An efficient genetic algorithm application in assembly line balancing

UniS

Thesis Submitted to the University of Surrey
for the Degree of
Doctor of Philosophy

by

Duminda Thilakawardana

School of Engineering
University of Surrey
Guildford, Surrey GU2 7XH
England

August 2002

To my parents, my wife, Shyamalie and my sons, Janith and Chanuth

This candle is for you, the beginner.
Turn it into a flame.

Samuel Taylor Coleridge, *Table Talk*

ABSTRACT

The main achievement of this research is the development of a genetic algorithm model as a solution approach to the single model assembly line balancing problem (SMALBP), considered a difficult combinatorial optimisation problem. This is accomplished by developing a genetic algorithm with a new fitness function and genetic operators.

The novel fitness function is based on a new front-loading concept capable of yielding substantially improved and sometimes optimum solutions for the SMALBP. The new genetic operators include a modified selection technique, moving crossover point technique, rank positional weight based repair method and dynamic mutation technique. The moving crossover point technique addressed the issue of propagating best attributes from parents to offspring and also supports the forward loading process. The new selection technique was developed by modifying the original rank-based selection scheme. This eliminates the high selective pressure associate with the original rank-based technique. Furthermore, the modified selection technique allows the algorithm to run long enough, if required, without premature convergence and this feature is very useful for balancing more complex real world problems. The repair technique included in this model repairs a higher proportion of distorted chromosomes after crossover than previous methods. Moreover, a third innovative feature, a moving adjacent mutation technique, strengthens the forward loading procedure and accelerates convergence.

The performance of the front-loading fitness function currently outperforms the published fitness functions and fifty-four published test cases generated from sixteen precedence networks are used to assess the overall performance of the model. Encompassing the new genetic algorithm concepts, forty-four test problems (81%) achieved the best solutions obtained by published techniques and twenty-four problems (44%) produced better results than the benchmark Hoffmann precedence procedure, the closest non-genetic algorithm method. The superiority of the genetic model over other heuristics is identified in this research and future developments of this genetic algorithm application for assembly line balancing problems is evident.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to many persons for their stimulating interactions and discussions throughout this study.

Firstly, I would like to sincerely thank Dr John Driscoll for providing me with the opportunity of doing a PhD in the School of Engineering, University of Surrey. Again, many thanks to my academic supervisors Dr John Driscoll and Dr Graham Deacon for their constructive guidance, discussions, and support throughout this study.

The financial support and assistance from the School of Engineering is gratefully acknowledged.

I wish to thank Mrs. Shirley Hankers, the Departmental Secretary for her continuous support in all aspects of my stay here.

Finally but not least, I would like to express my sincere gratitude to my wife, Shyamalie and my son, Janith for being so supportive while I engaged in this study.

LIST OF PUBLICATIONS

1. Thilakawardana D Driscoll J and Deacon G, A new assembly line balancing technique - a modified version of the Hoffmann procedure. *19th International Manufacturing Conference,* Belfast, Ireland, August 2002.557-566.

2. Thilakawardana D Driscoll J and Deacon G, An efficient genetic algorithm application in assembly line balancing. Submitted to the *European Journal of Production Research,* Special issue on assembly and transfer line balancing, 2002.

3. Driscoll J and Thilakawardana D, The definition of assembly line balancing difficulty and evaluation of balance solution quality. *International Journal of Robotics and Computer Integrated Manufacturing,* 17, 81-86, 2001.

4. Driscoll J and Thilakawardana D and Deacon G, Cost function experimentation in genetic algorithm line balancing. *16th International Conference on Production Research,* Prague, Czech Republic, August 2001.

5. Driscoll J and Thilakawardana D, Mixed model line scenarios using integrated software approach. *16th International Conference on Production Research,* Prague, Czech Republic, August 2001.

6. Driscoll J and Thilakawardana D, Variable size multi-manned station assembly line balancing analysis. *11th International Flexible Automation and Intelligent Manufacturing,* Dublin, Ireland, 2001.

7. Driscoll J and Thilakawardana D, Definition and evaluation of assembly line solution. *10th International Conference on Flexible Automation and Intelligent Manufacturing,* Maryland, U.S.A., 2000.1157-1166.

8. Driscoll J and Thilakawardana D, Using quantitative parameters to guide assembly line balancing. *International Conference on Production Research,* AIT, Bangkok, 2000.

ACRONYMS AND NOTATION

Acronyms

TS	Total Slack time
FR	Flexibility Ratio
WR	West Ratio
TI	Time Interval
OS	Order Strength
LE	Line Efficiency
BE	Balance Efficiency
BD	Balance Delay
SI	Smoothness Index
TV	Time Variability ratio
NP	Non deterministic Polynomial
RPWT	Rank Positional Weight Technique
COMSOAL	COmputer Method of Sequencing Operations for Assembly Lines
NULISP	Nottingham University Line Sequencing Program
MUST	MUltiple Solution Technique
IUFFD	Immediately Updated First Fit Decreasing
BRPW	Backward Recursive Positional Weight
FFD	First Fit Decreasing
DP	Dynamic Program
B&B	Branch-and-Bound
WC	Work Content
QS	Quant System
GA	Genetic Algorithm
PMX	Partially Mapped Crossover
ORD	ORder Crossover
CYC	CYclic Crossover
FRG	FRaGment reordering crossover
HSX	Heuristic Structural crossover
TPC	Two Point Crossover
SPC	Single Point Crossover
POS	POSitional Based crossover
UNI	UNIform Crossover
SX	Structural crossover
DPA	Dynamic PArtitioning crossover
HSM	Heuristic Structural Mutation
FBMCP	Fixed Boundary Moving Crossover Point
VBMCP	Variable Boundary Moving Crossover Point
GUI	Graphic User Interface
RAM	RAndom Mutation
ADM	ADjacent Mutation
FBRM	Fixed Boundary Random Mutation
VBRM	Variable Boundary Random Mutation
FBAM	Fixed Boundary Adjacent Mutation

VBAM	Variable Boundary Adjacent Mutation
CSR	Crossover Span Ratio

General notation

	Set of natural numbers (={1, 2, 3, …})
	Set of real numbers
$i \in I$	i is element of the set I
$I \subseteq J$	I is a subset of J
$\lvert x \rvert^{+}$	Smallest integer larger or equal to x

Notation for Assembly Line Balancing

T	Total work content
C	Cycle time
TS	Total slack time
m	Number of workstations
S_j	Station time
t_i	Task time
n	Number of elements
m^*	Theoretical minimum number of workstations
m_{max}	Maximum number of workstations
m^b	Minimum number of workstations achieved by any balancing technique
r	Number of arks in the precedence network
P	Total number of precedence relationships
t_{mim}	Minimum task time
t_{max}	Maximum task time
$P_{jt}I$	Project index
c_p	Number of precedence columns
c_{pav}	Average column position
t_{av}	Average task time
t_{sd}	Task time variance
S_{av}	Average workstation time
S_{max}	Maximum workstation time
t_k	Assigned element time
t_s	Slack time
V	Number of precedence violations

Notation for Genetic Algorithm

$d_a, e_a, f_a, k_a, k_b, k, h, I$	fitness function constants
S_{max}	Slowest station time
S_{max2}	Second slowest station time
$N_{v'}$	Number of precedence violations
z_I	Mean squared idle time

z_2	Mean idle time
$\bar{c}_k$	Fuzzy cycle time
$\bar{s}_{jk}$	Fuzzy completion time
SN_j	Number of connected networks
σ	Confidence coefficient
$\bar{S}_{mean}$	Sum of station time means
$\bar{S}_{\mathrm{var}}$	Sum of station time variance
P_j	Probability of a station not exceeding cycle time
RN_i	Non negative random integer
N_r	Random number between zero and one
e_i	i^{th} Gene (element) in the chromosome
e_j	Elements succeeding e_i
n_p	Number of elements immediately succeeding element e_j
f_i	Fitness function for infeasible chromosomes
f_f	Fitness function for feasible chromosomes
FF	Overall Fitness Function
X, Y	Components of front-loading fitness function
Z, E	Components of front-loading fitness function
R	Front-loading constant
K	Workstation shifter
l	Number of feasible links
P	Total number of precedence links
m_i	Number of workstations in the initial solution
α,β	Fitness function constants
G_w	Number of generations permitted per workstation
g	Number of generations
ΔZ	Total change of fitness
N_i	Number of identical chromosomes in the selection pool
N_{pop}	Population size
N_{POPF}	Feeding population size
n_i	Number of iterations (selection technique)
P_c	Crossover probability
s_n	Number of crossover spans
I_o	Crossover span overlapping index
c_s	Crossover span size
c_L	Left crossover boundary (for FBMCP)
c_R	Right crossover boundary (for FBMCP)
g_n	Number of overlapping genes(for FBMCP)
G_D	Number of generations per span
G	Total number of generations
s_i	Span index
G_L	Lower generation class boundary
G_U	Upper generation class boundary
E_j	Number of genes in j^{th} workstation
$c_{L'}$	Left crossover boundary (for VBMCP)
P_m	Mutation probability
m_s	Mutation span

b	Mutation span constant
N_m	Number of mutation points
m_L	Left mutation span boundary
m_R	Right mutation span boundary
pw_k	Positional weights of kth element(gene)
N_d	Number of duplicated elements (repair technique)
N_e	Number of elite chromosomes
N_b	Number of best chromosomes in the population

CHAPTER

INTRODUCTION

An assembly line is an industrial arrangement of workers, equipment and machinery for putting together components in a continuous flow process. It is generally used for mass production of a wide range of consumer goods. Assembly lines come in different shapes and sizes depending on the product being assembled. Consumer goods including furniture, electronic items, toys, 'white goods' such as freezers, washing machines, dryers and the most substantial, automobiles are a few examples. Aircraft-manufacturing assembly lines are the most complex assembly lines as far as the number of components involved in the assembly process, whereas automotive assembly lines are the most difficult lines to balance because of small cycle times (less than one minute).

Most cars manufactured today are a combination of start-stop lines during body build (robot lines) and continuous movement for final assembly (figure 1.1). A modern car manufacturing line produces around 400,000 cars annually, and costs approximately £110 millions. Table 1.1 shows the annual production and assembly labour cost of a few leading car manufactures in 1997 and indicates that the cost of direct labour involving in automobile manufacturing lines is enormous. Nevins and Whitney (1980) found that the direct labour cost of assembly was about 10-30 % of the total cost and replacing human operators with advanced automatic assembly lines is not cost effective due to huge capital investment. Therefore, the availability of relatively inexpensive and unskilled labour is still an attractive management choice for most assembly lines.

Figure 1.1. Passenger car assembly line
(http//www.autofacts.com)

	Toyota	Nissan	Honda	Ford	Daimler Chrysler	General Motors
Annual Volume (in thousands)	647	309	695	4,299	2,906	4,946
Labour hours per vehicle (assembly, stamping)	30.38	30.76	30.84	34.79	44.25	45.60
Total labour cost per vehicle (£)	1,063	1,077	1,079	1,556	1,991	2,052
Annual labour cost (£ million)	687	332	749	6,689	5,785	10,149

Table 1.1. Annual production and labour cost
(http://www.ai-online.com: The Detroit News)

In manual or semi automatic assembly lines, every manufacturer's prime intention is to make the most out of the line in terms of least overall assembly cost. The most effective way of saving millions of pounds is to decrease the number of workstations in the line. According to the figures issued by the Toyota Corporation, Japan, in 1977, the average saving per workstation per annum is £0.5 million! This implies, the fewer the operators in the line the lower the labour cost and the less the space required, giving a more cost effective production plan.

Generally, the best line length is achieved by grouping tasks into workstations along the production line and the grouping problem is known as the *assembly line problem*. This comprises two separate sub problems, the line *length problem* and the *cycle time problem,* which are solved sequentially.

The cycle time problem is minimising the cycle time when the number of workstations or production employees is fixed. This will maximise the production rate and generally occurs, when the organisation wants to produce the optimum number of items using a fixed number of workstations without purchasing new machines or without expansion.

The line-length problem is minimising the number of workstations when the required production rate, assembly tasks, task times and precedence requirements are given. This results in cost effective production plans and generally occurs when designing new assembly lines to achieve the forecast demand.

The line-length problem is more common than the cycle time problem. A large number of techniques have been developed since 1954 to solve this problem and, these techniques can be broadly categorized into three groups: exact algorithms, heuristic procedure and metaheuristic approaches. Exact algorithms become intractable when the problem size is large. Although heuristic procedures do not guarantee optimal solutions, they provide good near optimal solutions. Therefore, they have become increasingly popular among both practitioners and researchers in the late 60s and onwards.

The assembly line balancing problem was identified as an NP-hard combinatorial optimisation problem that cannot be solved in traditional ways. In 1975, John Holland introduced a simple but powerful metaheuristic technique called the '*Genetic Algorithm*' which has become more popular among researchers because of its excellent capability of addressing hard class combinatorial optimisation problems. This dissertation extends the use of the genetic algorithm in the area of NP hard line balancing.

The fist application of the genetic algorithm to the assembly line balancing was reported by Anderson and Ferries (1990) and has been followed by a number of procedures. Interestingly, none of the genetic algorithm methods were able to outperform solutions obtained by the benchmark Hoffman matrix procedure developed by Thomas Hoffmann in 1963.

Within this thesis a new genetic algorithm model is created to generate optimal or near optimal solutions that outperform the Hoffmann precedence matrix approach. This will be accomplished by modelling a new genetic algorithm line-balancing model, which includes a brand new fitness function, crossover and mutation techniques plus a modified selection scheme.

This thesis is divided in to seven chapters: Chapter 2 covers two main sections, firstly a detailed description of basic types of assembly lines; its complexity and standard performance measures, and previously published exact line-balancing heuristic and techniques. Secondly, the definitions and a conceptual framework of the classic genetic algorithm plus published applications of the algorithm to the assembly line balancing problem are given.

Chapter 3 starts with introducing the new frontloading theorem on which the novel fitness function is based. The design concepts of the fitness function and the theoretical proof are described in detail. The new crossover and mutation techniques with moving locus characteristics, plus the modified rank selection scheme are explained here.

The test programmes, the selected benchmark problems and performance evaluation criteria are the main theme of Chapter 4. Chapter 5 is the critical chapter representing the test results and discussion. Chapter 6 and Chapter 7 contain conclusions and future research directions respectively.

CHAPTER

LITERATURE REVIEW

This chapter will review the literature relevant to assembly lines and the Genetic Algorithm. Section one comprises five parts, with the first two introducing the origins and classifications of assembly lines. The third and forth parts present the complexity of assembly lines and their performance measures respectively. The last part reviews published line-balancing techniques. Section two mainly discusses the concepts of the Genetic Algorithm and its previous applications to the assembly line balancing problem.

2.1 ASSEMBLY LINE MANUFACTURE

2.1.1 THE ASSEMBLY LNE AND ITS ORIGIN

Assembly lines are an important class of manufacturing systems when large quantities of identical or similar products are to be made. The basic concept behind assembly lines is to break down the assembly process into individual stages and to allocate each stage to an operator, group of operators or a machine. A product passes along the line from stage to stage, reaching the end of the line fully assembled.

The assembly line was a vital development in the growth of U.S. industry in the first half of the 20th century and is still important today in manufacture of assembled products including automobiles, consumer electronics products, kitchen and laundry appliances (white goods), power tools and other discrete products made in large quantities.

Manual assembly lines are based largely on two fundamental work principles. The first is division of labour, argued by Adam Smith in England in his book *Wealth of Nations* published in 1776 (Smith and Sunderland, 1982). The second principle is interchangeable parts based on the work of Eli Whitney and others at the beginning of the 19th century.

Modern production lines can be traced back to the meat packing industry in Chicago, Illinois, and Cincinnati, Ohio, where overhead (un-powered) conveyors were used to move carcasses from one worker to the next. These conveyors were later replaced by powered chain conveyors to create "*disassembly lines*" – the predecessor to the assembly line.

American automotive industrialist Henry Ford observed the meatpacking industry. Together with colleagues, he designed an assembly line in 1913 in Highland Park, Michigan, for producing magneto flywheels. Ford later applied the assembly line technique to chassis fabrication and productivity was increased by a factor of eight, compared to previous single-station assembly methods. The success of the Ford Motor Company resulted in a drastic reduction in the cost of the model T Ford, the principal product of the company at the time. This forced his competitors and suppliers to imitate his method, and the manual assembly line became intrinsic to U.S. industry.

2.1.2 MANUAL ASSEMBLY LINES

A manual assembly line consists of multiple workstations arranged sequentially, at which human assembly workers perform operations (figure 2.1). The usual procedure on a manual line begins with the launching of a base part onto the front end of the line. A work carrier is often required to hold the part during its movement along the line. The base part travels through each workstation, where workers perform tasks that progressively build the product. Components are added to the base part at each station so that the entire work content has been completed when the product exits the final station.

