advantage of any good characteristics they might possess. However, their ratio in the population compared to the feasible solution is maintained considerably lower and this is controlled by introducing a penalty function which affects their fitness which eventually challenges their survival [58, 79].

2.5.4 *Initial population*

In most of the literature, the initial population is generated randomly. However, in many cases, there was a combined use of heuristics and random generation in order to ensure better and/or feasible candidate in the initial population.
Falkenauer et al. randomly sorts the tasks which don't have a precedence restriction (i.e. predecessor already assigned) and assigns them using First Fit heuristic [55]. Leu et al. used another method where in order to generate the initial population a ‘can not precede table’ is made for each task based on the precedence requirements. Then, a task without any precedence is randomly selected. The task is assigned to a station and removed from the ‘can not precede table’. The process is repeated for following tasks [71]. For the heuristic based chromosomes, heuristics are randomly chosen for every task in order to assign tasks according to their rank [64, 80].

2.5.5 Selection

A selection strategy is required to determine the mating pool of parents who will take part in the genetic recombination process.

2.5.5.1 Roulette wheel method

The Roulette wheel method is a proportionate selection method which largely depends on the fitness of the individuals in the population. The probability of an individual to get selected is the ratio of the individual’s fitness to the sum of the finesses of all of the individuals in that population [71, 77, 79].

2.5.5.2 Stochastic universal sampling

A limitation of the Roulette wheel method is if an individual has very low fitness the probability of selection for those candidate goes too low. The Stochastic Universal Sampling (SUS) uses a fitness proportionate scale similar to Roulette wheel method which basically works as a spectrum but the selection is obtained at evenly distributed intervals.
This interval is randomly decided. This allows a relatively better chance for the poor candidates to get selected [58, 81].

2.5.5.3  *Remainder stochastic sampling*

The remainder stochastic sampling seems to be the least used method for selection in this paradigm. This is also a fitness dependent method but different than the previous two. In this method, we have to take the ratio of fitness of each individual and the average fitness of the population. If the ratio is more than one the individual is copied to the mating pool the same number of times as the integer value. Having all such candidates assigned in the mating pool if there is still space for more individuals a new scale is formed based on only remainders and selection is continued as a similar process as Roulette wheel method. Kim et al. found the stochastic universal sampling as the better-performing selection criteria than remainder stochastic sampling [72].

2.5.5.4  *Tournament method*

The tournament method doesn’t use the fitness proportionate selection within the whole population. Instead, a random permutation of the whole population is created first, which then divided into blocks of individuals according to a predetermined tournament size. From each block, the best individual is selected for the genetic operations in order to form the following generation. As an example, Kim et al. have used a tournament size of 2 [66, 82].

2.5.6  *Crossover operator*

Crossover is a vital genetic operator mostly contributing towards the local search. At this stage of the evolution process, two parents exchange some of their characteristics in order to produce offspring, which would hopefully be a better solution, although not necessarily.
2.5.6.1 Modified Bin Packing Crossover

As mentioned before Falkenauer et al. proposed a unique method of genetic algorithm for bin packaging and assembly line balancing. They proposed a modified bin packaging crossover method for bin packaging problem (BPCX) and a modified bin packaging crossover method for assembly line balancing problems (modified BPCX) which are capable of dealing with the precedence requirements and avoiding the formation of cycles in the network. Recalling from the previous discussion on grouping based chromosome representation, only the group part of the chromosome takes part; not the tasks themselves.

In this two-point crossover method, firstly, crossover positions are selected randomly on both parents. The selected section of Parent 2 is injected into the chromosome of Parent 1 at the selected position. If one or more tasks are repeated, the stations with repeated tasks are deleted. If there are tasks which were deleted and did not belong to any other station, they are reassigned using FFD heuristic [55].

With reference to the precedence graph in Figure 6, an example is presented below:

<table>
<thead>
<tr>
<th>B</th>
<th>B</th>
<th>A</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>C</th>
<th>D</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks accommodated by corresponding stations:</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>a</th>
<th>e</th>
<th>b</th>
<th>b</th>
<th>b</th>
<th>c</th>
<th>c</th>
<th>c</th>
<th>d</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks accommodated by corresponding stations:</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s assume element c from parent 2 is selected and the insertion place is between C and D. The chromosome of intermediate offspring will be:

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>C</th>
<th>c</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Task 7 and 9 is repeated in C and Task 8 is repeated in D. So C and D are deleted.

<table>
<thead>
<tr>
<th>B</th>
<th>A</th>
<th>c</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Since Task 4, 5, 6 and 10 got deleted and don’t belong in the chromosome anymore, they are reinserted using FFD heuristic. For Task 4, due to the precedence constraint and target cycle time requirement, it has to go in a new station f, making the door open for Task 5 and 6 to be assigned to the same station. Task 10 can be accommodated in station c. For the purpose of decoding, the whole chromosome of the offspring is following:

<table>
<thead>
<tr>
<th>B</th>
<th>B</th>
<th>A</th>
<th>f</th>
<th>f</th>
<th>f</th>
<th>c</th>
<th>c</th>
<th>c</th>
<th>c</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tasks accommodated by corresponding stations:

In the example above task 3 between stations E and B were common hence, B was deleted.

As a result, there is no task 4. To avoid such issues, the deleted tasks which don't belong to any station are saved and following FFD (First Fit Descending) algorithm, they are
assigned to any existing station or a new station. As in the example, task 4 is reassigned to station C.

2.5.6.2 Order-based Crossover (OX)

In Order-based Crossover method, two ends of the first parent are copied to the first offspring. Instead of inserting the middle part of the chromosome of the second parent as in conventional crossover methods, it keeps but reorders the tasks of the first parent (in the middle part) according to the order of those tasks in the chromosome of the second parent. The process is repeated for the second parent in a similar way. This way feasible offspring are ensured from feasible parents [67, 71].