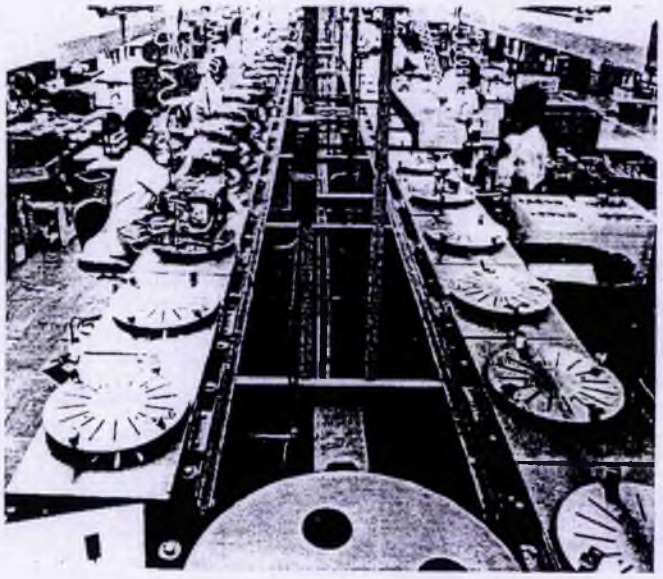


Figure 2.1. Manual assembly line in which the work proceeds around a loop (Courtesy of Jervis B. Webb Co.)

2.1.2.1 MODEL VARIATION

Today assembly lines are used to manufacture a large variety of products in diverse industrial environments. These products are different in shape, size and complexity; therefore, various production lines are used accordingly. In terms of the capacity of a production line to cope with model variations, three types of line can be distinguished: *single model lines*, *multi model lines* and *mixed model lines*. Figure 2.2 illustrates the basic types of assembly lines used in industry.

1 *Single-model lines*. These are specialized lines dedicated to the production of a single model or product. All workstations repeatedly have to perform the same tasks on identical work pieces. The workloads of all workstations remain constant over time.

2 *Multi-model lines*. These lines are used for the production of two or more models. Each model is produced in batches on the line. The models or products are usually similar in the sense of requiring a similar sequence of processing or assembly operations. It is for this reason that the same line can

be used to produce the various models with rearrangement of the line equipment. Line speed may be variable between models.

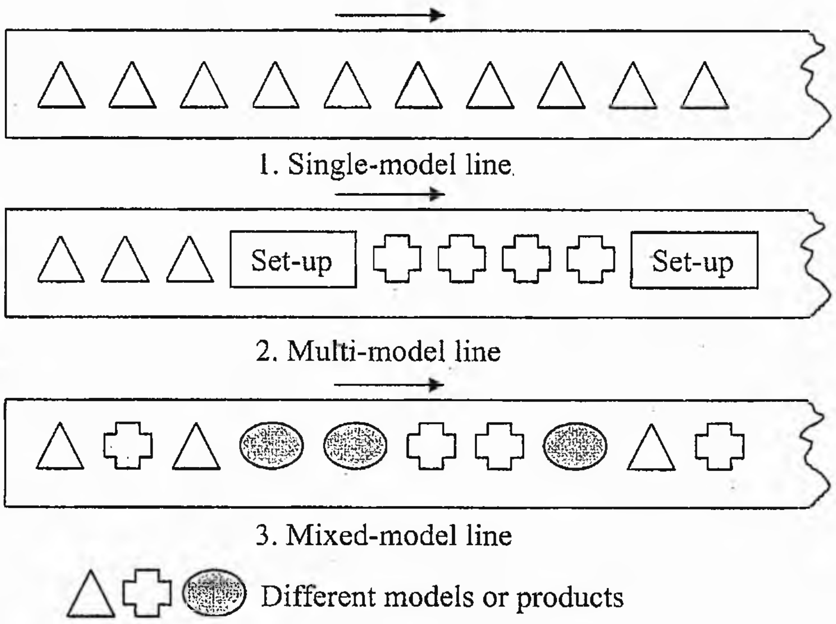


Figure 2.2. Assembly lines for single and multiple products

3 *Mixed-model lines.* They are also used for the production of two or more models, but the various models are intermixed on the line so that several different models are being produced simultaneously rather than in batches. Automobile and truck assembly lines are examples of this case.

2.1.3 BASIC TERMS OF ASSEMBLY LINE MANUFACTURING

The design and operation of assembly lines makes use of a number of specialised management terms. It is well worth defining these terms before the review of existing work and development of the balancing techniques.

1. *Assembly.* Is the process of collecting and joining two or more parts together in order to produce assembly or finished product (end product). It is characterized by the components used and the operation necessary to combine them. Components can be subdivided into *purchased items* and *sub-assemblies*

(intermediate goods). The relationships between components and the flow of material can be visualized by *assembly charts* (Buffa, 1983, p.196). The unfinished units of the product are called *work-pieces*.

2. *Operation.* Is a portion of the total work content in an assembly process. The time necessary to complete an operation is called *operation (task) time*. Operations are indivisible, which means they cannot be further subdivided without creating unnecessary additional work (Barnes, 1980).

3. *Workstation.* A workstation is an assigned location in the assembly line where a given amount of work is performed. Its dimensions, the machinery and equipment used, as well as the type of work assigned to it, characterize a workstation. One *human operator* generally mans an assembly line workstation, however, on short runs an operator may man more than one workstation. In lines manufacturing large products (e.g. aircrafts), workstations are frequently manned by several operators. The work content of a station (set of assigned tasks) is referred to as *station load*; the time necessary to perform the work is called *workstation time*.

4. *Cycle time.* Is the time the product spends at each workstation on the line when the line is moving at a standard pace. The cycle time is therefore the amount of time elapsing between arrival of successive units as they moved down the line at a standard pace. Extending this definition, the cycle time is the maximum operation time for closed stations. The pace at which the line operates (line speed) and the cycle time, together determines the rate at which products flow from the line. A positive difference between the cycle time and the workstation time is called *idle time*. The sum of idle times for all workstations of the line is known as *balance delay*.

5. *Precedence diagram.* A graphical description of any ordering in which work elements must be performed in achieving the total assembly of the product. Prenting and Battaglin (1964) introduced the precedence diagram (precedence graph) and figure 2.3 displays a typical precedence diagram of a 10-element

assembly line balancing problem. The circles represent task elements and the corresponding operation times are denoted as node weights. In this example, task A must be performed before tasks B and C and tasks D and E prior to task F etc.

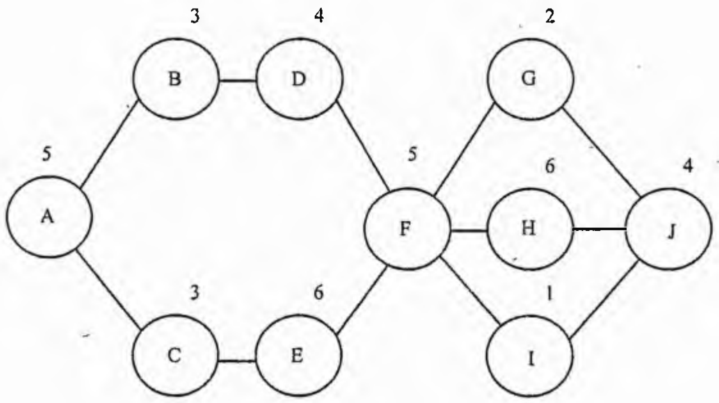


Figure 2.3. Precedence diagram

6. *Precedence matrix.* This is a square matrix, containing ones and zeros, in which rows are labelled with consecutive letters (or numbers) and the columns are labelled in the same order. Entries in the matrix are as follows:
   1. If the task element of row *i* immediately precedes the element of column *j*, a one (1) is placed in row *i* and column *j*.
   2. All other entries are zero.

j

i

	A	B	C	D	E	F	G	H	I	J
A	0	1	1	0	0	0	0	0	0	0
B	0	0	0	1	0	0	0	0	0	0
C	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	1	0	0	0	0
F	0	0	0	0	0	0	1	1	1	0
G	0	0	0	0	0	0	0	0	0	1
H	0	0	0	0	0	0	0	0	0	1
I	0	0	0	0	0	0	0	0	0	1
J	0	0	0	0	0	0	0	0	0	0

Figure 2.4. Precedence matrix

Figure 2.4 shows the corresponding precedence matrix of the above precedence diagram given in figure 2.3.

7. *Line balancing*. Is the assignment of task elements to workstations in order to satisfy a given set of objectives within precedence and cycle time constraints. Typical objectives are to minimise the number of stations and maximise efficiency.

2.1.4 CLASSIFICATION OF ASSEMBLY LINE BALANCING PROBLEMS

Research in assembly line balancing has been in existence since 1954. When Bryton (1954) pioneered the original study of the assembly line balancing problem. Over the last fifty years a large volume of literature has been published on this topic. Gagnon and Ghosh (1991) studied the historical growth and decay pattern (life cycle) of assembly line research publications and concluded that the number of publications is still on the increase.

According to the Ghosh and Gagnon (1989) classification, the Assembly Line Balancing (ALB) literature can be classified into two main groups: single model and multi/mixed model. Each group can be further divided into categories, such as deterministic and stochastic. Finally each category may be further sub-divided into two classes: simple and general (figure 2.5). Scholl (1999) presented the latest classification of assembly line balancing problems, taking into account characteristics including line speed and inventory buffers.

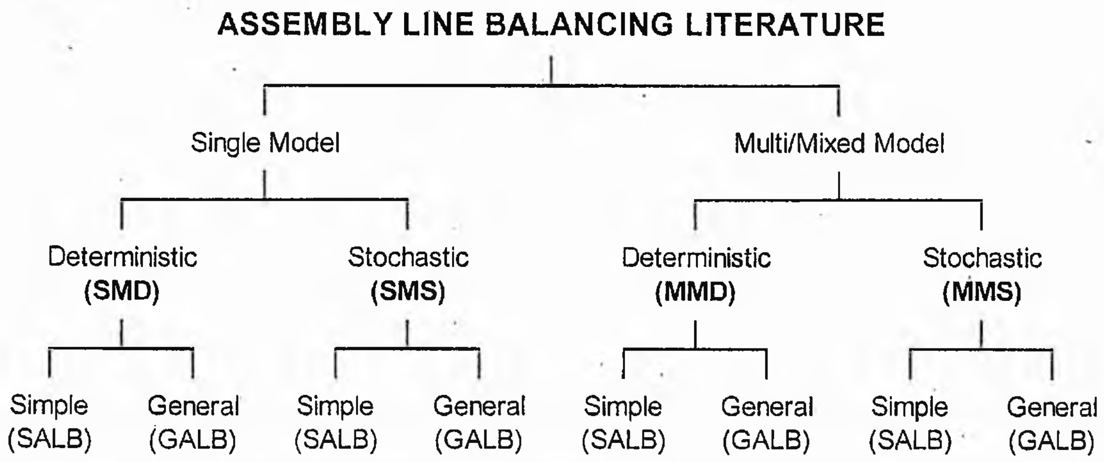


Figure 2.5. Classification of assembly line balancing literature (Ghosh and Gagnon, 1989)

The Single Model Deterministic (SMD) version of the assembly line balancing problem is the original and the simplest category of line-balancing problem. Introducing other restrictions including parallel workstations and zoning restrictions in to the model converts the problem into a General Assembly Line Balancing (GALB) problem.

The Single Model Stochastic (SMS) category introduces the concept of task-time variability. This is more realistic for manual assembly lines, where workers' operation times are seldom constant. Introducing stochastic task times cause many other issues to become relevant, including station time exceeding the cycle time, pacing effects on work's operation times, station length, size and location of inventory buffer and launch rate.

The Multi/Mixed model Deterministic (MMD) category assumes deterministic task times, but introduces the concept of the assembly line producing multiple products. Model selection, model sequencing, launching rate(s) and model lot sizes become more critical issues in this category of models.

The Multi/Mixed Stochastic (MMS) category respectively differs from its MMD counterpart in that stochastic times are allowed. All parameters arising from stochasticity that are relevant in the SMS problem are also present here. This category includes the most complicated assembly line balancing problems and not much research has been done so far in this area.

2.1.5 THE SIMPLE ASSEMBLY LINE BALANCING PROBLEM (SALBP)

The simple assembly line balancing problem has been intensively studied during the last five decades and the classical single model problem contains the following main characteristics:

1. Mass production of one homogeneous product by performing n operations over a set of workstations;
2. Paced line with fixed cycle time;
3. Deterministic operation times;

4. No assignment restrictions besides the precedence constraints;
5. Serial line layout, one –sided stations;
6. All stations are equally equipped with respect to machines and workers;
7. Fixed rate launching, launch interval equals cycle time;

The two most important parameters in production line design are the output rate and the cost of obtaining the output rate. It is possible to maximize the output rate of a production line at a fixed cost by changing the line configuration.

A substantial amount of research has been concerned with production line configurations, especially with the configuration along the length of the line (number of workstations). The classic study of this is called *assembly line balancing*. In general, assembly line balancing concerns allocating an equal amount of work to each station. Lack of such balance leads to a certain amount of line inefficiency, consequently a loss of production output, time wastage and increase in production costs (Chow, 1990, Downey and Leonard, 1992).

According to both Tonge (1961) and Prenting and Thomopoulos (1974), Bryton (1954) was the first to give an analytical statement of the assembly line balancing problem. However, the first published line balancing work was by Salveson (1955). In his pioneering work, Salveson noted that *Total Slack (TS)* is a function of the number of workstations m along the line. So,

$$\begin{aligned} TS(m) &= \sum_{j=1}^{m}(C - S_j) \\ &= mC - \sum_{i=1}^{n} t_i \\ &= mC - T \qquad (2.1) \end{aligned}$$

Where C is the cycle time, n is the number of tasks and S_j is the workstation time. T is the total work content and is a problem dependent constant. Since both C and T are constants, TS is a linear function of m, and therefore, it is minimized if the number of workstations along the line is minimized.

Also note that

$$\left[\frac{T}{C}\right]^{+} \leq m^{*} \leq n$$

Where m^* is the optimal (minimum) number of workstations needed and $[x]^+$ is the smallest integer larger than or equal to x. These bounds are referred as *theoretical minimum* (m_{min}) and *maximum* (m_{max}), respectively, of the number of stations required.

$$m_{\min} \geq \left[\frac{T}{C}\right] \quad \text{and} \quad m_{\max} \leq n$$

Therefore, perfect balance, as shown in figure 2.6, is rarely achieved in practice. A more common situation is that neither the workstation times are balanced (figure 2.7) nor is the maximum workstation time equal to the desired cycle time (figure 2.8). Generally, imperfect balance (figures 2.7 and 2.8) arises mainly due to two constraints to be taken in to account in assembly lines: *cycle time constraints* and *precedence constraints*.

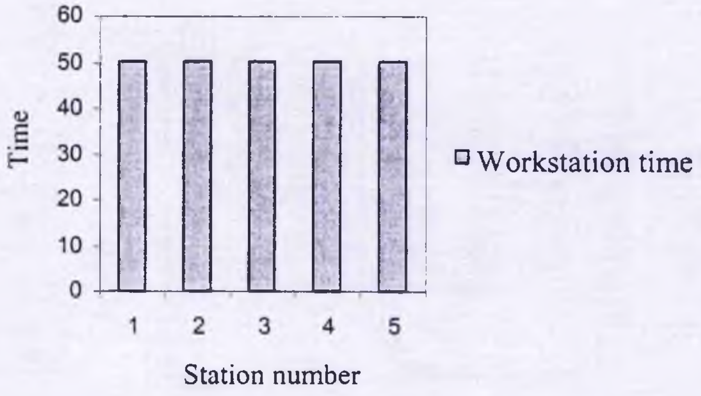


Figure 2.6. Perfectly balanced five-workstation assembly line

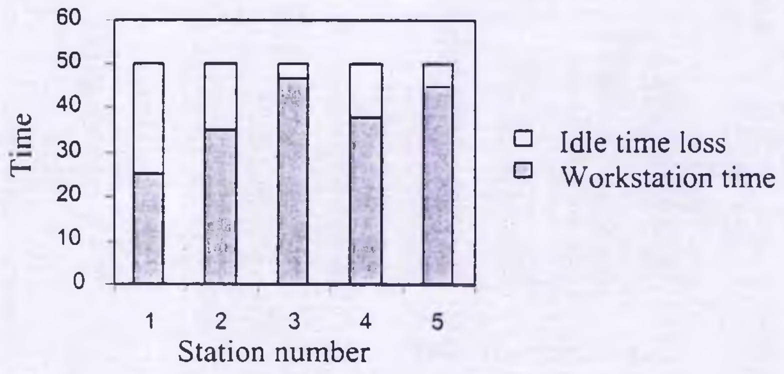


Figure 2.7. All stations not balanced

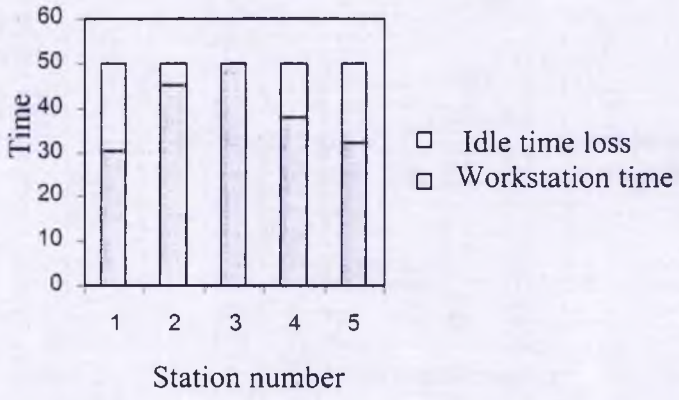


Figure 2.8. Maximum workstation time equals desired cycle time

2.1.6 COMPLEXITY OF THE SIMPLE ASSEMBLY LINE BALANCING PROBLEM

Simple assembly line balancing problems are members of the general class of combinatorial optimisation problems (Domschke and Drexl,1998, p.113). They are NP-hard because they may be reduced to the partition problem which is known to be NP-complete (Karp, 1972). These problems are related to various assignment, sequencing, grouping and selection problems and it is unlikely that a polynomially bounded optimal algorithm exists.

The most obvious correspondence results from omitting the precedence constraints. Then the simple assembly line balancing problem reduces to the well-known *bin-packing problem*, which is to pack a given collection of items into a minimum number of equal-sized bins. (Wee and Magazine, 1982).

In general, the complexity of a problem concerns the requirements of resources like computation time and storage space needed for solving the problem with a computer program (representing a particular algorithm). Most commonly, the computation time requirements are considered as the dominant factor deciding whether or not an algorithm is efficient enough to solve a problem and depends upon a variety of factors including problem size, cycle time, number of workstations in the line, precedence diagram, processing speed of the machine and the method used.

Depending on the precedence diagram, there may be an enormous number of feasible sequences with only one or a few of them optimal solutions. Theoretically, an optimal solution can be obtained by scanning all these feasible solutions, however it is not always practical, especially when the problem is large. If there are *n* tasks in a job and if these tasks are independent, then the total number of feasible sequences generated is *n!* However if these are dependent (i.e., there exists precedence requirements), and there are *r* arcs in the precedence diagram, then approximately, there would be $n!/2^r$ distinct sequences (Buzacott and Shanthikumar, 1993).

A number of numerical measures (besides the number of elements *n*) have been published in the assembly line balancing literature for measuring problem complexity. They are as follows:

1. *Order Strength* (Mastor, 1970, Bhattacharjee and Sahu, 1990): the Order Strength (OS) measures the relative number of precedence relations in the precedence diagram. Problems with large order strength are basically expected to be more complex than those with small OS values.

$$Order\ Strength = \frac{2P}{n(n-1)} \tag{2.2}$$

   Where P is the number of precedence relationships and n is the number of elements

2. *Flexibility Ratio (FR)*: Dar-El (1973) defined flexibility ratio $FR \in [0,1]$ and it is the number of zero entries in the (transitive) precedence matrix divided by the total number of entries. This measure is equivalent to the order strength because $FR = 1\text{-}OS$ (Elmaghraby and Herroelen, 1980).

3. *West Ratio* (Dar-El, 1973): The west ratio $WR = n/m$ is the average number of tasks per workstation and problems with small values of WR tend to be more complex than those with larger values.
4. *Time Interval* (Wee and Magazine, 1981b): The time interval $TI = [t_{min}/C, t_{max}/C] \subseteq [0,1]$ is a measure of the interrelation between the cycle time and the

task times. A small TI length indicates that the task times vary only in a small range. Furthermore, the position of TI in the interval [0,1] shows if the task times are large or small relative to the cycle time C. Following the above argument, problems with small TI and having a position near the right border of [0,1] are expected to be relatively complex.

5. *Time Variability ratio* (*TV*): A measure similar to the time interval is defined as follows:

$$TV = \frac{t_{\max}}{t_{\min}} \tag{2.3}$$

In contrast to TI, only one value is used to characterize a problem instance. Furthermore, this measure is independent of the cycle time, therefore, reflects the time structure of the precedence diagram rather than single problem instance. The complexity of problem instances is expected to grow with decreasing values of TV.

6. *Project Index (PjtI)*: It seems that none of these measures represents the true complexity of the problem due to omitting either precedence or task time distribution. However, Driscoll and Thilakawardana (2000c) defined a new compound measure called *Project Index* considering both task and precedence distributions and, mathematically expressed as follows:

$$PjtI = \frac{1}{4}\left[\frac{c_p - 1}{n-1} + \frac{c_{pav}}{c_p} + \frac{t_{av}}{C} + (1 - \frac{2t_{sd}}{C})\right] \times 100\% \tag{2.4}$$

Where c_p is number of precedence columns, c_{pav} is average column position, t_{av} represents the average task time and t_{sd} is the task time variance.

2.1.7 PERFORMANCE MEASURES

Two main measures of solution quality have been defined to support line-balancing modelling: *Line Efficiency (LE)* and *Balance Efficiency (BE)*, (Kilbridge and Westter, 1961b). Both measures are dimensionless and scaled between zero and one hundred percent, with one hundred percent representing excellence. Line Efficiency represents positive achievement in line utilization and is the key measure of economic performance. Balance Efficiency is representative of the distribution of work with consequent personnel satisfaction combined with increased opportunities for greater output. The mathematical expressions for these measures are as follows:

$$LE = \frac{\sum_{i=1}^{n} t_i}{m \times C} \times 100\% \tag{2.5}$$

$$BE = \left(1 - \frac{\sum_{j=1}^{m} \left| S_j - S_{av} \right|}{m \times S_{av}} \right) \times 100\% \tag{2.6}$$

In addition to the above two main measures the following measures also can be found in assembly line balancing literature. Kilbridge and Wester (1961b) proposed *Balance Delay (BD)*, a measure of assembly efficiency, and is the ratio of the total idle time and the total time spent by the product in moving from the beginning to the end of the line.

$$BD = \frac{m \times C - \sum_{i=1}^{n} t_i}{m \times C} \tag{2.7}$$

Moodie and Young (1965) introduced another measure called *Smoothness Index (SI)*, which measures the equality of the distribution of work among the workstations.

$$SI = \sqrt[2]{\sum_{j=1}^{m} (S_{\max} - S_j)^2} \tag{2.8}$$

Both these measures are noted as weak because of data sensitivity and non-linearity.

2.1.8 LINE BALANCING APPROACHES

Since the pioneering study of Bryton (1954) a large number of heuristic and exact algorithms have been suggested to solve the NP-hard assembly line balancing problem. In the early 1950's, researchers suggested analytical techniques including linear and integer programming but later on they realized that these techniques would fail when the problem size is large. As a consequence practitioners developed heuristic approaches, solving assembly line balancing problems at a low cost of computation time and low volume of computer storage. Although, these techniques would not guarantee optimal solutions, significantly good results were reported. This section will review the previously published heuristic and exact algorithms for the single model deterministic assembly line balancing problem.

2.1.8.1 EXACT (OPTIMUM-SEEKING) ALGORITHMS

Baybas (1986a) provided a comprehensive survey of optimum seeking assembly line balancing techniques and showed most of the methods are based on the well known Branch-and-Bound (B&B) technique. The Branch and Bound technique has received particular research attention and is widely used as an optimal procedure for larger problems. In general, when the number of tasks is large, all exact algorithms fail, in the sense that the CPU time grows exponentially.

The early linear programming and dynamic programming models have been replaced by the more realistic Integer Programming (IP) formulations and solution techniques. Neither integer programming nor the recently introduced goal programming has been used on other than single product line studies. The shortest path technique, perhaps due its added computational efficiency, has been popular for obtaining optimal solutions for the smaller assembly line balancing formulations.

Salveson (1955) formulated the single assembly line balancing problem as a liner-programming problem encompassing all possible combinations of station assignments. His model, by definition can result in split tasks and therefore, may result in infeasible solutions. Bowman (1960) was first to provide a '*nondivisibility*' constraint, by changing the linear-programming formulation to one of integer programming. White (1961) showed that those constraints are redundant and constructed a new zero-one program based on Bowman's formulation.

Klein (1963) proposed an approach to solve the assembly line balancing problem based on integer programming. It requires the solution of a series of assignment problems. Klein proposed to generate all feasible sequences of tasks for the given precedence diagram and then, the assignment minimising the total idle time can be determined by solving the associated assignment problem. This has to be repeated for each feasible sequence and the optimal solution would be among the solutions to the series of the assignment problems.

Thangavelu and Shetty (1971) applied general integer programming for the simple assembly line balancing problem. Their zero-one programs were solved by applying the additive algorithm of Balas (1965), as presented by Geoffrion (1967). Since this method is a general integer programming method and it relies on integer programming theory and codes.

Patterson and Albracht (1975) developed a specialised method, which also relies on IP (Integer Programming) techniques for solving line-balancing problems. This method examines sequences of 0-1 programs for feasible solutions.

Schrage and Baker (1978) proposed an efficient method for generating all feasible sets and a method for assigning to each feasible set a label that can be used as a physical address for storing information about the set within the framework of a dynamic programming approach to sequencing problems with precedence constraints.

Using a Jackson (1956) type of enumeration tree, Van Assche and Herroelen (1979) constructed a frontier search method but one that is not based on IP theory or codes. They represent a tree search procedure, dominance rules, bounding arguments and branching heuristics with a node representing the assignment of tasks to a single workstation.

Wee and Magazine (1981a) also reported a branch-and-bound method, not based on general IP codes or theory. They used simple heuristics termed IUFFD (Immediately Updated First Fit Decreasing) and backward recursive positional weight (BRPW) and immediately updated to generate m_{max}. The first one is a variation of the well-known *bin packing* heuristic based on the *First-Fit-Decreasing* (FFD) rule, which is one of the most efficient heuristics for the bin-packing problem.

Talbot and Patterson (1984) proposed a new formulation, one not involving binary decision variables. The decision variable in this formulation is defined to be the number of the workstation a particular task is assigned to.

In the recent past, a few exact methods based on branch-and-bound techniques have been reported in the literature. FABLE developed by Johnson (1988) and EUREKA developed by Hoffmann (1992) have gained much attention due to excellent performance. Scholl (1999, p.118) described that FABLE and EUREKA may be seen as prototypes for future developments.

2.1.8.2 HEURISTIC PROCEDURES

Talbot et al (1986) published an excellent comparative evaluation of heuristic line balancing techniques and showed that published heuristic decision rules vary from simple list processing procedures, which consider a single attribute of each work task to optimal seeking procedures that have had the amount of computation time to devote to each search limited by an externally imposed time limit.

To analyse these heuristic procedures, they are grouped into three categories. These categories allow capturing salient features of each approach, and to emphasize the similarities and differences among them. The first category consists of single-pass decision rule procedures which implement list processing prioritising schemes for task assignment based on a single attribute of each task. The second category consists of procedures that produce multiple single-pass solutions and select the most attractive solution. The last category consists of procedures that attempt to improve a solution or a station assignment by some iterative backtracking methods.

(a) Single-pass decision rule procedures

Helgeson and Birnie (1961) proposed the well-known and popular 'Ranked Positional Weight technique' (RPWT). In this technique, each task is given a weight equal to its task time plus the sum of all the task times of all the elements that follow it on the precedence diagram. A Complete description of the technique and an illustrative example is given in section 2.1.9. Since the method is very popular in the literature, it appears in almost every comparative study and in several textbooks. Ignall (1965) reported that the method results in solutions far from the optimum. Mastor (1970) also supported Ignall (1965), showing that the technique performs worse than almost all of the other techniques compared in his study..

Tonge (1960, 1961) developed a heuristic technique for the problem consisting of three phases: (a) simplification of the initial problem by grouping adjacent tasks into compound tasks, (b) solution of the more simple problems by assigning tasks to stations at the least complex level possible, breaking up the compound tasks into their elements only when necessary for a solution; and (c) smoothing the resulting balance by transferring tasks among stations until the distribution of assigned time is as even as possible.

Kilbridge and Wester (1961a) proposed a technique developed primarily to balance lines without the aid of computers. The main feature of the technique is to group tasks into columns in the precedence diagram where tasks are placed as far left as possible without violating the precedence relations. In such a diagram, tasks can be permuted among themselves in each column and some of the tasks can be moved laterally from their columns to positions to their right without violating the precedence relations. Then, two properties of the tasks in the diagram, *permutability* with column and *lateral transferability*, are exploited in an attempt to achieve optimum balance.

As Kilbrige and Wester (1961b) stated, the technique is not a mere mechanical procedure, since a fair amount of judgement and intuition must be used to derive a meaningful solution. It is a simple, powerful technique, especially for large cycle times, when one station crosses several columns. On the other hand, for low cycle times, where one column may require two or more stations, a fair amount of adjustment is necessary.

Kilbridge and Wester (1962a) applied the technique to a problem taken from industry in which fixed facilities and positional restrictions exist. They also examined the relation of balance delay (equation 2.7) with various problem parameters, e.g. the range of task time, cycle time and degree of precedence relation flexibility. They report that balance delay is very sensitive to the right selection of cycle time.

Agrawal (1985) developed a procedure utilising a decision rule called 'largest set rule' for allotting the work to stations. The procedure computes the cumulative time for each task, which is the time for performing the task and all the tasks preceding it. Then, the largest cumulative time, which is less than the cycle time, is selected and the associated tasks are assigned to the workstation. The procedure is repeated on the truncated precedence diagram until all the tasks are assigned. After the work is allotted to workstations, the designer should decide on the sequence in which these operators should be positioned on the line. Although the procedure is computationally efficient, there is no apparent guarantee of yielding a good solution.

Baybars (1986b) suggested a procedure that consists of five phases. The first four phases reduce the size of the problem by utilizing various properties of the problem and the last phase is a single-pass heuristic procedure applied on the reduction problem. The procedure starts with the last tasks in the precedence diagram and proceeds backwards. The tasks with the most unassigned immediate predecessors among the tasks with no unassigned followers are assigned first. Detailed computational results on the 70-task problem of Tonge (1961) as well as results of other problems reported in the literature indicate that the procedure finds the optimal solution in most cases with minimal computational time.

(b) Multiple single pass solution procedures

Tonge (1965) proposed a procedure, which assigns tasks to stations by randomly selecting a heuristic procedure for choosing the next task to add to the current station. Based on his studies, it is reported that random selection of heuristics for choosing the next task does as well as or better than, using an individual heuristic procedure alone, and randomly choosing the tasks without an intervening choice of heuristic procedures.

Arcus (1966) developed a technique called 'Computer Method of Sequencing Operations for Assembly Lines' (COMSOAL), in which the main idea is the random generation of a feasible sequence. The .technique assigns the same probability of selection to the tasks with no unassigned predecessors and fits the remaining station time. Judging on the basis of the yield of good balances, Arcus has explored methods of biasing the tasks available for selection producing the best results with a compound selection models.

Buxey (1974) improved COMSOAL further with paralleling of workstations leading to possible reductions in total idle time. He applied the same approach to the RPWT of Helgeson and Birnie (1961). Each workstation that is duplicated is assumed to have an effective cycle time of *C* times the station multiple. Thus, a rage of times becomes available and there is more likelihood of a better fit. Multiple stations also enable the production to be greater than the limitation imposed by $t_{max.}$.

Pinto et al (1978) presented a heuristic network procedure based on the shortest-route formulation of Gutjahr and Nemhauser (1964) in which the nodes represent a collection or subset of tasks that can be performed in some order without prior completion of any task not in the subset. Pinto et al (1978) utilized other heuristic procedures, including RPWT, largest task time, smallest task time and random task assignment, to generate the nodes. The set of nodes generated is combined to form a composite network.

Schofield (1979) developed a procedure called 'Nottingham University Line Sequencing Program' (NULISP) which is similar to COMSOAL. This technique can solve both *line balancing* and *cycle time problems,* handling various zoning constraints and task times larger than the cycle time. The details of the feasible sequence generation are not reported due to copyright reasons; however, it is stated that a weighted random selection procedure is utilized to generate solutions. The major advantage of NULISP is the feature of considering various factors, including grouping of tasks for a variety of reasons, separation of one group of tasks from others for reasons of skill differences, safety considerations, and fixing of tasks at certain workstations to account for fixed facilities on the line.

Akagi et al (1983) proposed a method, which allows the assignment of more than one worker to a workstation. Tasks are assigned to workstations according to a couple of rules reported in the literature. The procedure was repeated for a different number of workers at each workstation. In the second phase of this two-phase technique, tasks are assigned to workers within each workstation.

Nakasu and Leung (1995) developed procedure similar to COMSOAL in the sense that the best design is selected among the several generated via simulation. Performance measures of minimizing the number of workstations, cycle time, balance delay and a combination of these are considered. The procedure allows the task times and cycle time to be sampled from various probability distributions. Neither experimental results nor comparison of the procedure with the others in the literature are given; thus, it is impossible to comment on the performance of the procedure.

(c) Backtracking procedures

Hoffmann (1963) developed an enumeration method, by generating all feasible station assignments that do not exceed cycle time and selects the best arrangement from among these by use of a triangular precedence matrix. The procedure selects as the first workstation the feasible subset of tasks that leaves the least idle time, then selects from the remaining tasks that subset that leaves the least idle time in the second workstation, etc. the method is coded in FORTRAN that can solve problems with up to 99 elements. Although the method may be computationally very expensive, Gehrlein and Patterson (1978) demonstrated that by suitably modifying the method it could be used to solve problems of moderate sizes.