2.5.6.3 Modified PartiallyMapped Crossover (Modified PMX)

Unlike order based crossover, in modified partially mapped crossover (assuming two-point crossover) a section of the chromosome is exchanged. This might lead to infeasible solution hence, a repairing strategy is required [78].
2.5.6.4 Heuristic Structural Crossover (HSX)

In this method, the line balancing problem is considered as a partitioning problem in order to be able to treat the groups as the basic units of operation. They inherit the structural crossover (SX) method by Von Laszewski [83]. In this method, a group from one of the two parents is selected and copied to an offspring. The remaining groups are copied from the other parent. On the other hand, multiple groups in terms of workstations are chosen using some heuristic based on problem-specific information in the HSX procedure which also requires an adaption step in order to cope up with violation of precedence during crossover operation. All the workstations are sorted in descending order of their difference of absolute workload deviation in parent 1 and parent 2 and a set B is formed. From the set of workstations, a section is selected at less than half of the total number of stations and a subset Q is formed (Q⊂B). Then a new offspring is formed according to following steps. If a task from parent 2 is assigned to a workstation that is an element of Q the task is assigned to the same workstation as in parent 2. Otherwise, the task is assigned to the same workstation as in parent 1.

Followed by the crossover the new offspring now goes through an adaption procedure. In order to satisfy the precedence requirements while maintaining the good characteristics introduced by genetic operations, a greedy heuristic technique based on best fit decreasing rule is introduced. The tasks in the newly formed offspring whose workstations are copied from parent 1 and also appear in Q are marked. Also, the tasks that violate the precedence requirements are marked. Previously marked items are listed as tasks to be reassigned. The number of immediate predecessors is calculated for all the tasks to be reassigned. A task from the list is chosen such that there are no immediately preceding task (in the list of tasks
to be reassigned). If there are more than one, the task with largest processing time is selected. If the tie still exists, the task is chosen randomly among the tasks are in the tie. The last workstation is identified where the immediate predecessors (doesn’t require reassignment) of our selected tasks are assigned. If there were no preceding tasks the first station is selected. The earliest among the workstations where the immediate following tasks are assigned are also identified. If there are no following tasks, the last workstation is selected. If there exist a workstation where the sum of the already assigned tasks plus task time of the task to be assigned is less the mean workstation time then the task is reassigned to the workstation with largest workstation time. Else the task is reassigned to the workstation with minimum workstation time. If there is any tie, it is broken randomly. Then the task is removed from the list of tasks to be reassigned and the procedure repeats for the rest of the tasks in the list [82].

2.5.6.5  *Uniform order-based crossover*

In the Uniform order based crossover a random binary bit string is produced with the same length as the parent’s chromosome. Genes from the locus of the parent 1 are copied to offspring 1 if the same locus is carrying a gene equals 1 on the binary string. A subset of genes from parent 1 is listed corresponding to the locus on the binary string where gene equals 0. These are reordered according to their order in the chromosome of parent 2, then they are reassigned to the empty loci on offspring 1. Another offspring is produced in the same way [77].
2.5.6.6 *Fragmented reordering crossover (FRG)*

FRG procedure is aimed to maintain inheritance of positions and relative order of elements in the structure. It provides changes within the fragment which do not violate the precedence constraints. FRG doesn’t require a reordering followed by crossover to ensure feasibility of solutions. In this method, all elements of chromosome of the parent 1 are copied to the same positions on the chromosome of the offspring 1. Then all elements of a random fragment in the offspring 1 are reallocated within the fragment according to the order in which they appeared in parent 2. Since both of the parents were the feasible solutions, the offspring will be feasible as well. The process is repeated for offspring 2 [68].

2.5.7 *Mutation operator*

The goal of a mutation operator is to allow the genetic algorithm a way out from getting trapped in local neighborhood search, thus incorporating random diversity which would prevent premature convergence of the algorithm.

2.5.7.1 *Modified bin packing mutation (modified BPM)*

The mutation operator corresponding to group-based representation and modified bin packaging crossover is modified bin packaging mutation. In order to execute the mutation operator, few randomly selected stations are eliminated. The tasks of those eliminated
stations are reassigned using FF (First Fit) algorithm. They might be assigned to an existing station or a new station following the FF. For a better chance of improvement of mutated offspring, emptiest stations are always among the eliminated ones and at least three stations are eliminated and reinserted. As a utility operator, an inversion could be executed in order to increase the chance to save some good attributes. This can be achieved by switching these genes away from either end and genes with such attributes should be staying close together. Again, it only affects the grouping part of the chromosome and doesn’t affect the fitness of the individuals [55].

2.5.7.2  **Heuristic structural mutation (HSM)**

In this method, some tasks in the chromosome are randomly selected based on mutation rate and reassigned. The reassignment procedure is same as the adaption procedure described for the HSX.

2.5.7.3  **Scramble sub-list mutation (SSM) and Scramble mutation (SM)**

In the scramble sub-list mutation two points are randomly selected and the genes at the middle section are reordered either based on heuristic or randomly (Leu et al., 1994). Similarly, for scramble mutation, only one point is randomly selected where the first part is kept unchanged and the later part is reordered [77].

2.5.7.4  **Fragmented reordering mutation (FRGm)**

Unlike other mutation strategies where the procedure dictates changing individual genes, a new method FRGm was suggested where a whole offspring from FRG crossover operator is being replaced by a randomly generated integer vector.
2.5.8 Survival

The survival scheme decides how a new population combines individuals from the previous population and new offspring.

2.5.8.1 Elitism

Elitism is the most used survival strategy as observed in the literature on this topic. In this strategy, the elitist individuals in the parent population are changed only with the offspring with better fitness. This allows having best individuals from both pools [67, 71, 72].

2.5.8.2 Parents always replaced by offspring

In this method of survival scheme, the parent population is always dissolved and the offspring is survived irrespective of their fitness. However, it allows having more diversity in each population [77].

2.6 Tabu Search

Tabu search techniques for ALB problems have gained the attention of the researchers in late 90s. The initialization process for Tabu search can be executed randomly or using some other heuristic technique. The Tabus come into action during neighborhood search. Irrespective of the search mechanism, in order to avoid cycles or repetition of the same path, the algorithm remembers a predefined number of previous moves in a Tabu list. If a match is found with the Tabu list no matter the value of the objective function, that move is forbidden. Like other search methods the algorithm generates multiple neighborhood solutions at each iteration and evaluates them. Solutions promising most improvement or leading to least deterioration is selected for the next iteration [37]. Different versions of
Tabu search in terms of search strategy, evaluation criteria etc. are suggested by Chiang et al. and Lapierre et al. [84, 85]. Tabu search has also been used to solve ALB problems with parallel stations, two-sided lines, sequence dependent tasks etc. [86-88]. Tabu search is popular for being used as a tool in more robust metaheuristics like genetic algorithms [89, 90].