Moodie and Young (1965) suggested a two-phase procedure. In the first phase, selecting the task with no unassigned predecessors and fitting the remaining station time in the order of largest performance time obtain a preliminary balance. In the second phase, tasks are shifted between stations in an attempt to reduce idle time and distribute the idle time equally to all workstations. Similar to this procedure, Sarker and Shanthikumar (1983) developed another procedure that enables the balancing of lines, some involving task times greater than the cycle time.

Nevins (1972) introduced a general purpose heuristic program called the 'best bud search' that does not attempt to minimize number of workstations directly; instead, an upper bound on the number of workstations is imposed and the problem is solved for that many workstations. If the attempt is successful, the number of workstations is decremented by one, and another attempt is made until it is either impossible or computationally impractical to get a smaller number of workstations. Nevins (1972) tested the problems solved by Tonge (1961) and obtained as good or better results.

Dar-El (1973) developed a method called MALB for the cycle time problem starting with the minimum theoretical cycle time and proceeds with the generation of a feasible sequence of tasks, which are grouped into workstation assignments. If a feasible sequence cannot be extended, the method applies a backtracking procedure, which either partitions the tasks correctly or results in an increase of one time unit of the cycle time. The method is further improved by imposing rules which limit the backtracking iterations. This method performs better than COMSOAL and 10-SPP (a method selecting the best of 10 solutions, each obtained by using a different ranking system, e.g. as with RPWT) in the problems tested.

Bennett and Byrd (1976) presented a two-stage 'trainable heuristic procedure'. In the first stage, the procedure is trained by accumulating experience on the effectiveness of several heuristic rules on small problems for which the optimum is known. In the second stage, the findings of the stage are used to provide near optimal solutions, which is fed to an optimisation procedure as a starting point. The authors have used several empirical rules and values with no apparent justification.

Dar-El and Rubinovitch (1979) developed another method that generates alternative solutions of equal quality by employing exhaustive enumeration to generate all or some subset of the solutions. However, the computational requirements of the exhaustive enumeration grow exponentially with the number of subsets saved.

This method is called 'Multiple Solutions Technique' (MUST) and performs better than or gives equal quality results with MALB (Dar-El, 1973) in every case.

Hackman et al (1989) suggested a branch and bound procedure that incorporates several heuristic fathoming rules to reduce the size of the tree. The authors reported that the procedure outperforms the other branch-and-bound procedures in the literature. This procedure can also be adapted to solve the cycle time problem.

Shtub and Dar-El (1990) utilized MALB for a multi-objective approach for both line balancing and cycle time problems. The objective functions consist of the traditional objective of minimizing the total idle time and minimizing the number of sub assemblies handled at each workstation.

Reference	Year	Solution methodology
Buxey	1979	Monte Carlo simulation
Agrawal	1985	Single-pass similar to RPWT
Lau and Shtub	1987	Procedure based-on a hybrid line concept
Hackman et al	1989	Branch-and-bound algorithm with
Yano and Bolat	1989	Heuristic branch-and-bound procedure
Easton	1990	DP with relaxation and fathoming
Hoffmann	1990	An enumeration procedure based on
Shtub and Dar-El	1990	Procedure utilizing assembly chart
Faland et al	1992	Heuristic shortest-path
Leu et al	1994	Genetic Algorithm
Miltenburg and Wijngard	1994	DP and heuristic for U-Line problem
Anderson and Ferris	1994	Genetic Algorithm Hoffmann's heuristic
Kim et al	1996	Genetic Algorithm
Scholl and VoB	1996	Priority-ranking heuristic and tabu search Exact method in the literature

Table 2.1. Recent researches conducted on assembly line balancing problems.

Easton (1990) presented a dynamic programming (DP)-based approach with relaxation and fathoming that relies on dynamic upper bound. The dynamic programming formulations of the assembly line balancing problems with realistic sizes however require excessive storage and computation time.

Driscoll et al (2002) developed a heuristic approach based on the Hoffman's precedence matrix procedure. This approach generates feasible task assignments randomly unlike the Hoffman procedure for a number of times (user defined) for each workstations and the best combination (combination with least idle time) of tasks are assigned to the workstation. Starting with the first workstation, the procedure is repeated for the next workstation and so on until all the tasks are assigned to workstations. Experiments with a number of benchmark problems in the literature showed that this technique outperforms the Hoffman procedure and solved most small-medium problems with optimal solutions.

Concluding this group of approaches, Erel and Sarin (1998) presented the most recent survey of the assembly line balancing procedures and are listed with some basic information on the approaches in table 2.1. Considering the large number of studies reported in the literature, it was concluded that the development of procedures for single model deterministic versions of the problem still continue to be an attractive research area.

2.1.9 ILLUSTRATIVE LINE BALANCING EXAMPLE

An example problem with 12 tasks and a cycle time of 10 time units is considered for illustration. The problem network is shown in figure 2.9. The well-known Rank Positional Weight (RPW) technique developed by Helgeson and Birnie (1961) is used for balancing and the step-by-step computation is shown below.

The largest cycle time for doing this work would be equal to the total work content (*WC*) that is:

$$WC = \sum_{i=1}^{n} t_i \tag{2.9}$$

In this example this is equal to 50 time units, and involves only one workstation. The smallest cycle time for doing the work is equal to the largest element time (t_{max}). In the example, this is 7 time units and would require 8 workstations, which is calculated from:

$$\frac{\text{Total element times}}{\text{cyle time}} \text{ (Rounded where necessary to the next integer up)}$$

In practice there is a limitation on the feasible number of workstations, which can be calculated using the following guide.

1. Each workstation time must be less than or equal to the cycle time.
2. The minimum number of work stations $m*$ is

$$m* = \frac{\sum_{i=1}^{m} t_i}{C} \text{ (Rounded where necessary to the next integer up)}$$

3. The feasible number of workstations is equal to the number of elements where the element time is greater than half the cycle time.

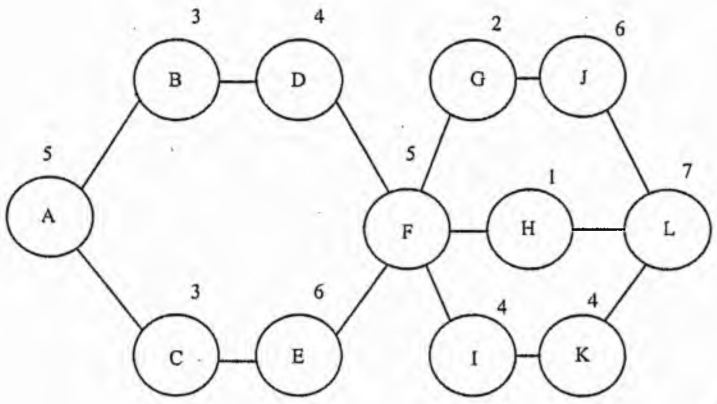


Figure 2.9. Precedence network

Element	Positional Weight
A	34
B	27
C	29
D	24
E	25
F	20
G	15
H	8
I	15
J	13
K	11
L	7

Table 2.2 Positional weights

The steps involved in the Rank Positional Weight method are as follows:

1. Develop the precedence network in the normal manner.
2. Determine the positional weight for each work element (a positional weight of an element is defined as the element's time plus the time of all elements that

must follow according to precedence relationships). The positional weights for the problem are shown in table 2.2.

3. Rank the work elements based on the positional weight in step2. The work element with the highest positional weight is ranked first (table 2.3).
4. Assign the element with the highest positional weight to workstation1.
5. Calculate the remaining time (C- $\sum_{k=1}^{n} t_k$), where t_k is the assigned element times and k is the number of elements assigned to the workstation.
6. Assign the element with next highest positional weight which fulfils:
   a. Precedence restrictions
   b. Element time $\leq$ (C- $\sum_{k=1}^{n} t_k$),
7. Calculate remaining slack time $t_s = C - \sum_{k=1}^{n} t_k$),
8. Repeat steps 6 and 7 with decreasing t_s values until one of the following conditions apply:
   a. Precedence restrictions prohibit further assignment
   b. $t_s = 0$
   c. Remaining element times are greater than t_s
9. Start second workstation selecting first unassigned element with highest positional weight.
10. Continue until all elements have been assigned

Rank	Element	PW
1	A	34
2	C	29
3	B	27
4	E	25
5	D	24
6	F	20
7	G	15
8	I	15
9	J	13
10	K	11
11	H	8
12	L	7

Table 2.3 Element ranks according to positional weights.

Referring to the above procedure and using the following standard layout on the example the solution would be:

Cycle time $C = 10$ units

Rank	Element Number	Check on precedence	Element time	$(C-\sum t_k)$	Comment
Workstation 01					
1	A	✓	5	5	Assigned
2	C	✓	3	2	Assigned
3	B	✓	3	-1	Not assigned
4	E	✓	6	-4	Not assigned
Elements B and subsequent elements fail on time restriction					
Workstation 02					
3	B	✓	3	7	Assigned
4	E	✓	6	2	Assigned
5	D	✓	4	-2	Not assigned
6	F	x			Not assigned
Element F and subsequent elements fail on precedence restrictions					
Workstation 03					
5	D	✓	4	6	Assigned
6	F	✓	5	1	Assigned
7	G	✓	2	-1	Not assigned
8	I	✓	4	-3	Not assigned
9	J	x			Not assigned
10	K	x			Not assigned
11	H	✓	1	0	Assigned
12	L	x			Not assigned
No time remaining					
Workstation 04					
7	G	✓	2	8	Assigned
8	I	✓	4	4	Assigned

9	J	✓	6	-2	Not assigned
10	K	✓	4	0	Assigned

No time remaining

Workstation 05

9	J	✓	6	4	Assigned
12	L	✓	7	-3	Not assigned

Fail on time restriction

Workstation 06

12	L	✓	7	3	Assigned

The final solution to the above problem is summarized in table 2.4.

Workstation	Elements	Station time	Slack time
01	A, C	08	02
02	B, E	09	01
03	D, F, H	10	00
04	G, I, K	10	00
05	J	06	04
06	L	07	03

Table 2.4 Workstation details and station times

Performance measures:

$$\text{Line Efficiency(LE)} = \frac{50}{6\times 10}\times 100\% = 83.3\%$$

$$\text{Balance Efficiency} = \left(1-\frac{0.83}{6\times 0.83}\right)\times 100\% = 83.33\%$$

2.1.10 ASSEMBLY LINE BALANCING SOFTWARE

Chase (1975) revealed that only five percent (5%) of companies used the published techniques to balance their lines! Later Milas (1990) showed the situation has not changed much. The main reason for this dilemma was identified as the practitioner's unfamiliarity with the published algorithms; the complexity of the algorithms, making their comprehension by practitioners difficult; the inflexibility of the algorithms to model the actual conditions of assembly lines and the non-availability of user friendly ready to use software packages for assembly line balancing. However, two software packages have been reported in the literature.

Chang and Sullivan (1991) developed a software package called QS (Quant Systems) for assembly line balancing. The software was based on five sets of heuristics having objectives of maximizing line efficiency and providing a balanced workload. It solves the assigned problem using the five techniques and overall the best is considered as the solution.

A~Line is a second integrated software package developed recently at the University of Surrey. It is capable of project organization, data management, problem analysis, and balancing and results evaluation. A~line 1.4, has been released for review and demonstrator versions and an initial description of the software models published (Driscoll and Thilakawardana, 2000a and 2000b)).

The default random generator model was used as a comparator to new balancing approaches, and to validate and test the new software enhancements. There are a number of supportive features in the package. Data validation is included and has been revised to identify both critical and non-critical errors. Redundant precedence restraints are an example of non-critical errors that can be removed automatically by software identification.

Automated drafting of precedence diagrams is included along with the ability to manually adjust a diagram for clarity. Test cases of over two hundred elements can be handled by the package and following data validation plots of theoretical line efficiency, often refereed to, as '*saw-tooth diagrams*' are available for cycle time selection.

Balancing is in two forms, individual balance investigations and balance scanning. Balance scanning will automatically manage and process a continuous series of balances over a range of acceptable cycle times and installed balancing models. The results are then available for presentation in the form of line efficiency and balance efficiency graphs plotted against theoretical results.

Individual detailed balances at prescribed cycle times can be processed on a station-by-station basis with a display of current element, all current station assignment attempts and total assignment history. Statistics are recorded on individual balance model performance when best solutions are found.

2.1.11 SUMMARY

The single model deterministic assembly line balancing problem continues to generate interest both due to its practical and theoretical nature as evidenced by recent publications. These publications have addressed four types of line balancing problems. The single model deterministic is the simplest version of an assembly line and the most researched. A limited number of papers have been published on multi/mixed deterministic and multi/mixed stochastic.

Determining the optimal solution or set-up of an assembly line for least cost is considered as the assembly line balancing problem and it comprises two separate sub problems: the *cycle time problem* and *line balancing problem*, which must be solved sequentially. Assembly line balancing problems are combinatorial optimisation problems, which are considered as NP-hard problems. They are very complex in nature and cannot be solved in polynomial time.

During the last four decades a large number of exact and heuristic techniques have been developed to solve the problem, but none of them guarantee a 100% optimal solution. It can be seen that very early techniques are based on mathematical programming optimisation techniques and later it was realised that these exact techniques become prohibitive beyond limited problem dimensions. As a result of this, researchers started focusing on heuristic approaches and they later gained much attention over exact methods. Therefore, heuristic techniques will remain the only computationally efficient and sufficiently flexible methodologies capable of addressing large scale, real-world assembly line balancing problems, particularly for the multi/mixed models and general assembly line balancing categories.

After the introduction of a metaheuristic approach called the Genetic Algorithm (GA), by John Holland in 1975, its applications to NP-hard combinatorial optimisation problems were studied and an excellent capability of solving the NP-hard class of problems efficiently in short convergence times was identified. The application of the Genetic Algorithm to assembly line balancing problems has been explained in the second part of this literature review.

2.2 THE GENETIC ALGORITHM AND ITS APPLICATION TO LINE BALANCING

This section consists of two main sub-sections. The first section explains the basic mechanisms underlying the classic Genetic Algorithm and its key components including coding, fitness functions, initialisation, genetic operators a nd termination, and its applications to the assembly line balancing problem The second section reviews p revious a pplication o f G enetic A lgorithms t o t he a ssembly l ine b alancing problem, identifies the drawbacks of the existing models and consequently the contribution possible from this research. Genetic Algorithms are biologically inspired computational models and much of the terminology has been borrowed from the field of genetics, mathematics and computer science, therefore, a glossary is provided in appendix A to help the reader with the terminology.

2.2.1 PRINCIPLE BEHIND THE GENETIC ALGORITHM

A Genetic Algorithm is a set of procedures which, when repeated, enables solutions to be found for specific problems. To accomplish the desired objectives, Genetic Algorithms create successive populations of different solutions until an acceptable solution is reached. Within the generation of each successive population, improvements in the quality of chromosome solutions are achieved. In such a manner, a Genetic Algorithm can quickly move to a successful outcome without examining every possible solution in the search domain. The concept used is based upon the fundamental processes that control the evolution of biological organisms, namely natural selection and reproduction. These two processes together improve an organism's ability to survive within its environment in the following manner:

1. Natural selection determines which organisms have the opportunity of reproduction and survival within a population.
2. Reproduction involves genes from two separate individuals combining to form offspring that inherit the survival characteristics of their parents.

The Genetic Algorithm seeks to imitate the way in which beneficial genes reproduce themselves through successive populations and hence contribute to the ability of an organism to survive.

2.2.2 BACKGROUND TO GENETIC ALGORITHMS

The beginnings of Genetic Algorithms can be traced back to the early 1950s when several biologists used computers for simulation of biological systems (Fisher (1958)). Inspired by the principle of natural genetics and the theory of evolution the research completed in the late 1960s and early 1970s at the University of Michigan under the direction of John Holland led to Genetic Algorithms.

Deoxyribo Nucleic Acid (DNA) is the basic building block of chromosomes present in every living organism that determines many traits of the organism. The growth of the organism, from the first fertilized egg up to the adult individual, is carried out by highly complex physicochemical processes, which are governed by DNA that constitutes the 'recipe' how to 'make' the individual. Each chromosome is composed of *genes*, which can be though of as the basic 'unit' of information. The order of genes appearing in the chromosome decides the characteristic features of individual species in a population. The different traits of one generation are passed on to the next through various genetic operators. Combining this process with the survival of the fittest, leads to a population well adapted to the environment.

In the Genetic Algorithm, the basic building block is also the *gene*. These genes represent problem elements and their values are known as *alleles*. A number of genes arranged in an order make a *string (chromosome)*. Theses strings represent feasible or infeasible solutions in the problem search space. A group of strings are collectively known as *schemata*, representing different solutions of the problem. Figure 2.10 shows the analogy between the Genetic Algorithm and the principle of natural genetics.

Goldberg (1989a) and Liepins and Hillard (1989) presented detailed insights into different aspects of Genetic Algorithms. In Genetic Algorithms each solution is stored in the form of an artificial chromosome and is represented by a string of bits, numbers etc. The search proceeds in parallel in the neighbourhoods of the better solutions due to the multiplicity of solutions. The trials become less random as the number of generations progress, since the number of desirable chromosomes increases in the population. Thus Genetic Algorithms are intrinsically parallel. New candidate solutions are obtained from the current population by applying artificial genetic operators. Efficient solutions to large combinatorial optimisation problems at a very low computational cost are possible by the application of judicious combinations of these operators.

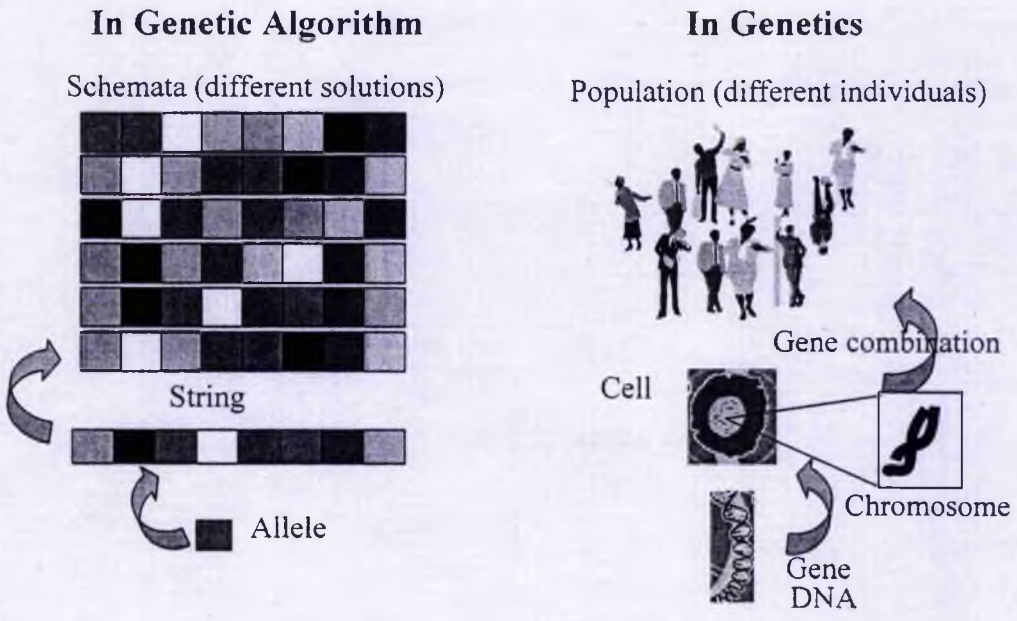


Figure 2.10. Analogies of two genetic systems

The application of Genetic Algorithms to the assembly line balancing problem can be traced back to Anderson and Ferries (1990), who applied the Genetic Algorithm technique to solve the assembly line balancing problem for the first time. Early research on these problems tended, naturally, to use classical operations research techniques. Conventional search techniques, including integer programming (Graves and Lamar, 1983), dynamic programming (Held et al, 1963) and hill climbing (Ackley, 1987) are often incapable of optimising the assembly line balancing problem. On the other hand, the branch and bound technique (Pinto et al, 1981) enabled researchers to continually find global solutions, but tended to be highly computationally expensive.

Current research interests focuses on the Genetic Algorithm advantages, which include:

1. Optimises with continuous or discrete parameters;
2. Does not require derivative information;
3. Simultaneously searches from a wide sample of the cost surface;
4. Deals with a large number of parameters;
5. Is well suited for parallel computers;
6. Optimises parameters with extremely complex cost surfaces; it can jump out of a local minimum;
7. Provides a list of optimum parameters, not just a single solution;
8. Works with numerically generated data, experimental data or analytical functions;

These advantages are intriguing and produce quality results where traditional optimisation approaches have failed, whilst acknowledging the Genetic Algorithm is not the best way to solve every problem. For example, traditional methods have been well tuned to quickly find the solutions of a well-behaved convex analytical function of only few variables. For such a problem, calculus-based methods outperform a Genetic Algorithm, quickly finding the optimal solution while a Genetic Algorithm is still analysing the fitness of the initial solution. However, many realistic problems do not fall into this category.

The large population of solutions that gives the Genetic Algorithm its power is also its bane when it comes to speed on a serial computer, because the fitness function of each of these solutions must be evaluated. However, if a parallel computer is available, each processor can evaluate a separate function at the same time. Therefore, the Genetic Algorithm is optimally suited for such parallel computations.

2.2.3 BASIC COMPONENTS OF THE GENETIC ALGORITHM

The Genetic Algorithm begins like other optimisation algorithms, by defining the optimisation parameters including fitness function, crossover and mutation techniques and probabilities etc. It ends like other optimisation algorithms too, by testing for convergence. In between, however, a Genetic Algorithm is very different from other search algorithms. The basic components of the classical Genetic Algorithm are illustrated in figure 2.11 and the corresponding steps are described below:

1. Define Genetic Algorithm parameters and fitness function.
2. Create initial population.
3. Evaluate the current population.

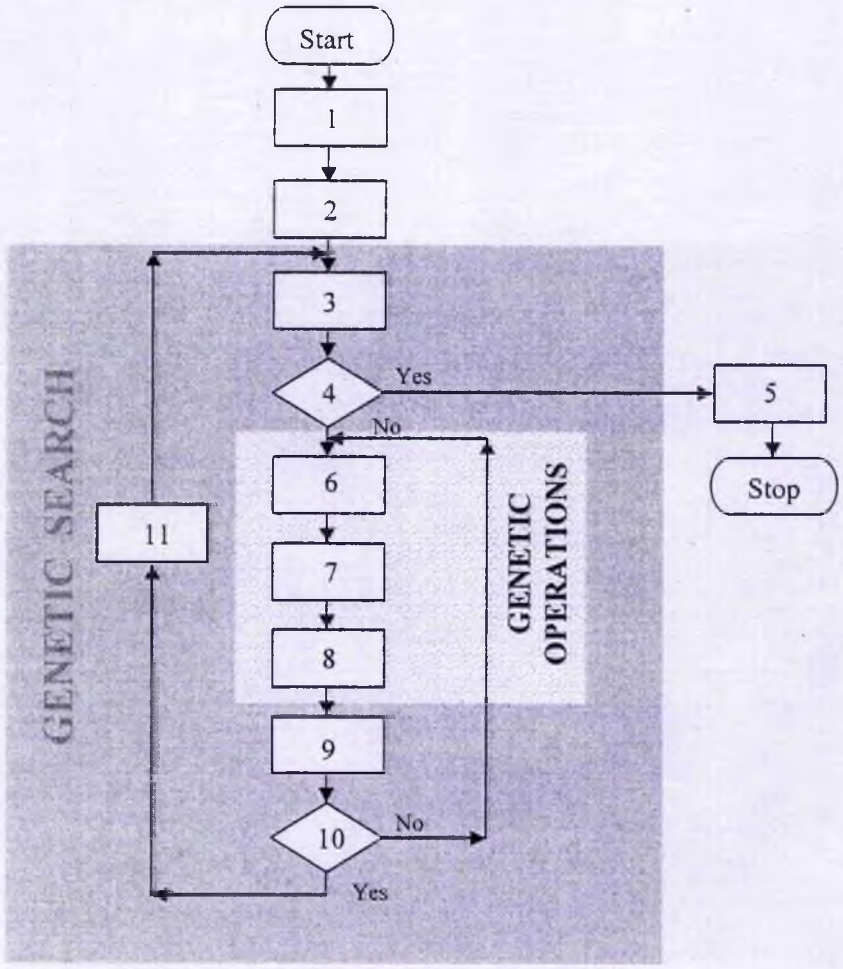


Figure 2.11. Flow chart of the classical Genetic Algorithm

4. If the termination criterion is met, go to step 5, otherwise go to step 6.
5. End.
6. Select two parents and go to step7.

7. Undergo crossover operation according to crossover probability and generate offspring.
8. Undergo mutation according to mutation probability.
9. Copy the offspring to the new population.
10. If the number of chromosomes in the new population is the same as the current population, go to step 11, otherwise go back to step 6 and repeat steps 6-9.
11. Replace the current population with the new population.

2.2.3.1 ENCODING

A Genetic Algorithm starts with an initial population of chromosomes representing different possible solutions to the problem. These chromosomes will produce new chromosomes undergoing genetic operations such as crossover and mutation. Representing the actual problem by chromosomes is consequently the first step when starting to solve problems with the Genetic Algorithm. The mapping of the real problem to artificial chromosomes is known as encoding.

The method of representation (encoding) has a major impact on the performance of the Genetic Algorithm. Different coding schemes may cause different performances in terms of accuracy and computation time. Liepins and Vose (1990) discuss the issue of solution encoding in detail and showed that the existence of a good representation makes a problem easily solvable by Genetic Algorithm. However, defining good technique is a challenge.

Several encoding techniques have been developed over the last two decades including binary encoding, permutation encoding, value encoding and tree encoding. Binary encoding is the most common, mainly because the original work by Holland (1975) used this type of encoding. Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems.

Two chromosome representations applicable to the assembly line balancing problem are introduced.

1. Sequence- oriented representation (Fox and McMahon, 1991).
2. Partition-oriented representation (Bruns, 1993).

Falkenauer (1992) examined both representation schemes, and concluded that the sequence-oriented representation is the best for the assembly line balancing problem representation. One of the advantages of using this scheme is that it provides considerable flexibility in choosing genetic operators (Kim et al, 1996). Many genetic operators that have been developed for sequencing problems are available, and the representation makes it possible for them to be adapted to the assembly line balancing problem.

(a) Sequence-Oriented encoding

The most natural representation of the assembly line balancing problem solution is sequence-oriented encoding, consisting of mapping a possible solution of the problem in the search space into finite chromosomes. In assembly line balancing, the number of workstations in the solution, determining line efficiency (equation 2.5), depends significantly on the element assignment sequence.

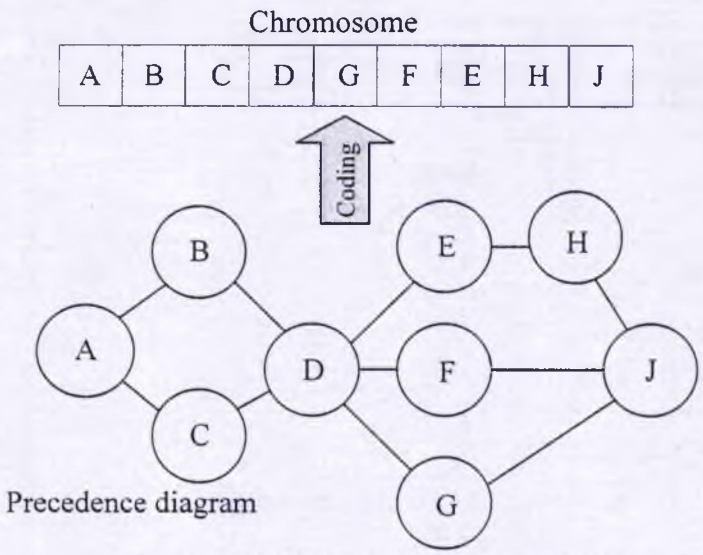


Figure 2.12. Chromosome representation

Transforming this sequence-oriented task assignment into a chromosome is simple and straightforward. The letter placed in a chromosome represents each task element, and therefore, the length of the chromosome is equal to the total number of elements in the problem (*n*). Tasks are ordered in the chromosome from left to right relative to their order of processing. Figure 2.12 illustrates this coding scheme for the 9-element problem considering the element assignment order [A B C D G F E H J].

2.2.3.2 DECODING

Decoding is exactly the reverse of encoding. When the order of elements in a sequence-oriented representation does not violate precedence constraints, it is called a *feasible sequence*. The feasible sequence carries many possible assignments rather than one fixed assignment. In order to determine the best assignment, the chromosome should be properly decoded.

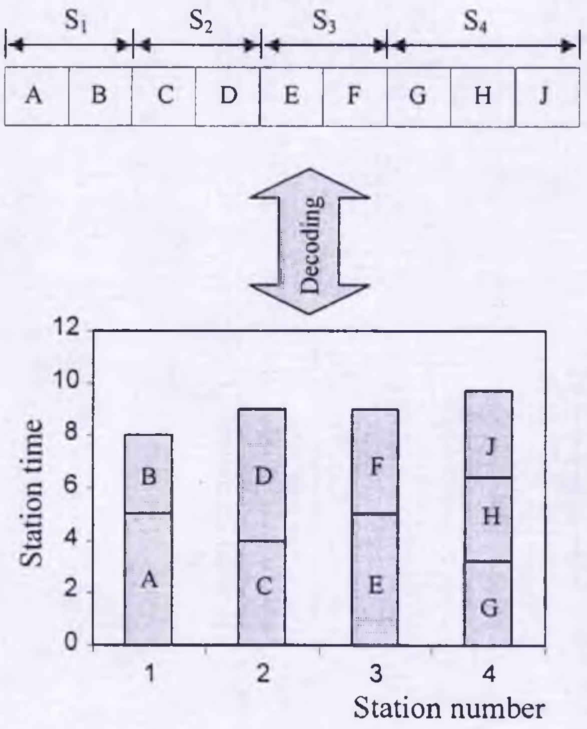


Figure 2.13. Decoded solution

When the cycle time is given, the procedure used by Fox and Mahon (1991) is employed for decoding. A workstation is created, and tasks are assigned to the workstation in the order they appear in a feasible sequence while not exceeding the cycle time. This process is repeated until all the tasks are allocated. This decoding method is straightforward, and is used for type I (line balancing problem) and type II (cycle time problem) projects. Figure 2.13 shows the decoded solution of the illustrating chromosome in figure 2.12, with a sample ten unit cycle time.

2.2.3.3 FITNESS FUNCTION

A fitness function plays a key role in a Genetic Algorithm. It forces the algorithm to search for optimal solutions based on its numerical value and serves as the only link between the problem and the algorithm. A fitness function ranks chromosomes in the population, so better individuals have a better chance for survival and reproduction and it must be defined cautiously to deal with the engineering realities of the problem. The first attempt to apply the Genetic Algorithm to the assembly line balancing problem was made in 1990 and since then several fitness functions have been introduced for line balancing. The recent applications and the fitness functions developed for line balancing are listed in table 2.5.

Anderson and Ferris (1990) pioneered the application of the Genetic Algorithm for the assembly line balancing problem and suggested that the fitness function must include an element corresponding to the total time for the operations assigned to the lowest station. Furthermore, infeasible solutions violating precedence constraints are avoided by assigning a large penalty cost. The fitness function developed is shown in equation 2.10. S_{max} is the highest station time, S_{max2} is the second highest station time, S_{min} is the lowest station time and V is the number of precedence violations. The constants d_a, e_a, f_a, and k_a are chosen as follows: $d_a > 1$ in order that the precedence violations are removed as quickly as possible. e_a is chosen so that the second slowest is taken into account.The constant k_a is chosen so that the values of the fitness function lie within reasonable bounds.

Minagawa and Kakazu (1992) proposed a fitness function (equation 2.11) for single model deterministic task times type line-balancing problems. Minimization of cycle time is adopted as the evaluation criterion for the line balancing performance.

Reference	Fitness function	
Anderson & Ferris (1990)	$\exp(-k_a(S_{\max} + (d_a \times V) + (e_a \times S_{\max 2}) + f_a \times (S_{\max} - S_{\min})))$	(2.10)
Minagawa and Kakaz (1992)	$\dfrac{\sum_{i=1}^{n} t_i}{C \times m}$	(2.11)
Falkenauer & Dechmbre (1992)	$\dfrac{\sum_{j=1}^{m} \left(\dfrac{S_j}{C}\right)^k}{m}$	(2.12)
Anderson & Ferris (1994)	$\exp(-hv_i)$	(2.13)
Leu et al (1994)	$\sqrt[2]{\dfrac{\sum_{j=1}^{m}(S_j - C)^2}{m} + \dfrac{\sum_{j=1}^{m}(C - S_j)}{m}}$	(2.14)
Tsujimuya et al (1995)	$\sum_{j=1}^{m}(\bar{c}_k - \bar{s}_{jk})$	(2.15)
Kim et al (1996)	$m - \dfrac{m}{\sum_{j=1}^{m} SN_j}$	(2.16)
Suresh et al (1996)	$\sqrt{\sum_{j=1}^{m}(S_{\max} - S_j)^2}$	(2.17)
	$(1 - \prod_{j=1}^{m} P_j)$	(2.18)
Kim et al (1998)	$\dfrac{1}{m}\sum_{j=1}^{m}\sqrt{\lvert S_j - S_{mean}\rvert}$	(2.19)
Ponnambalam et al (2000)	$w_1 f_1(x) + w_2 f_2(x) + ... + w_i f_i(x) + ...w_u f_u(x)$	(2.20)
Sabuncuoglu et al (2000)	$\sqrt[2]{\dfrac{\sum_{j=1}^{m}(S_{\max} - S_j)^2}{m} + \dfrac{\sum_{j=1}^{m}(S_{\max} - S_j)}{m}}$	(2.21)

Table 2.5. Published fitness functions for line balancing

The line balancing problem is NP-complete and can be reduced to the NP-complete bin packing problem (Garey and Johnson, 1979) which it contains as a special case. Falkenauer and Delchambre (1992) pointed out the close connection between the line balancing problem and the bin-packing problem and developed a cost function suitable for the bin-packing problem (equation 2.12). This cost function was then applied to the line-balancing problem with efficient crossover and mutation operations. The constant *k* expresses the concentration on the well-filled 'elite' bins in comparison to the less filled ones. Several values of *k* have been experimented with and it was found that *k* = 2 give good results. Larger values of *k* seem to lead to premature convergence of the algorithm, as local optima, due to a few well-filled bins, are too hard to escape.