2.7 Simulated Annealing

The Simulated Annealing (SA) method uses the idea of Annealing, a metal heat treatment process. In the annealing process, metals are heated to a very high temperature and cooled down slowly. The heating of metal allows the metal molecules to move freely. Following that, if the metal is cooled down at a slow rate, the movement of molecules are restricted gradually and forms more uniform microstructure with better mechanical properties than that of before the heat treatment. In the simulated annealing process, a similar set of actions are used to find a globally optimal solution. At a high temperature, the solution instances are freer to form, thus more room for exploration with the whole space. As the temperature goes down with time, the search space is more narrowed down and more focus goes into exploitation. Inevitably, the temperature at the beginning of the process is quite high and goes down with a time function,

\[ T_y = \alpha T_x \]  

Where, \( T_y \) is the temperature of current iteration, \( \alpha \) is the cooling rate, \( T_x \) is the temperature of the previous iteration.
Within two candidate solutions, the measure is taken based on their performance based on objective function. If a new solution with fitness $f(y)$ is better than the previous solution with fitness $f(x)$ the new solution is set as the solution for that instance. If the new solution is worse, the replacement new solution is decided based on following probability a.k.a. Metropolis criterion (assuming a minimization problem):

$$p(x \rightarrow y) = \begin{cases} 1, & \text{if } f(y) \leq f(x) \\ \exp \left( \frac{f(x) - f(y)}{T_y} \right), & \text{otherwise} \end{cases}$$  \hspace{1cm} (5)

The iteration process continues until the termination criteria is achieved (e.g. $T_y = T_{END}$) [91-93].

In the context of ALB problems, typically, an initial solution is generated based on a prioritized list of feasible tasks. Now as the search advances (the cooling process starts), a workstation is selected randomly and subject to a trade or transfer. A trade would result in the last task of the workstation to be exchanged with the first task of the following workstation. A transfer would occur when an exchange is not possible due to the imposed constraints. This is a one-way transfer of a task from the randomly chosen workstation to the following station. If a workstation is selected for transfer which has only one task assigned to it, would require a compression i.e. merging with the following station, in order to avoid an empty workstation. If the last workstation is selected for a transfer, a following new workstation will be created and the last task of the selected workstation will be transferred to the newly formed station. Once an entire solution is generated, the objective function value for the solution is calculated. If the performance of the newly formed solution is better than the current solution, the new solution is taken as the current solution.
If the newly formed solution is inferior to the current solution, it still can be selected based on the probability given by Metropolis criterion as discussed above. The algorithm advances as the cooling process proceeds at the given cooling rate until the termination criteria are achieved [94]. However, there are more approaches to explore solutions in the neighborhood in simulated annealing algorithms, for instance, methods suggested by Roshani et al. [95] and Hong et al. [96].

The application of simulated annealing has been observed for both of the SALB problems and GALB problems. Roshani et al. have proposed a simulated annealing method for solving ALB problems with multi-manned workstation, taking the line efficiency, line length and smoothness index as the performance criteria [95, 97]. Suresh et al. suggested a simulated annealing algorithm for stochastic assembly line balancing [98]. Simulated annealing has also been used for hybrid algorithm along with genetic algorithms, particle swarm optimization algorithms etc. [67, 99].

2.8 Ant Colony Algorithms

The Ant Colony algorithm mimics the social coordination of ants to find the shortest path to food. In the natural system, the scout ants take random paths at the beginning and leave a chemical called Pheromone on their way. Since the ant on the shortest path will come back early and then start for destination again before the other ants are back, the pheromone level will be higher on this path compared to any other path. When the next ant comes back to the nest it finds a higher pheromone level on the previous ant’s path. The second ant now has to decide which way to go. The decision is probabilistic, and the probability of selection of a path will depend on the pheromone level on the path. Essentially, over time
the difference of pheromone level will be significant and a very high pheromone level on the shorter path will have very high probability of selection by the ants. However, since the selection of paths is probabilistic, there will always be a chance for an ant selecting a different path than the one with high pheromone. The pheromones evaporate after a certain time [93].

As it sounds very familiar to shortest path problems, the Ant Colony Optimization (ACO) techniques have gained a good popularity, particularly in the field of discrete optimization. Artificial agents to replicate ants are created to look for solutions on alternative branches. At the beginning, all the possible paths have the same probability of getting selected. Each of the ants comes up with a completed path and pheromone level on each path is decided based on the frequency of the path getting used previously. The probability of selection of a path at the next run will be based on the pheromone level on that path [100].

In the context of ALB problems, an ant has to assign all of the tasks to workstations following the precedence requirements to complete a path. For type – 1 problems, an ant selects a feasible task to assign to a workstation as long as it is open (further task is still assignable) or open a new workstation if the previous workstation is closed (no further assignment is possible). As the iteration process proceeds, all ants have their assembly line layouts. Before the next run, the likelihood for a task to get selected for a certain workstation is quantified as pheromone level by the frequency of that particular assignment happening previously. The process continues until the stop criteria are achieved [101].

ACO has been used for a range of generalized assembly line balancing problems. McMullen et al. proposed an ACO algorithm to address mixed model ALB problems with
parallel workstations and stochastic task processing times [101]. Blum et al. have suggested a Beam-ACO approach which uses a hybrid version of ACO and beam search technique to solve time and space constrained type – 1 SALB problems also known as TSALBP-1 [102]. A more generalized ant colony optimization method is proposed by Bautista et al. for time and space constrained SALB problems (TSALBP) [103]. Further multi-objective ACO algorithms are discussed by Chica et al. and McMullen et al. [104, 105].

2.9 Honey Bee Algorithms

The Honey Bee Optimization Algorithms (HBA) adapt the social foraging behavior of honey bees. In the natural systems, a certain kind of bees i.e. scout bees randomly search over nearby places and looks for potential sources of nectar. Once the scouts are back to the hive they drop the collected nectar and move to a place called the dance floor, to perform waggle dance. This dance is the means of communication for the scout bees (a.k.a. the foragers) information regarding the flower patches just explored. The dance contains three information regarding the flower patches: direction, distance, and quality of nectar. Followed by the waggle dance, the follower bees follow different scout bees to different flower patches. More follower bees will be sent to the more promising sources, i.e. the flower patches those contain more nectar and takes less effort to collect them. This allows for the bees to explore a big area at the same time. The next waggle dance would let decide if a source is no longer a good choice or more bees are needed to that source. This is a continuously updating system and the bees keep the system running during the harvesting season. The honey bee algorithm starts with assigning a predefined number of scout bees randomly in the search space (randomly generated solutions) followed by evaluating their fitness. The search space explored by the scout bees with highest fitness values are chosen.
for the neighborhood search. For further search, more bees are assigned for searching near the spaces with higher fitness. It can be based on a direct ratio of the fitness value or a probabilistic selection based on the fitness value. The remaining bees are assigned randomly thus making room for further exploration [106].