Following the previous application in 1990, Anderson and Ferris (1994) proposed a new cost function (equation 2.13), which is less, complicated than the earlier one. In this cost function, the value of a solution is defined as follows:

$$v_i = \max(S_j) + k_b N_v \qquad (2.22)$$

Where S_j is the total time for operations assigned to station *j*, $N_{v'}$ is the number of precedence violations, and *k* is a constant which is set equal to the largest single operation time. *h* is chosen to make the fitness lie in a particular range. One of the drawbacks of this scheme is that they alter the relative fitness of different chromosomes in a quite complicated way. Scaling of fitness values was suggested to overcome this problem and this scaling procedure was used to achieve some degree of control over the speed of convergence of the algorithm.

Leu et al (1994) claimed that most of the assembly line balancing heuristics listed in the literature had not simultaneously considered more than one objective. Consequently, they developed a cost function with multiple evaluation criteria. This cost function consists of two objectives. The first objective is to be taken to be the minimization of mean-squared idle times that is the minimization of z_1.

$$z_1 = \sum_{j=1}^{m} \frac{(S_j - C)^2}{m} \qquad (2.23)$$

This objective would tend to provide workload balance, thereby mitigating concerns of inequity among workers. The second objective (z_2) is to minimize man idle time. Since the first objective is measured in squared time units and the second in (linear) time units, assuming the first objective is more critical than the second. So the over all objective function ($\sqrt[2]{z_1} + z_2$) is given in equation 2.14.

Tsujimura et al (1995) solved the fuzzy assembly line balancing problem using Genetic Algorithms representing the imprecise data using fuzzy numbers. The balance delay definition is used as the evaluation function (equation 2.15). $\bar{c}_k$ and $\bar{s}_{jk}$ are the fuzzy cycle times and fuzzy completion times required to complete all jobs assigned to workstations of each chromosome respectively.

A Genetic Algorithm for solving assembly line balancing Type-I problems with multiple objectives was developed by Kim et al (1996). In Type-I problems, the objective is to minimize the number of workstations, used in many line-balancing problems to give lower labour cost and reduced space requirements. The proposed cost function is given by equation 2.16. SN_j is the number of connected networks in G_j representing precedence relations of tasks assigned to workstation j. The performance comparison between the proposed algorithm and the known algorithms showed this approach is promising

Suresh et al (1996) made use two different fitness functions for solving both deterministic as well as stochastic assembly line balancing problems with varying objectives by making corresponding changes in the cost function. In the first model, the smoothness index proposed by Moodie and Young (1965) was used as the cost function (equation 2.17). The objective of this is to reduce the balance delay and distribute the idle times at each station by arranging the work elements subject to cycle time and precedence constraints in such a way that the smoothness index of the balance is minimized. The station time S_j of each station is calculated using the following relation.

$$S_j = S_{mean} + \sqrt[q]{S_{var}} \qquad (2.24)$$

σ is the confidence coefficient for normally distributed work elements times and S_{max} and S_{var} are the sum of the means and the sum of the variances, respectively, of all the tasks allocated to those particular workstations.

The second evaluation function is based on the trade and transfer method proposed by Reeve (1971). The objective of this was to minimize the probability of a line stopping by rearranging the jobs, subject to all constraints. The probability of a station not exceeding the cycle time is given by P_j . It is the area under the normal curve corresponding to the value of z_r given by

$$z_r = \frac{C - S_{mean}}{\sqrt{S_{\text{var}}}} \tag{2.25}$$

Therefore, the probability of the line stopping is given by equation 18 and was used as the second cost function. Suresh et al (1996) concluded that both the cost functions gave improvements.

Workload smoothing in assembly lines has many beneficial features: it establishes the sense of equity among operators, and more importantly, contributes to increasing the output. Kim et al (1998) suggested a heuristic-based Genetic Algorithm to solve an assembly line balancing problem with workload smoothness as the objective. The workload smoothness was evaluated by using the Mean Absolute Deviation (MAD) proposed by Rachamadugu and Talbot (1991) and used as the cost function (equation 19) for this model. This model was compared with the existing heuristics and with an existing Genetic Algorithm. The results confirmed that this algorithm outperforms the existing heuristics and in many cases, it also improved cycle time.

Ponnambalan et al (2000) proposed a multi-objective Genetic Algorithm to solve assembly line balancing problems. The following four-performance criteria were used as objective functions

1. The number of workstations ($f_1(x)$).
2. The line efficiency ($f_2(x)$).

3. The smoothness index before trade and transfer (*f_3 (x)*).
4. The smoothness index after trade and transfer (*f_4 (x)*).

The weighted sum approach proposed by Murata et al (1996) was used for combining multiple objective functions into a scalar fitness function (equation 2.20). $f_i(x)$ is the ith objective function, w_i is a constant weight for $f_i(x)$, and u is the number of objective functions. The w_i terms are randomly generated using the following relation:

$$w_i = \frac{RN_i}{\sum_{j=1}^{u} RN_j} \qquad (i=1, 2, \ldots, u) \qquad (2.26)$$

Where RN_i and RN_j are non-negative random integers. It was concluded that this multi-objective Genetic Algorithm performed better than the other six heuristic methods. An interesting challenge exists however the Hoffman enumeration procedure performed better than those genetic models.

Sabuncuoglu et al (2000) developed a fitness function (equation 2.21) to minimize the number of workstations taking in to account better balance of workstations. The cost function consists of two objectives. That is, minimizing the number of workstations and obtaining balanced stations. The first part of the fitness function aims to find the best balance among the solutions that have the same number of stations in the solution assuming the first objective is more critical than the second.

2.2.3.4 THE POPULATION

The classic Genetic Algorithm developed by Holland (1975) is what has become known as the *generational Genetic Algorithm*, keeps two populations most of the time: the current population and the future one.

In the generational Genetic Algorithm, each iteration (generation) proceeds by constructing the new population through genetic operators application on to chromosomes in the current population, and then the populations are swapped, the population just constructed becomes the current population in the next generation.

The chromosomes in the new generation come from three sources: some are produced by recombination (i.e. crossover); a few are produced by crossover and mutation. The rest, if necessary, is simply copied unchanged from the current population (elitism). In some implementations the mutation is applied after the crossover and reproduction to chromosomes selected at random from the new population. Figure 2.14 illustrates this process.

The other Genetic Algorithm model is called *steady state*. It always keeps only one population. The generations proceed by modifying some of its members. The offspring obtained by performing crossover on the best chromosomes in the population and replacing the worst ones. This process is shown in figure 2.15. The order is obtained by a selection technique (roulette wheel, tournament etc.). The other operators (mutation and inversion) are applied to chromosomes selected at random from the resulting population.

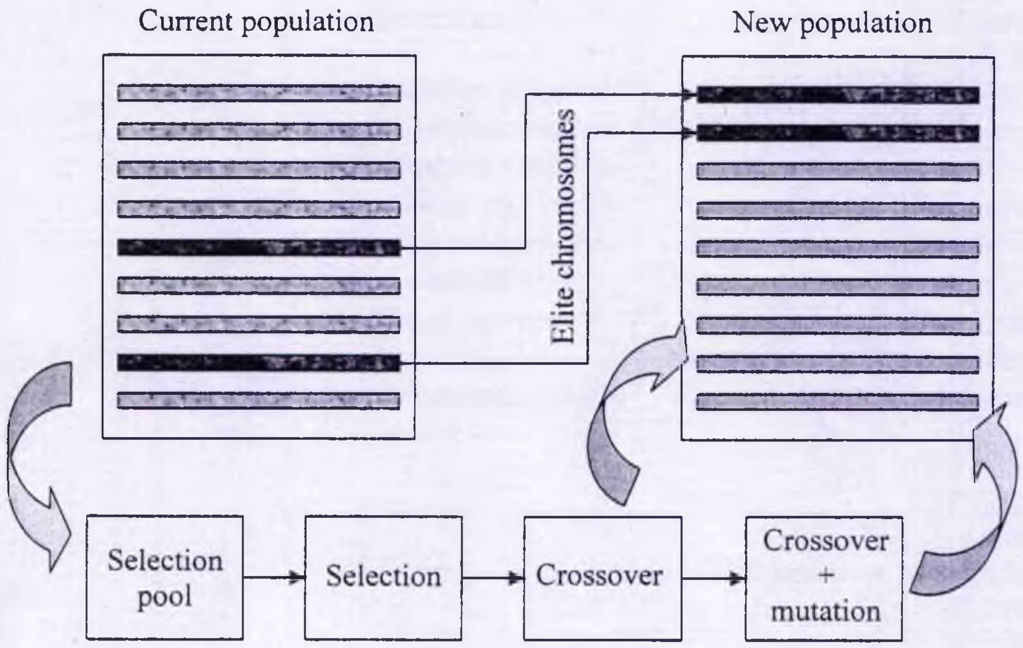


Figure 2.14. Generational Genetic Algorithm (Holland 1975)

The maximum number of individuals that can be generated by crossover in the steady state Genetic Algorithm is half the population size. That is the maximum crossover rate is 0.5. Moreover, since at least half a current population is carried over into the next generation, the maximum reproduction rate is 0.5 (except for mutation). The main advantage of the steady state Genetic Algorithm over orthodox generational model is that the steady state model reduces the memory requirement.

Syswerda (1989) and Falkenauer (1992) both used the above steady state Genetic Algorithm for their models. Syswerda used proportional selection technique to select parents, and always performed crossover on and replaced *just two* chromosomes at each generation, and always applied the mutation to the new chromosomes. However, Falkenaure used the tournament selection method to obtain parents.

The size of the population to use for a Genetic Algorithm has always been an interesting question for researchers. It is clear that the more the chromosomes in the population the faster the convergence. Maintaining a larger sample of the search space will improve the chance of finding the regions containing the best solutions, without being misled by local optima. However, larger populations require larger processing time on evaluating the fitness function and applying the genetic operators.

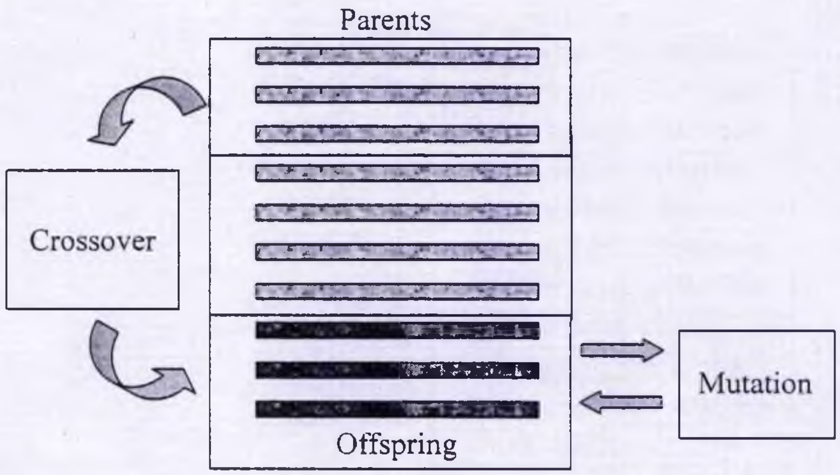


Figure 2.15. Steady state Genetic Algorithm

A small population would lead to high sampling errors and could more easily lead to premature convergence. The studies done by Goldberg (1989b) indicated that the size of the population in a Genetic Algorithm using binary strings grows exponentially with the length of the chromosome.

In parallel Genetic Algorithms, (one chromosome per processor, as in Talbi and Bessiere (1991)) fitness function evaluations and the operator applications are performed in parallel. In this case, the bigger the population, the better.

2.2.3.5 INITIALISATION

At the beginning of optimisation, a Genetic Algorithm requires a group of initial solutions. Anderson and Ferris (1994) mentioned the performance of the Genetic Algorithm scheme is not as good from the reselected starting population as it is from a random start. There are two basic ways of generating this initial population. The first consists of using randomly produced solutions created by a random number generator. This method is preferred for problems about which no prior knowledge exists or when assessing the performance of an algorithm.

The second method employs a priori knowledge about the given optimisation problem. Using this knowledge, a set of requirements is obtained and solutions, which satisfy those requirements, are collected to form an initial population.

Appendix B summarizes the previous initial populations used with published Genetic Algorithms for line balancing problems. In every model, it consists of a population containing randomly generated solutions plus a few solutions generated by heuristic techniques. Anderson and Ferris (1990) included a solution generated by the COMSOAL method, which was developed by Arcus (1966). Suresh et al (1996) introduced a modified Genetic Algorithm working with two populations, one of which consisted of infeasible solutions, and the other containing both random solutions and few solutions generated by a heuristic technique, exchanging specimens between populations at regular intervals.

2.2.3.6 GENETIC OPERATORS

In the classical Genetic Algorithm there are two basic genetic operators: selection and reproduction, which can be further divided into crossover, mutation and inversion. Some of these operators were inspired by nature and, in the literature, many versions of these can be found. It is not necessary to employ all these operators in a Genetic Algorithm because each one functions independently of the others. The choice or design of operators depends on the problem and the representation scheme employed. For example, operators designed for binary strings cannot be directly used on chromosomes coded with integers or real numbers.

2.2.3.6.1 *Selection*

The main goal of the selection procedure is to reproduce more copies of chromosomes whose fitness values are higher. In a Genetic Algorithm, the selection is based on the natural law of the survival of the fittest among the chromosomes. It has a significant influence on driving the search towards a promising area and finding good solutions.

There are two parameters associated with selection schemes (Whitley, 1989): *selective pressure*, the probability of the best chromosome being selected compared to the average probability of selection of all chromosomes, and *population diversity* which is the portion of chromosomes of a population that is selected during the selection phase. These two parameters have great influence on the performance of the Genetic Algorithm, and a good selection scheme must have a balance between these two. If selective pressure is too great, the population diversity decreases and this may result in a premature convergence. Weak selective pressure makes the search ineffective. Cavicchio (1970) adopted an innovative mechanism, the so-called pre-selection scheme, to maintain population diversity and a similar scheme was used later by De Jong (1975) in an optimisation study.

Since 1975, a number of selection schemes have been suggested for Genetic Algorithms. Some of the techniques relevant to the assembly line balancing problem are described in the following sections.

(a) Roulette wheel selection

The simplest selection scheme is the roulette-wheel or proportional selection proposed by Holland (1975). Its mechanism is based on the operation of a roulette wheel and runs as follows.

The chromosomes are first mapped to roulette wheel slices, such that each chromosome's slice is equal in size to its fitness. A random number is generated and the individual whose slice contains the random number is selected. The process is repeated until the desired number of individuals is obtained. Table 2.6 shows the selection probability and cumulative probability for 11 individuals. Chromosome one is the fittest chromosome and occupies the largest slice, whereas chromosome ten as the second least fit chromosome has the smallest slice in the roulette wheel (figure 2.16). Chromosome 11, the least fit slice, has fitness value zero and gets no chance for reproduction.

Number	Fitness Value	Selection Probability	Cumulative probability
	f(x)	$\frac{f(x)}{\sum f(x)}$	
01	2.0	0.18	0.18
02	1.8	0.16	0.34
03	1.6	0.15	0.49
04	1.4	0.13	0.62
05	1.2	0.11	0.73
06	1.0	0.09	0.82
07	0.8	0.07	0.89
08	0.6	0.06	0.95
09	0.4	0.03	0.98
10	0.2	0.02	1.00
11	0.0	0.00	1.00
Total	11		

Table 2.6. Fitness values and selection probabilities

The following algorithm can be used to simulate the roulette wheel process:

1. Calculate the cumulative probability of each chromosome;
2. Generate a random number (N_r) between zero and one;
3. Go through the population and the cumulative probabilities from zero to one, find the chromosome in which the cumulative probability is equal or just greater than N_r. Stop and return the chromosome where you are.

Step 1 is performed only once for each population.

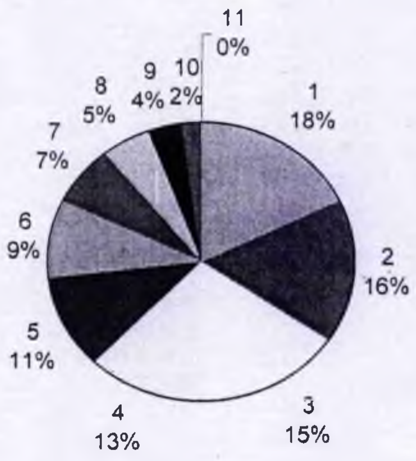


Figure 2.16. Roulette Wheel

One of the drawbacks of roulette wheel selection is that extraordinary chromosomes would take a significant portion of the roulette wheel resulting premature convergence. Davis (1991) overcame this problem by ranking or linear normalizing. It gives all the chromosomes a better chance to be selected.

(b) Rank-based fitness assignment

The nature of scaling procedures associated with classical roulette wheel selection led Baker (1985) to consider a nonparametric procedure for selection called rank-*based fitness assignment.* In this technique, the population is sorted according to the fitness values. Chromosomes are then assigned a number dependant only its position in the chromosome's rank and not on the actual fitness value. Figure 2.17 shows one of the ways Baker allocated trials according to rank.

Rank-based fitness assignment overcomes the scaling problems of the proportional fitness assignment. (Stagnation in the case where the selective pressure is too small or premature convergence where selection has caused the search to narrow down too quickly.) Since the reproductive range is limited, no chromosomes generate an excessive number of offspring. Ranking introduces a uniform scaling across the population and provides a simple way of controlling selective pressure.

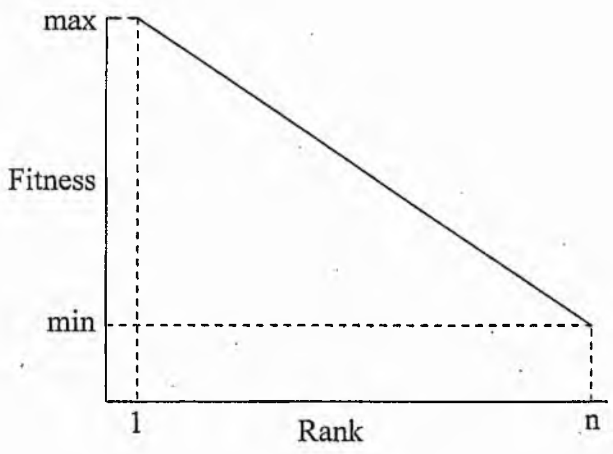


Figure 2.17. Fitness assignment mechanism in sorted selection scheme (Baker, 1985)

2.2.3.6.2 Reproduction

The Genetic Algorithm can be roughly described as proceeding from one population to another, the new population being obtained by application of *crossover* and *mutation* to the chromosomes in the current population. This section describes more about these two operators and their respective roles in Genetic Algorithms. Since the actual implementation of each of these operators can vary widely from one Genetic Algorithm to another, they are discussed under two sections: the classic operators and those proposed for assembly line balancing applications.

(a) Crossover

Crossover is the genetic operator that combines (mates) two chromosomes (parents) to produce new chromosomes (children). The idea behind crossover is that the new chromosomes may be better than both of the parents if it takes the best characteristics from each parent. Generally, crossover occurs during evolution according to a user-definable crossover probability.

Crossover probability (P_c) defines how often crossover will be performed. If $P_c = 0$, then there is no crossover and the entire new population is made from exact copies of chromosomes from the old population. When $Pc > 0$, a part of the new population is formed by crossover and if the crossover probability is one, then all the new offspring are made by crossover.

The Classic Crossover

There are two classic crossover techniques: single point crossover and two-point crossover. They are the oldest crossover techniques, having been presented and studied by Holland (1975).

1. Single-point or one point crossover

The simplest crossover is called the one-point crossover. Frantz (1972) defined this generalized, single-parameter crossover operator in his study of positional non-linearity. As the name implies, a crossover point is randomly selected and then the two chromosomes are interchanged at this point to produce two new offspring. Figure 2.18 illustrates this process.

2. Two-point crossover

Cavicchio (1970) defined two-point crossover operator. In this operation, two crossover points are selected randomly and then, the genes between successive crossover points are exchanged between the two parents to produce two new offspring (figure 2.19). The segment between the first gene and the first crossover point is not exchanged between chromosomes. The disruptive nature of two point crossover appears to encourage the exploration of the search space, rather than favouring the convergence to highly fit chromosomes early in the search, thus making the search more robust.

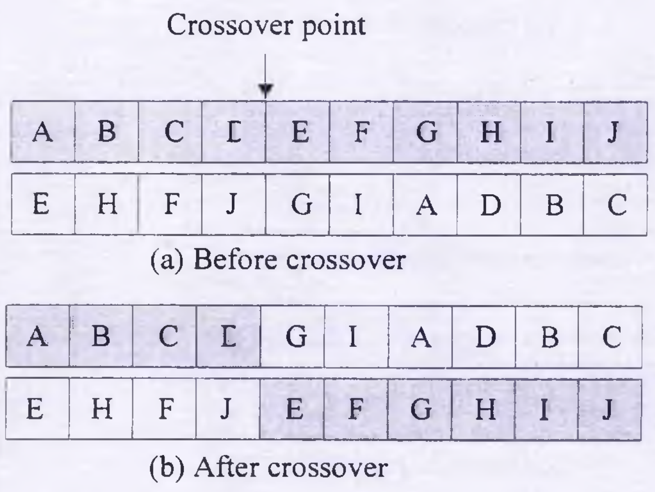


Figure 2.18. Single point crossover

It is difficult to make a firm comparison of the effect of two-point crossover against single-point crossover. The experience with Genie (Chambers, 1995) suggests that there is not much to choose between them. However, two-point crossover is more disruptive on longer chromosomes, since the segment being swapped is longer, and more likely to be substantially different (Frantz, 1972, De Jong, 1975).

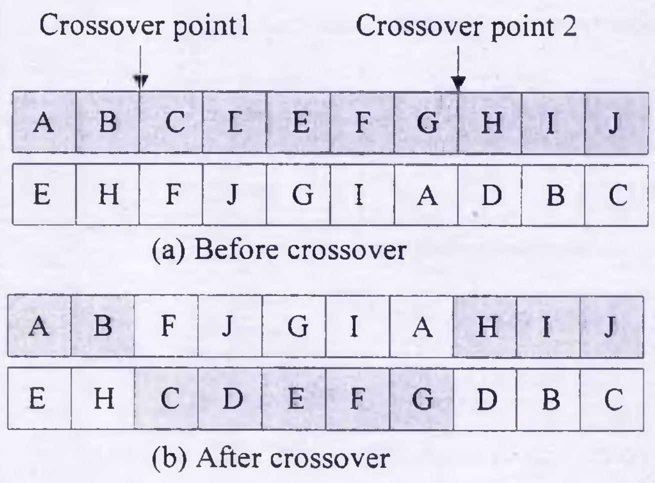


Figure 2.19. Two-point crossover

Good performance of a Genetic Algorithm requires the correct choice of Crossover probability (P_c). De Jong (1975) also experimented with crossover probabilities and generation gap values. As a result of these studies he suggested a crossover probability $P_c = 0.6$ as a reasonable compromise between good-on-line and off-line performance; later studies by Mercer (1977) and Grefenstette (1986) suggested that higher crossover rates ($0.9 > P_c > 0.8$) are better with more accurate selection procedures.

Crossover Operation For Assembly Line Balancing Applications

In addition to the classic genetic operators, a number of crossover techniques have been developed for assembly line applications. Appendix B lists some the new techniques and their applications to the assembly line balancing problem and are described in this section.

1. *Partially Mapped Crossover (PMX)*

This operation was suggested by Goldberg and Lingle (1985) and is aimed at maintaining inheritance of adjacency and relative order of elements in the solution structure. First, all the elements from parent 1 are copied to the same positions of the child; then using pair wise exchange some elements of the child are being relocated in order to make a random fragment of child to be an exact copy of the same fragment of parent 2 (figure 2.20).

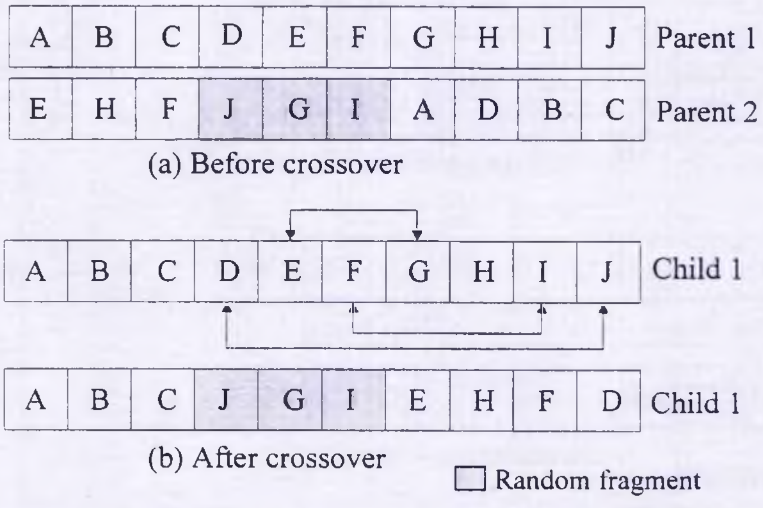

A B C D E F G H I J Parent 1
E H F J G I A D B C Parent 2
(a) Before crossover
A B C D E F G H I J Child 1
A B C J G I E H F D Child 1
(b) After crossover
□ Random fragment

Figure 2.20. Partially Mapped Crossover (PMX) (Goldberg and Lingle, 1985)

2. Order crossover (ORD)

Davis (1985a) defined the order crossover operator. This operator preserves the order, adjacency and absolute positions of part of the elements and the relative order of the remaining elements. A child chromosome inherits elements of a random fragment from parent 1, in the same order and position. The remaining elements are inherited from parent 2 in the same order they appear in parent 2, beginning with the first position following the end of the random fragment and skipping over all elements already present in the child. Figure 2.21 illustrates this operation.

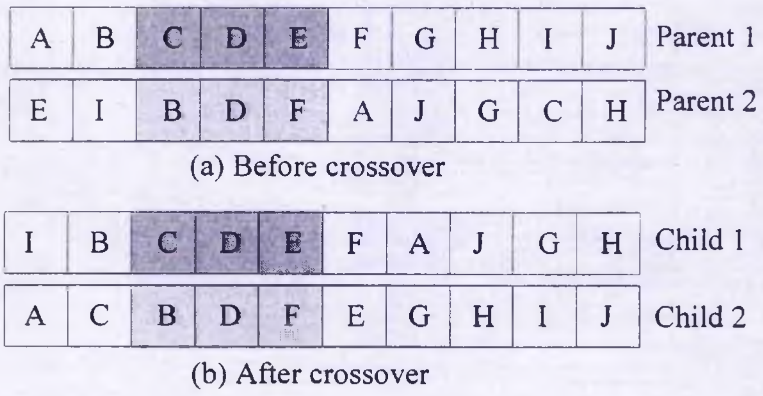

A B C D E F G H I J Parent 1
E I B D F A J G C H Parent 2
(a) Before crossover
I B C D E F A J G H Child 1
A C B D F E G H I J Child 2
(b) After crossover

Figure 2.21. Order crossover (Davis, 1985a)

3. Position based crossover (POS)

This crossover procedure was proposed by Syswerda (1989) and is intended to preserve inheritance of positional information. A randomly chosen number of locations are selected in parent 1 and the child chromosome inherits the elements in these positions. The remaining elements are inherited in the order, in which they appear in parent 2, skipping over all elements already included into the child chromosome (figure 2.22). Although this operator is similar to order crossover (except the requirement of adjacency of elements being copied from parent1), Staekweather et al (1991) showed that it has significantly different properties.

4. Cycle crossover (CYC)

Oliver et al (1978) suggested this crossover technique. It allows inheriting the absolute positions of elements from parent structures. The Starting element of parent 1 (first or randomly defined) is inherited by a child. The element, which is in the same position in parent 2, cannot consequently be placed in this position. The position of this element is found in parent 1, and the element is inherited by a child to the same position. This continues until the cycle is completed by encountering the initial element from parent 1 in parent 2. All elements, which were not yet copied to the child, are copied from the same positions of parent2.

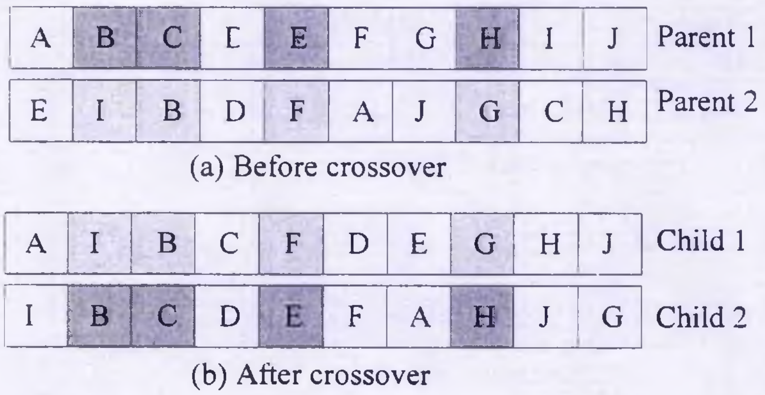

A B C D E F G H I J Parent 1
E I B D F A J G C H Parent 2
(a) Before crossover
A I B C F D E G H J Child 1
I B C D E F A H J G Child 2
(b) After crossover

Figure 2.22. Position based crossover (POS) (Oliver et al., 1987)

5. *Fragment reordering crossover (FRG)*

This procedure was proposed by Rubinovitz (1995) particularly for assembly line balancing problems. All the other crossover operators, when applied on the assembly line balancing solution vector, results in loss of feasibility of the offspring structure. The fragment reordering crossover operator is aimed at maintaining the inheritance of positions and the relative order of elements in the structure, providing changes within the fragment, which do not violate the precedence constraints.

The fragment reordering crossover procedure may be considered as a special case of position based crossover and reversed version of order crossover. First, all the elements from parent 1 are copied to the same positions of a child chromosome. Then all the elements of a random fragment in the child chromosome are reallocated within the fragment according to the order in which they appeared in parent 2. Precedence relations within the fragment are inherited from parent 2. Consequently, if parent 1 and parent 2 are feasible sequences, the child inherits this feasibility.

Rubinovitz compared this technique with a selection of five crossover methods used in previous studies of Genetic Algorithms (Goldberg, 1985, Davis, 1985, Oliver et al, 1978, Syswerda, 1990, Starkweather et al, 1991) and concluded that cycle crossover and fragment reordering crossover procedures dominated.

In addition to the above crossover operators, a number of other crossover techniques have been defined in the literature including *uniform crossover* developed by Syswerda (1989), *heuristic structural crossover (HSX)* suggested by Kim et al (1998) based on the structural crossover (SX) proposed by Laszewski (1991) and the most recent method called *dynamic partitioning (DPA)* technique developed by Sabuncuoglu et al (2000). Appendix B lists a selection of the crossover techniques applied for assembly line balancing models. Other than the specific crossover technique developed by researchers for their models, both single point and two point crossover methods are generally employed.

(b) Mutation

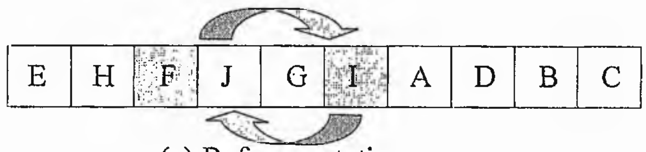
Mutation is a genetic operator that alters one or more genes in a chromosome from its initial state. This can result in entirely new genes being added to the gene pool. With these new genes, the Genetic Algorithm may be able to arrive at a better solution than was previously possible. Mutation is an important part of genetic search as it helps to prevent populations from stagnating at any local optima. Mutation occurs during evolution according to a user-specified mutation probability.

Mutation probability (P_m) defines the probability of mutation of chromosome. If there is no mutation, offspring are formed by crossover or copy without any change. If mutation is performed, part of the chromosome is changed. One hundred percent mutation probability means the whole chromosome is changed and zero percent indicates no change at all. Mutation is generally used to prevent a Genetic Algorithm falling into local extrema, but it should not occur very often, because then a Genetic Algorithm will in fact change into a random search.

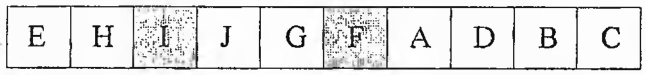
A number of mutation techniques are available and described following for assembly line balancing problems.

Classic Mutation

This is the smallest possible random modification of a chromosome. According to the mutation probability, one or more pairs of genes are selected randomly and they are swapped to produce new offspring. Figure 2.23 illustrates this process and genes F and I are randomly selected and they are swapped to produce the new chromosome.



(a) Before mutation



(b) After mutation

Figure 2.23. Classic mutation

Mutation Operations For Assembly Line Balancing

1. Leu et al (1994) suggested the scramble mutation. A random cut-point is selected and the genes after the cut-point are randomly replaced (scrambled), whilst assuring feasibility.

2. Tsujimura (1995) defined the following mutation operator as follows:
    a. Generate an integer number p in the range [1, (n/m*)] randomly. Where n is the number of genes in the chromosome and m is the theoretical minimum number of workstations.
    b. Generate randomly a position *pos* in the chromosome.
    c. Replace the element at the position *pos* within the defined neighbourhood which is within [*pos-p* , *pos+p*].

       Example:

       Chromosome = [A B C D E F G] n= 10 m = 3 p= [1...3]

       Generate *pos* and p randomly. *pos* =3 and p = 2.

       [A B **C** D E F G] → [**A** B C D E F G] or [A B C D **E** F G]

3. Kim et al (1998) developed a mutation technique called Heuristic Structural Mutation (HSM) that randomly chooses some genes according to the mutation rate and marks those genes for reassignment. The reassignment is performed by the same adaptation procedures as used in the Heuristic Structural Crossover. According to the table in appendix B, The classic mutation technique is the one most employed in line balancing models and mutation probability was set to fairly low values. (Between .01 and 0.03).