Özbakır et al. suggested a fuzzy multi-objective honey bee algorithm for two balancing sided assembly line. They have used a heuristic based representation scheme where each task on the array is represented by their location and the corresponding heuristic used to assign to them is labeled in that location. The initial solutions are generated randomly. Based on their fitness, a number of better solutions are selected for neighborhood search. A Higher number of onlooker bees are assigned to these better solutions and fewer onlookers are assigned to the remaining solutions. In this algorithm, the neighborhood search is performed by swapping the heuristics of two adjacent tasks or shifting a heuristic. The probability of having a swap of shift is the same (50%). The scout bees continue to search randomly. The fitness values of bee generated solutions are compared to the original and updated if better solutions are found. At each iteration step, a new population is generated by the representatives of the selected (from neighborhood search) search space and random solutions constructed by scout bees [107].

Akpinar et al. solved a type – I mixed model assembly line problem with parallel workstations, zoning constraints, and sequence dependent setup times between tasks. They also suggested the idea of allowing honey bees to communicate within a multiple colony structure. They also looked into extending the model by adding low medium and high variability of set up times [108]. Tapkan et al. used honey bee algorithm to solve ALB
problem with positional, zoning and synchronous task constraints using fuzzy approaches with objectives of maximizing work slackness index and line efficiency and minimizing total balance delay [109]. Ozbakir et al. used the bees algorithm to solve type – I two-sided ALB problems with zoning constraints [110]. Kalayci et al. used a similar algorithm to solve a sequence dependent disassembly problem [111]. Biao et al. proposed a hybrid honey bee mating algorithm to solve two-sided mixed model ALB problems where the local search is enhanced by simulated annealing method. Instead of the above-mentioned bee foraging technique, honey bee mating algorithm uses crossover operator between the queen and the drone to explore the search space, more of a genetic like approach. A queen is the best solution in the pool and can be mated with multiple drones stochastically. The mating would result in broods which pass through a local improvement scheme afterward. If the brood is better than the hive queen, the queen is replaced by that brood. The iteration process continues until the termination criteria are achieved [63].

2.10 Bacterial Foraging

The most recent addition to the metaheuristics for solving ALB problems is the Bacterial Foraging technique [112]. The idea is rewritten based on their discussion. This technique is influenced by the foraging strategies used by the *E. coli* bacteria in the natural system. The process is completed in following three steps [113, 114]:

1. Chemotactic event or Chemotaxis: The *E. coli* bacterium uses flagella to move towards food sources with an objective of maximizing energy per unit time. The bacterium either swims or tumbles with the help of flagella in order to move. During swimming, the bacterium follows a certain path; after a tumble, its direction is
pointed towards a random direction with a bias of certain degree to have the
direction it was following before. During the foraging process, *E. coli* continuously
alters their moves. The concentration level of the nutrients dictates the duration of
swim and tumble. If the concentration is increasing bacteria spends more time
swimming and less time tumbling.

2. Reproduction event: When some of the population members consume sufficient
nutrients, they reproduce each by splitting into two at the same location. Rest of the
bacteria have not got enough nutrients in their lifetime to be healthy in order to
reproduce.

3. Elimination – dispersal event: A change in the environment that contains the
bacteria may kill a group of bacteria in some region even with a high concentration
of nutrients. This may also result in having the bacteria dispersed into a new
location irrespective of the concentration level of the nutrients.

In the optimization domain, the chemotactic event of bacterial foraging represents a search
for improvement on candidate solutions. Similar to the bacteria, the solutions can be
exploited in direction based search (swim) as long as the improvement is promising or
randomly changing its direction with a predefined bias of having the same direction
(tumble). After a predefined period of search, half of the inferior candidate solutions
(bacteria) are removed from the population. The lower the objective function value, higher
the probability of getting eliminated. The superior half are allowed to reproduce and keep
the population size consistent. In order to execute exploration, some of the randomly
selected candidate solutions are removed, and replaced by the same number of randomly
generated solutions [113-115].
For ALB problems, the process starts with randomly generated vectors. The number of vectors is equal to the maximum number of stations. Each random vector contains entries at the same number to the number of tasks. For the search purpose, the chemotactic event will take place. This event would continuously change the assignment weight of the tasks which will eventually change the line configuration and generate alternate solutions. The solutions are evaluated based on the objective function. If a newly generated solution is found to be more promising the same direction vectors are applied again. This means the search would proceed one more step in the same direction. The event continues until the predefined step count achieved. For reproduction purpose the solutions are sorted based on objective function value and the inferior half is removed from the population. The remaining populations are allowed to reproduce by making a copy of them and the population size remains constant. A predefined elimination – dispersal event probability defines how many solutions will be removed and replaced by randomly generated solution. Astagun et al. used their algorithm to solve type – 1 SALB problem and U-shaped ALB problem [112].
CHAPTER THREE:

Problem Formulation and the Multi-Objective Optimization Algorithm

3.1 Outline of the Model

In this chapter, several formulations of the assembly line problem are presented. The multi-objective version of SALBP type – 1, SALBP type – 2, and SALBP type – E are presented. As will be shown later in chapter 5, the developed software tool will facilitate building different variants of the problem. This chapter is divided into two main parts. The first part introduces the different types of objectives and problem constraints that will be mixed and matched to build several versions of the problem. The second part sheds the light on some principles related to multi-objective optimization and the non-dominated sorting genetic algorithm (NSGA II).

3.2 Nomenclature

\[ t_i \]  
Processing time of task \( i, i = \{1, ..., n\} \)

\[ S_s \]  
Set of tasks assigned to station \( s, s = \{1, ..., m\} \)

\[ T_s \]  
Station time of station \( s, (T_s = \sum_{i \in S_s} t_i) \)

\[ n \]  
Total number of tasks

\[ m \]  
Theoretical total number of stations

\[ m_R \]  
Realized total number of stations

\[ C_T \]  
Theoretical cycle time \( \left( C_T = \sum_{i=1}^{n} t_i / m \right) \)
\( C_R \)  
Realized cycle time  

\( X \)  
A line balance solution  

\( O_t(X) \)  
t-th objective function value of solution \( X \), \( t = \{1, \ldots, 6\} \)  

\( d_p \)  
Number of solutions dominate \( p \)  

\( D_p \)  
A set of solutions which are dominated by \( p \)  

\( \prec_d \)  
Crowded comparison operator  

\( N \)  
Set of tasks to be assigned, \( i \in N \)  

\( M \)  
Set of workstation under consideration  

### 3.3 Objectives

A line balancing problem can be represented as an acyclic digraph where the tasks are represented by the nodes/vertices and precedence relations are represented by the arcs/edges of the graph. We are required to assign tasks from a set of tasks \( N \) to a set of workstations \( M \) such that given objectives are achieved. A multi-objective Generalized Assembly Line Balancing model is developed with following objectives:

\( O_1 : \)  
Minimizing the number of workstations given a certain cycle time or production rate  

\( O_2 : \)  
Minimizing the cycle time/Maximize the production rate given a certain number of workstations  

\( O_3 : \)  
Maximizing the line efficiency  

\( O_4 : \)  
Minimizing the line Smoothness Index  

\( O_5 : \)  
Minimize infeasibility index in terms of cycle time  

\( O_6 : \)  
Minimize infeasibility index in terms of number of station
Objectives 5 and 6 are actually developed to help in diversifying the search space while solving the problem and will be discussed further later in the thesis. The user is given the privilege to select one or more objectives from the above mentioned list, according to the nature of the problem under consideration. The model under consideration is a serial line, deterministic single model problem and excludes aspects of assignment restrictions, machine breakdowns, alternative process routing and material handling.

Let’s define following set of constraints:

\[
\begin{align*}
\text{CX}_1: \quad & \text{All precedence requirements must be met, i.e. a task } i \text{ can only be assigned if all of its predecessors, } \forall h_i \in H_i \text{ are already assigned.} \\
\text{CX}_2: \quad & \text{Each task will be assigned to exactly one workstation only.} \\
\text{CX}_3: \quad & \text{Sum of task times in each station must be equal or less than given cycle time (i.e. } \forall s: \sum_{i \in s} t_i \leq C_T, OR, \forall s: T_s \leq C_T) \\
\text{CX}_4: \quad & \text{Allowable number of workstations is given as } m.
\end{align*}
\]

Each of the objectives is subjected to some of the above listed constrained as stated in the following discussion.

\[O_1(X) = Min(m_R)\] (6)

Selecting this objective exclusively reduces the problem to SALB Type 1 problem where the number of station \(m_R\) is to be minimized for a given cycle time \(C_T\). Conventionally, this cycle time is a constant. If a potential solution inherits at least one station where the sum of the task times belonging to that station exceeds \(C_T\), the solution is regarded as an infeasible solution. The given value of \(C_T\) a decision variable often dictated by required
production rate. We introduced a concept to measure the degree of their non-conformance and regard it as one of the objective parameters. This concept is further discussed in the formulation of O5. \( O_1 \) is subjected to constraints \( CX_1, CX_2 \) and \( CX_3 \). When \( O_1 \) is accompanied by \( O_4 \), \( CX_3 \) is relaxed.

### 3.3.2 \( O_2 \): Minimizing Cycle Time

\[
O_2(X) = Min(C_R)
\]  

(7)
Selecting this objective exclusively reduces the problem to SALB Type 2 problem. In this type of problems, the objective is to minimize the cycle time for a given number of stations. This maximal allowable number of stations often dictated by physical constraints (space, transportation etc.) and regarded as a constant. Analogous to the previous objective, if the realized number of stations \( (m_R) \) is greater than the given number of stations \( (m) \), conventionally the solution would be regarded as infeasible and often ignored. The given number of workstation \( (m) \) is a decision variable here. However, we have suggested a concept and quantified the non-conformance and minimizing this quantity is introduced as an objective. More detail is discussed in the formulation of \( O_6 \).

\( O_2 \) is subjected to constraints \( CX_1, CX_2 \) and \( CX_4 \). If \( O_2 \) is accompanied with \( O_5 \), \( CX_4 \) is relaxed.

### 3.3.3 \( O_3 \): Maximizing Line Efficiency

\[
O_3(X) = Max \left( \frac{\sum_{i=1}^{n} t_i}{m_R \cdot C_R} \right)
\]  

(8)
Selecting this objective exclusively reduces the problem to SALB Type E. Unlike the previous two, the cycle time \( (C_R) \) and the number of station \( (m_R) \) both are variable and minimized simultaneously for optimal solution. \( O_5 \) is subjected to constraints \( CX_1 \) and \( CX_2 \).
3.3.4 \( O_4: \text{Minimizing Line Smoothness Index} \)

\[
O_4(X) = \min \left( \sum_{s=1}^{m} (C_R - T_s)^2 \right)
\] (9)

Incorporating the objective of minimizing line smoothness index along with any of the above-mentioned objectives (\( O_1, O_2, O_3 \)) would make the corresponding problems into GALB problems. At this point, the problem will also become a multi-objective problem where the objectives are totally different parameters. \( O_4 \) is subjected to \( CX_1 \) and \( CX_2 \). Additionally, it may also subject to constraints \( CX_3 \) and \( CX_4 \), if accompanied with \( O_1 \) or \( O_2 \) respectively.

3.3.5 \( O_5: \text{Minimize infeasibility index in terms of cycle time} \)

\[
O_5^*(s) = \begin{cases} (T_s - C_T), & \text{if } T_s > C_T \\ 0, & \text{otherwise} \end{cases}
\]

\[
O_5(X) = \min \left( \sum_{s=1}^{m} O_5^*(s) \right)
\] (10)

This is a new concept introduced in this thesis. As mentioned before in this model the solutions are allowed to go infeasible but their degree of infeasibility would be measured and minimizing this infeasibility is an objective here. \( O_5 \) is complementary to \( O_1 \). Which means that while minimizing the number of stations (\( m_R \)) for a given value of cycle time (\( C_T \)), the constraint of (\( \forall s: T_s \leq C_T \)) would be relaxed. The violation of this constrained is quantified by \( O_5(X) \). The motivation is allowing this infeasible solution relaxes the migration of task elements among stations which help to avoid the search from getting trapped. Also, allowing a little bit of violation of constraint at some point might give a
significantly better performance. In such case, the constraint might worth compromising. O4 is subjected to constraints CX1 and CX2.

3.3.6 O6: Minimize infeasibility index in terms of number of station

\[
O_6(X) = \begin{cases} 
  \text{Min} (m_R - m), & \text{if } m_R > m \\
  0, & \text{otherwise}
\end{cases}
\]  

(11)

This objective is a complementary to O2 and has the same purpose as O5. The number of stations is a given value which puts a constraint. Allowing flexibility to this constraint helps exploration and exploitation of the search space. However, the violation is measured by \(O_6(X)\). If there is significant improvement with a little compromise in the constraint the decision makers will have the chance to review it. O6 is subjected to constraints CX1 and CX2.

3.4 Multi-Objective Optimization Algorithm

In order to address the multi-objective aspect, we are going to incorporate using the Non-Dominated Sorting in Genetic Algorithm (NSGA) [118]. In NSGA, individual objective function values are considered for evaluation as a set instead of turning them into a single value. The final optimal solution may not be one single solution but a set of solutions which are non-dominated by any other solutions in the objective space. In other words, solutions of this set would dominate all other solutions; however, within the set, they would not dominate each other. This set is called the Pareto optimal set or Pareto front. A line balancing solution \(X_1\) would considered to dominate \(X_2\) (in other word \(X_1\) would be non-dominated by \(X_2\)) if both of the following criteria are met [116]:

1. \(X_1\) is not worse than \(X_2\) in any of the objectives

   i.e. \(\forall t \in \{1,2,...,l\} : O_t(X_1) \leq O_t(X_2)\) for a minimization problem
2. $X_1$ is strictly better than $X_2$ in at least one of the objectives

i.e. $\exists t \in \{1, 2, ..., l\} : O_t(X_1) < O_t(X_2)$

The initial NSGA was introduced by Srinivas et al. in 1994 [117]. Later in 2002, an improved version NSGA II was introduced by Deb et al. [118]. NSGA II incorporates elitism and uses a fast non dominated sorting approach. Our discussion would be focused on NSGA II. There are four aspects of NSGA II as following:

1. Fast Non-dominated Sorting
2. Density estimation and crowded comparison approach
3. Main loop
4. Constraint handling

3.4.1 Fast Non dominated Sorting

The aim of this sorting is to group and identify solutions according to their domination level. Each solution in the population under consideration is compared against others to identify the domination relationships. In order to do the Fast Non-dominated Sorting, following entities are computed for each solution:

1. The number of solutions which dominates a solution $p$, known as domination count $(d_p)$.
2. A set of solutions ($D_p$) that are dominated by solution $p$.

This is obvious that for the solutions of first non-dominated set of solution $d_p = 0$, since no other solution dominates the entities of that set. Now each solution $q$ of set $D_p$ will be
visited and their domination count \((d_p)\) will be reduced by one as visited. After completing the visit, the solutions of set \(D_p\) which has \(d_p = 0\) after the reduction, will be placed in a separate set \(Q\). Set \(Q\) contains the second non-dominated front. The following fronts can be found using same procedure until all fronts are defined.

\[
\begin{align*}
\text{Table 6: A fast non-dominated sorting procedure [118]} \\
\text{Finding the domination relationships} \\
\text{Step 1: for each } p \in P \\
D_p = \emptyset, d_p = 0 \\
\text{Step 2: for each } q \in P \\
\quad \text{if } p < q \text{ then } D_p = D_p \cup \{q\} \\
\quad \text{elseif } q < p \text{ then } d_p = d_p + 1 \\
\quad \text{If } d_p = 0 \text{ then } F_1 = F_1 \cup \{p\} \\
\text{Non-dominated sorting} \\
\text{Step 3: } r = 1 \\
\text{Step 4: while } F_r \neq \emptyset \\
Q = \emptyset \\
\text{Step 5: for each } p \in F_r \\
\text{Step 6: for each } q \in D_p \\
\quad d_q = d_q - 1 \\
\quad \text{if } d_q = 0 \text{ then } q_{\text{rank}} = r + 1, Q = Q \cup \{q\} \\
\text{Step 7: } r = r + 1 \\
\text{Step 8: } F_r = Q
\end{align*}
\]

\subsection{Density estimation and crowded comparison approach}

From any evolutionary algorithms, a common expectation is that the algorithm would spread the search across solution space. This demands for maintaining diversity in the population. NSGA II uses density estimation and crowded comparison approach to maintain diversity.
Figure 7: Density estimation and crowded distance calculation

Crowding distance ($p_{distance}$) is a measure of how closely solutions are placed compared to neighboring solutions. Apparently, this is based on the average distance of a point from two neighboring solutions. Computation of crowding distance includes following stages:

1. For each objective, sort the population based on ascending order of objective value
2. For each objective, boundary solutions (with minimum and maximum objective values) crowding distance is set to infinite.
3. Other intermediate solutions are assigned a distance value as following:

$$p_{distance} = O_t(p) + \left| \frac{O_t(p + 1) - O_t(p - 1)}{O_{t}^{\text{max}} - O_{t}^{\text{min}}} \right|$$  \hspace{1cm} (12)

4. Overall crowding distance for a solution is calculated by summing the crowding distance for all of the objectives.
Having the crowding distance computed for all of the candidates, the crowded comparison can take place using crowded comparison operator \((\prec_d)\). The operator prefers a solution within a pair if:

1. A solution belongs to a front with better rank than the other.
2. A solution belongs to the same front but located in a less crowded region.

### 3.4.3 Main loop

The initial population \(P_0\) of size \(N\) is developed randomly. The solutions of initial population are sorted based on the non-domination. The solutions are assigned to ranks equal to their non-domination level (level 1, level 2, …). The first offspring population \(Q_0\) of size \(N\) is generated using traditional tournament selection, recombination and mutation operators.

In order to generate \(z\)-th population, the parent and offspring populations are combined into a population \((R_z = P_z \cup Q_z)\) of size \(N\) and sorted according to the non-domination. Only the best \(N\) solutions will be selected for next generation. The candidates belonging to the earlier ranks are selected. Since only \(N\) solutions will be selected, from the last selected front remaining number of solutions to make population size \(N\) will be selected. Since they are selected from the same front, the crowded comparison operator \((\prec_d)\) is used for selection.

*Table 7: Main loop of NSGA II [118]*

| Step 1: | Combine parent population and offspring population, \(R_z = P_z \cup Q_z\) |
| Step 2: | Execute Fast non-dominated sort of \(R_z\) |
| Step 3: | \(P_{z+1} = \emptyset\) and \(r = 1\) |
Step 4: Until $|P_{z+1}| + |F_r| \leq N$

Execute crowding distance assignment for $F_r$

$P_{z+1} = P_{z+1} \cup F_r$

$r = r + 1$

Step 5: Sort $(F_r, \prec_d)$$

Step 6: $F_r^* \subset F_r$: $F_r^*$ = set of first $(N - |P_{z+1}|)$ elements of $F_r$

$P_{z+1} = P_{z+1} \cup F_r^*$

Step 7: $Q_{z+1} = \text{Recombination}(P_{z+1})$

Step 8: $z = z + 1$

### 3.4.4 Constraints handling

Deb et al. mentioned about two approaches for handling the constraints in NSGA II.

The first approach used binary tournament selection method to handle the constraints. For a single objective model, there might be three cases:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>Both solutions feasible</td>
</tr>
<tr>
<td>Case 2</td>
<td>One feasible, other infeasible</td>
</tr>
<tr>
<td>Case 3</td>
<td>Both infeasible</td>
</tr>
</tbody>
</table>

In the case of multi-objective problems we need to define a new parameter constrained-domination as, a solution $p$ is considered to constrained-dominate other solution $q$ if:

1. Solution $p$ is feasible and $q$ is not feasible
2. Both are infeasible but $p$ has a smaller overall constraint violation
3. Both are feasible and solution $p$ dominates solution $q$
The second approach is called the Ray–Tai–Seow’s constraint handling approach [119]. In this method, a non-domination check for the constraint violation is suggested instead of adding the violation indices for all constraints. This will lead to three rankings. The first ranking would count for non-dominated sorting for all objective values. The second ranking would count for non-dominated sorting of all constraint violation values, and the third ranking would count for non-dominated sorting of combined objective values and constraint violation values. This combination would simply merge the sets and requires no penalty parameters. The combined non-dominated sorting would be expected to handle the constraints and objective values in an effective manner.
CHAPTER FOUR:

Solution Algorithm

4.1 Introduction to the Solution Algorithm

In this chapter, a novel solution algorithm using progressive modeling will be presented. Since we have multiple variants of the problem, the compiled algorithm will be problem dependent. The chapter introduces the novel representation of the current problem using a graph $G(V, E, W)$ of vertices, edges and workstations; followed by a couple of initialization algorithms for SALBP type – 1 and SALBP type – 2. Several improvement algorithms will be presented followed by the master algorithm.

4.2 Solution Representation

In this study, our solution to an assembly line problem is represented using a Graph $G(V, E, W)$ where $V$ stands for graph vertices, $E$ stands for graph edges, and $W$ stands for the newly introduced object of workstation. Every vertex is a place holder for a work element. A work element contains the basic information of task time, label or number, description and an associated list of predecessors. Using this information, a basic graph $g(V, E)$ can be developed. Our solution algorithm is responsible for creating the third component $W$ by allocating these workstations for a certain groups of tasks. Figure 8 shows the novel representation. Structure of the graph $g(V, E)$ stay same throughout the solution process. By the means of switching workstations, instances of the graph can be used to explore our search space.
This novel presentation will have many implications on how our solution algorithms work and why they will be distinguished from those that are already existing in the literature.

![Graph G(V, E, W)](image)

*Figure 8: The novel individual assembly line graph G (V, E, W)*

### 4.3 Initialization Algorithms

The proposed algorithm is a workstation-oriented algorithm but unlike the traditional forward and backward tracking approaches developed in the literature, this algorithm has the ability to start anywhere on the graph and can incorporate any priority rule to enhance the initial solution.

The first initialization algorithm (Table 9) is concerned with generation of individual solution for SALBP type – 1 while the second (Table 10) is concerned with creating individual for SALBP type – 2. For SALBP type – E, smoothness index or any additional objectives they can be handled easily using the multi-objective nature of the developed problem.
Both algorithms use an ‘available task list’ which is a list of unassigned tasks. At the beginning, all available tasks are available in a ‘task pool’ for selection. Once a task is selected and assigned to a workstation, it is removed from the ‘available task list’ and the ‘task pool’. The ‘task pool’ is updated with immediate predecessors and successors of the tasks assigned to the current station.

Table 9: Initialization algorithm # 1 (SALBP type - 1)

Step 1: Create available task list
Step 2: Update the task pool to include all the elements in the task list
While (True)
Step 3: Create a new workstation and add it to workstations list
Step 4: Select a task from the task pool randomly or using a priority rule
Step 5: Assign this task to the current workstation by adding it to its task list
Step 6: Remove it from available task list
Step 7: Update the candidate task pool by:
  o adding the immediate predecessors or successors of the tasks belong to the current workstation
  o removing the lastly assigned task from the task pool
Step 8: If the available task list is empty Exit while
Step 9: Check the cycle time and filter out those tasks that can’t be added to the workstation from the candidate task pool
Step 10: If the pool is empty, add the available task list items to the pool and go to 3, else go to 4.
Step 11: End While
Table 10: Initialization Algorithm # 2 (SALBP type - 2)

Step 1: Create available task list
Step 2: Update the task pool to include all the elements in the task list
Step 3: Create a list of workstation using the upper bound of workstations ($UB_m$)

While (True)

Step 4: Select a task from the task pool randomly or using a priority rule
Step 5: Assign this task to any valid workstation between its earliest and latest workstation.
Step 6: Update the candidate task pool by:
   a) adding the immediate predecessors or successors of the tasks belong to the current workstation
   b) removing the lastly assigned task from the task pool
Step 7: If the available task list is empty, exit while, else go to 4
Step 8: End while

4.4 Improvement Algorithms

To improve the already existing solutions we can

1. Swap any two task between two consecutive workstation
2. migrate one task from one workstation to the other
3. merge two workstations together
4. split one workstation into two separate workstation

using these algorithms might create some infeasible solutions. Objectives $O_5$ and $O_6$ discussed in the previous chapter take care of this part and the multi-objective optimization process will take care of such violations. The solutions which have $O_5$ or $O_6$ with a value equal to zero are feasible solutions.

4.5 Master Algorithm

The master algorithm is a population-based algorithm that starts with an initial population and is terminated after a certain number of iterations. The algorithm is described in Table
11. The input parameters for this algorithm are the input pop size, max number of iterations, problem data, and objectives. Please refer to 3.4.3 for more detailed steps.

*Table 11: Master algorithm based on NSGA II*

**Step 1:** Initialize to create the first parent population, $P_z$

**Step 2:** for $z = 1$ to max no. of iteration

**Step 4:** Create the offspring (improved) population, $Q_z$

**Step 5:** $R_z = P_z \cup Q_z$

**Step 6:** Use NSGAII to generate the Pareto Front, $F_r$

**Step 7:** Reduce the merged population size to $N = |P_z|$

**Step 8:** Next $z$
CHAPTER FIVE:

Software Developed and Results

5.1 Introduction to Software Interface and Benchmark Problems

In this chapter, the developed software and the output results will be demonstrated. The interactive software tool allows the user to define the problem precedence diagram, production output calculations, and list of objectives. In order to test the developed models and algorithms, a set of benchmark problems from the literature have been tested. Our tool managed to get the optimum solutions for all of the test problems which proves the validity of the developed algorithms and their effectiveness.

5.2 Input data

The first input from in this application is responsible for capturing the basic problem elements: Title, # of tasks, Task IDs, times, and predecessors. By compiling this information, the basic graph \( g(V, E) \) can be developed. Figure 9 and Figure 10 demonstrate two different versions of this form one with task description provided and one without.

The second input form is concerned with determining the cycle time or the production rate. This piece of information could be collected in many different ways as Figure 11 demonstrates.

The third input form is concerned with determining the objectives \( O_1-O_4 \). Objectives \( O_5 \) and \( O_6 \) are defined internally and both are hidden from the user. See Figure 12 for illustration.
Figure 9: Precedence diagram data without task description

Figure 10: Precedence diagram data with task description
Figure 11: Production rate and cycle time calculation form
5.3 Output Form

The output form displays the results (line efficiency, balance delay ratio, optimum number of workstations, optimum cycle time, and some other related measures of performance). Both workstation allocation \( G(V,E,W) \) and workstation loading chart are depicted. A table with all priority rules values are also presented. See Figure 13 for further details.
Figure 13: Assembly line balancing hosted in FactDesign Environment [121]
5.4 Benchmarking and Results

In order to test our model, a series of well-reported benchmark problems have been modeled and tested based on data sets presented by Scholl [120]. While selecting problems we aimed at covering problems with different sizes and complexity levels. Our software tool managed to get the optimum solution for all the problems instances developed. For the problems with a number of workstations less than 10, the optimum solutions were reached during the first iteration. We used a population size of 100 and A Pareto front size of 20 for all problem instances.

<table>
<thead>
<tr>
<th>Mertens</th>
<th>$C_T$</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>10</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$m_R$</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Jaeschke</td>
<td>$C_T$</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Jackson</td>
<td>$C_T$</td>
<td>7</td>
<td>9</td>
<td>10</td>
<td>13</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Mitchell</td>
<td>$C_T$</td>
<td>14</td>
<td>15</td>
<td>21</td>
<td>26</td>
<td>35</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Heskiaoff</td>
<td>$C_T$</td>
<td>138</td>
<td>205</td>
<td>216</td>
<td>256</td>
<td>324</td>
<td>342</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Sawyer</td>
<td>$C_T$</td>
<td>25</td>
<td>27</td>
<td>30</td>
<td>33</td>
<td>36</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Kilbridge</td>
<td>$C_T$</td>
<td>56</td>
<td>57</td>
<td>62</td>
<td>69</td>
<td>79</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Tonge</td>
<td>$C_T$</td>
<td>160</td>
<td>168</td>
<td>176</td>
<td>185</td>
<td>195</td>
<td>207</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>Arcus1</td>
<td>$C_T$</td>
<td>5048</td>
<td>5408</td>
<td>5824</td>
<td>5853</td>
<td>6309</td>
<td>6842</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>Arcus2</td>
<td>$C_T$</td>
<td>5755</td>
<td>5785</td>
<td>6016</td>
<td>6267</td>
<td>6540</td>
<td>6837</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>27</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>$C_T$</td>
<td>9400</td>
<td>10027</td>
<td>10743</td>
<td>11378</td>
<td>11570</td>
<td>17067</td>
</tr>
<tr>
<td></td>
<td>$m_R$</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>9</td>
</tr>
</tbody>
</table>
CHAPTER SIX:

Conclusions and Future Research

The assembly line balancing problem (ALBP) aims at maximizing the production efficiency by means of a minimizing number of workstations and/or minimizing cycle time. In this thesis, a multi-objective generalized model has been presented. A novel solution algorithm that introduces an extended graph notion of $G(V, E, W)$ have been presented for the first time. The non-dominated sorting algorithm has been adopted to handle the multi-objective aspect of the problem. A new software tool has been developed to demonstrate the results. A long list of well-known benchmark problems has been utilized to test and validate the developed models and algorithms. The main contributions of this thesis can be described as follows:

1. Modeling and optimization of several multi-objective variants of the problem (ALB type – 1, ALB type – 2, ALB type – E).
2. Introducing a unique solution representation scheme using an extended graph notion $G(V, E, W)$. A novel algorithm complies with such representation is developed.
3. The proposed algorithm eliminates a lot of computational overhead of developing ordered lists, repairing mechanisms, encoding and decoding operations when compared to the well famous metaheuristics such genetic algorithms and ant colony optimization.
4. Adoption of the robust and relatively fast NSGA II in order to address the multi-objective aspects of the problem.
5. Infeasible solutions are allowed to generate in the algorithm. However, minimizing the degree of violation is included as a minimization objective. This provides some flexibility for the improvement algorithm while exploring the search space (i.e. less chance of getting trapped). Also, the user gets a chance to decide on minor adjustment of the constraints as a trade-off for overall better performance.

The current research can be extended in the following ways:

1. The novel graph representation can handle any formulation of the problem. A wide range of task and workstation constraints are listed in Chapter 1.3.2 and 1.3.3 which can extend the problem formulation and make it more practical.

2. The algorithm developed can be easily extended to handle both the mixed and multi-model versions of the problem.

3. Taking advantage of simulation tools, stochastic task time can be modeled and optimized.

4. The simple assembly line balancing for a problem size of less than 200 tasks could be considered easily solvable. All optimum solution could be easily reached, thanks to higly constrained structure of the problem which contributes dramatically to get those solutions. There is a trend now in the literature to test newer algorithms with larger problems of 1000 tasks or more. We are going to consider those problems in the near future.
References