(c) Inversion

The third classic operator is inversion. It proceeds by inverting the order of genes on a randomly selected segment of the chromosome. This technique can be seen in classical Genetic Algorithms where chromosomes are represented by binary values. Consequently applications cannot be seen in assembly line balancing models because of the violation of precedence constraints.

2.2.3.7 ELITISM

Convergence of Genetic Algorithm solutions is one of the most challenging theoretical issues in evolutionary computation. Several researchers have explored this problem from different perspectives. Rudolph (1994) proved that a classical Genetic Algorithm never converges to the global optimum, but a modified version, which maintains the best chromosomes in the population, does. This is because, when creating new populations by crossover and mutation, there is substantial chance, that the best chromosome may be lost. Elitism is the name of a method which first copies the best chromosome (or a few best chromosomes) to the new population. The rest of the population is generated in a classical way. Elitism rapidly increases the performance of the Genetic Algorithm.

2.2.3.8 TERMINATION

Termination is the criterion by which the Genetic Algorithm decides whether to continue or stop searching. Researchers have developed several termination criteria and each of the termination criteria are checked after each generation to see if it is time to stop. A selection of termination criteria include:

1. *Generation number.* This method stops the evolution when the user specified maximum number of evolutions have been run and it is always active.

2. *Evolution time.* A termination method that stops the evolution when the elapsed evolution time exceeds the user-defined maximum evolution time. By default, the evolution is not stopped until the evolution of the current generation has completed, but this behaviour can be changed so that the evolution can be stopped within a generation.

3. *Fitness threshold.* A termination criterion that stops the evolution when the best fitness in the current population becomes less than the user specified fitness threshold and the objective is set to minimize the fitness.

4. *Population convergence.* A termination method that stops the evolution when the population is deemed as converged.

As far as assembly line balancing applications are concerned, generation number criterion is used as the termination method. (Appendix B).

2.2.4 GENETIC ALGORITHM PROGRAMMING ENVIRONMENTS

Following Holland's original Genetic Algorithm concept, many variations of the basic algorithm have been introduced. However, the important and distinctive feature of all Genetic Algorithms is the *population handling* technique. The original Genetic Algorithm adopted a Generational replacement policy (Davis, 1991) and later many subsequent Genetic Algorithms used the *Steady-State* policy (Davis, 1991). Filho et al (1993) reviewed software environments for programming Genetic Algorithms and classify them into three main categories as follows.

1. *Application-oriented systems*: they are designed for use by business professionals who wish to utilize Genetic Algorithms in specific application domains, without having acquired detailed knowledge of the working of Genetic Algorithms. PC/BEAGLE is software developed by Forsyth (1989) supporting scheduling, telecommunication etc. The OMEGA Predictive Modelling System is a powerful predictive m odel e xploiting a dvanced G enetic A lgorithms t echniques f or u se i n the financial domain.

2. *Algorithm-specific systems*: They embody a single powerful used by developers requiring a general-purpose Genetic Algorithm for their applications and researchers interested in the development and testing of a specific algorithm and genetic operators. The most well known system in this category is the pioneering GENESIS (Davis, 1991, Grefenstette, 1981, Grefenstette, 1987), which has been used to implement and test a variety of new genetic operators in Europe. GENITOR developed by Whitley (1989), is another well-known software package in the Genetic Algorithm field, containing examples of floating-point, integer and binary representations.

3. *General-purpose systems*: these are the ultimate in flexible Genetic Algorithm programming systems. Not only do they allow the user to develop their own Genetic Algorithm a pplications a nd a lgorithms, but a lso p rovide u sers with t he opportunity to customise the system to suit their own purposes. These systems provide a comprehensive tool kit, including: a sophisticated graphic interface; a high level language for programming Genetic Algorithms and an open architecture. EnGENEer (Robbins, 1992) and GAME (Alippi and Treleaven, 1991) are two substantial examples.

2.2.5 SUMMARY

Anderson and Ferris (1990) pioneered the application of Genetic Algorithms for the assembly line balancing problem. Since then, a number of Genetic Algorithm models have been developed to address this problem by modifying the classical Genetic Algorithm proposed by Holland (1975). These modifications include the development of new fitness functions, genetic operators (crossover and mutation techniques) and selection methods.

The first cost function developed by Anderson and Ferris (1990) was very complex and it consisted of a number of parameters and constants. But, the fitness functions suggested in the late 1980s seemed to be simple and most of them are based on the assembly line balancing performance measures (Line efficiency, smoothness index, balance delay, mean absolute deviation). Subsequent every published has introduced a new fitness function, and some papers have suggested new genetic operators and selection techniques in addition to a cost function.

A number of crossover techniques have been developed besides the classic genetic operators and several repair techniques being introduced to maintain the feasibility of the new offspring. The necessity of propagating the best portion of a chromosome during genetic operation has been highlighted by several researches for better performances of the algorithm (Falkenauer, 1992 and Davis et al, 1991). No crossover technique has been developed to address the issue.

The classic roulette wheel selection has been the most employed selection scheme for many Genetic Algorithm models. The tournament selection and the rank-based selection schemes are employed in a few cases. High selective pressure is a major problem in all the selection techniques and several procedures are proposed to eliminate this problem and introduce diversity to solutions.

The performance of the Genetic Algorithm depends upon a number of key factors, including population size, initial population, crossover and mutation probabilities, number of elite chromosomes, and problem complexity. Developing a general-purpose Genetic Algorithm model with a more efficient selection of these factors appropriate for the assembly line balancing problem, is the challenge within this work. From the review of existing literature, acknowledging a specialist interest in assembly line balancing, a new Genetic Algorithm approach improving fitness function and genetic operators is identified as the way forward.

Several researchers have claimed that their techniques outperformed the existing heuristic techniques but have not finally yielded the optimum solution. In many cases, the solution obtained by the Hoffman matrix procedure seemed to be better than those obtained by the Genetic Algorithm (Ponnambalam et al 2000). It seems that the current research on the application of Genetic Algorithms to the assembly line balancing problem is promising and not too far from the end goal. The next chapter describes the new Genetic Algorithm model developed to address some of the above problems by introducing a new fitness function, crossover and mutation techniques and a modified rank-based selection scheme to deal with high selective pressure problems.

CHAPTER 3

A GENETIC ALGORITHM LINE BALANCING MODEL

The previous chapter reviewed advantages and limitations of existing Genetic Algorithm models and indicated the potential for line balancing. The design details of the new Genetic Algorithm model for the single model deterministic assembly line-balancing problem are presented in this chapter. The model consists of a new fitness function, a modified selection scheme, novel genetic operators for crossover and mutation and a repair technique. The new fitness function, which is the key component of the model, introduces and uses a front-loading theorem, and described at the beginning of the chapter.

3.1 THE FRONT-LOADING THEOREM

Since the single model assembly line balancing problem is a combinatorial optimisation problem, there exist a large number of feasible solutions. Therefore, in the majority of the instances, finding optimal solutions within a polynomial time is very hard. However, the front-loading theorem describes a theoretical approach for yielding optimal or near optimal solutions in polynomial time.

> *Theorem: an optimal solution or theoretically minimum number of workstations (m^*) can be found by packing workstations progressively (workstation w_{j+1} after w_j for j from 1 to m^*-1) to their full capacity.*

Proof: Case1 – At least one perfectly balanced solution exists

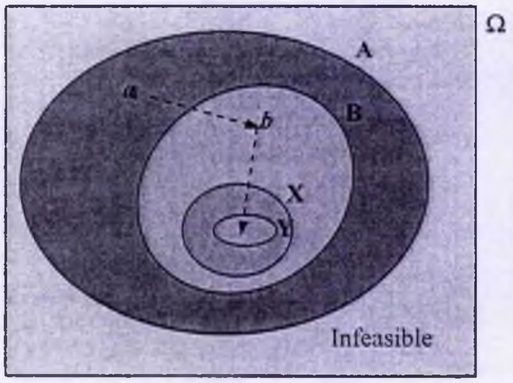


Figure 3.1. Solution domain

Let x (x = {X | perfectly balanced solution(s)}) be the perfectly balanced solution(s), and they are represented by the set X. Similarly, let y (y = {Y | fully front-loaded solutions (figure 3.2(y)}) be the theoretically fully front-loaded solution(s), representing the set Y. To explain the theorem, consider an arbitrary solution ***a*** (figure 3.1) in the feasible set A (a = {A | a solution with number of workstations $\leq m^*+2$}). Where m* is the theoretical minimum number of workstations. Moving elements progressively from latter workstations to early workstations reduces the number of workstations and results in the solution leaping from set A to set B (b = {B | a solution with number of workstations $\leq m^*+1$}) and eventually to set Y. The station time distribution of these solutions are represented by ***a, b,*** and ***y*** respectively in figure 3.2. The solution ***a*** consists of three fully packed workstations (1, 4 and 5). The progressive loading process improves this solution by packing four workstations (1, 2, 3 and 5) including the first three, and then further loading results in the perfect balance where all the stations are fully packed to capacity.

Therefore, if there is at least one optimum solution;

$$Y = X$$

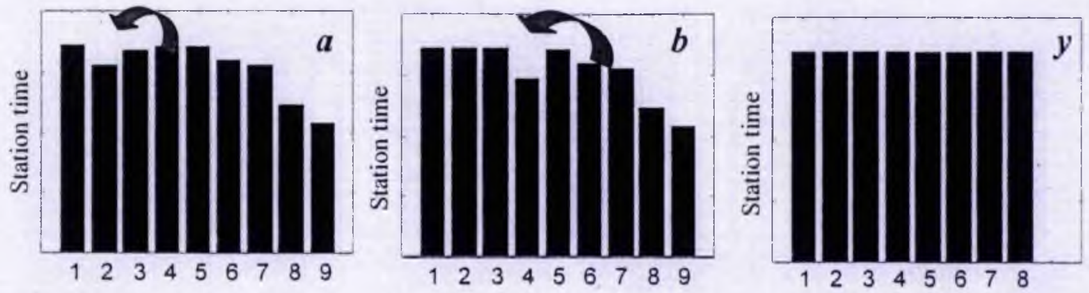


Figure 3.2. Progressive loading of workstations: solution (***a***) first station is packed; solution (***b***) first three stations are packed; solution (*y*) all the stations are fully packed and total number of stations equal to eight.

Proof: Case2- A perfectly balanced solution does not exist

In instances where there is no perfectly balanced solution(s), solutions yielding theoretically minimum number of workstations are considered as optimal solutions. Generally, several such solutions exist, and let them be represented by, *d* (*d* ={D| solutions with theoretically minimum number of workstations (*m**)}). In such cases, it is obvious that, the theoretically fully front-loaded solution(s) is one of them. Therefore;

$$Y \subseteq D$$

Generally, generating a fully front loaded solution from a single pass heuristic decision rule may not be possible. However, a technique evaluating multiple solutions with back tracking incorporated could easily slip into this solution. Driscoll et al (2002) verified the above theorem and the possibility of generating front loaded solutions by developing a multi-pass heuristic technique for simple assembly line balancing problems. This technique is a modification of the original Hoffman (1963) procedure, attempting to fill front workstations as early as possible by considering many feasible combinations at sub-workstation level. The new fitness function is defined based on this concept and the complete design procedure is described next.

3.2 DESIGN OF THE FRONT-LOADING FITNESS FUNCTION

The fitness function is the key component in the Genetic Algorithm and the only link between the algorithm and the actual problem being solved. The fitness function distinguishes the better and the worst chromosomes in the population and provides an important feed back for the search process. This section illustrates the development of the front-loading mathematical model.

Solving the assembly line balancing problem involves searching for feasible sequence(s) ending up with a minimum number of workstations. Throughout the searching, the algorithm assesses both feasible and infeasible chromosomes and at any stage of the evolution process, a population could consist of some feasible chromosomes (*a*, *b*, *c*) and infeasible chromosomes (*p*, *q*, *r*), whilst the global optimum solution is *x* (figure 3.3).

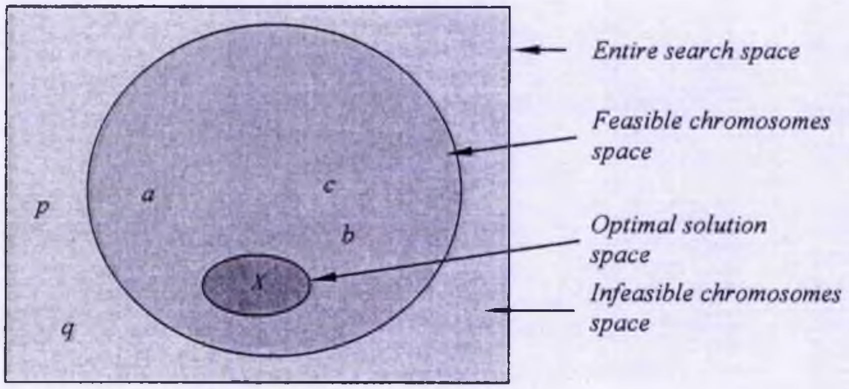


Figure 3.3. A search space and its feasible and infeasible parts

The design of the fitness function answers the question of how to evaluate both feasible and infeasible chromosomes. It is almost certain that these feasible and infeasible chromosomes in the population influence the other parts of the Genetic Algorithm; some genetic operators might be applicable to feasible chromosomes only. However, many search algorithms start with a population containing both feasible and infeasible chromosomes. Infeasible chromosomes allow the system to explore the search space in different directions.

There are two basic ways of designing fitness functions: developing a single fitness function to evaluate both feasible and infeasible chromosomes or designing two separate independent evaluation functions and establish some relationship between them. The second option was employed in this research and the design concepts for evaluation of infeasible and feasible is described in the following sections.

3.2.1 FITNESS FUNCTION DESIGN FOR INFEASIBLE CHROMOSOMES

The issue of handling infeasible chromosomes and extending a fitness function to evaluate infeasible chromosomes is a difficult problem and has been debated for years. Since these chromosomes are adding diversity to the search process, their contribution is vital and should not be discarded. Powell and Skolnick (1993) and Michalewicz and Xiao (1995) reported good results of their evolutionary algorithms, which worked under the assumption that any feasible chromosome is better than an infeasible solution and the proposed fitness function is designed based on the same assumption.

The total number of feasible links in a chromosome is a good measure of comparing two infeasible chromosomes. The higher the number of links the better the chromosome and the closer it is to a feasible chromosome. Therefore, it is used to evaluate infeasible solutions and the procedure for finding the number of feasible links is described here.

Consider a chromosome shown in figure 3.4 having *n* elements. The elements in the chromosome are represented by e_i (i = 1 to n).

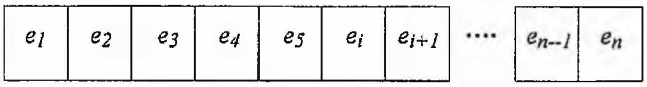


Figure 3.4. Chromosome representation

Procedure:

1. Select an element e_i from the chromosome.
2. Find all the elements (e_j) that immediately succeed the element selected in step 1 from the corresponding precedence diagram (j=1 to n_p). Where n_p is the total number of immediately succeeding elements.

3. If the locus of the e_j in the chromosome is to the right of e_i, this link is considered as a feasible link. Repeat this step for all elements e_j (j=1 to n_p) and record the total number of feasible links corresponds to each element e_i.
4. Repeat steps 1 to 4 for all the elements in the chromosome (i=1 to n).

The following illustration shows the application of the above procedure. The selected feasible and infeasible chromosomes and the precedence are shown in figure 3.5. The corresponding precedence matrix of the test problem shown in figure 3.5 is given in figure 3.6. The immediate successors of a selected element (step2) are found using the precedence matrix in which, entries in the first column (i) are the task element and, in the corresponding row, the elements heading the column (j) in which are ones (1), are the immediate successors.

A	C	B	E	D	F	G	H	I	J

(a) Feasible chromosome

E	A	C	B	F	D	G	H	I	J

(b) Infeasible Chromosome

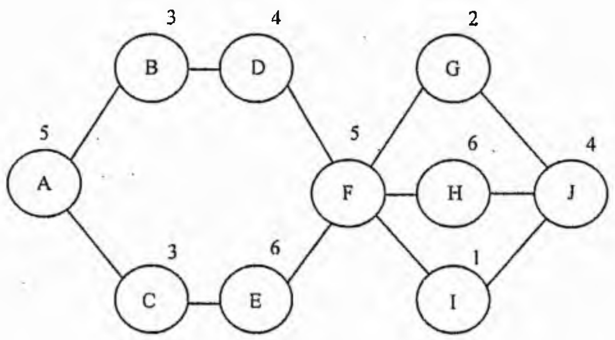


Figure 3.5. Feasible and infeasible chromosomes and the precedence network

Table 3.1 and 3.2 are prepared to find the total number of feasible links for the feasible and infeasible chromosomes respectively, for the example shown in figure 3.5. The first column indicates the element locus (position) and the second shows the corresponding element. The elements to the right of the selected element are shown in the third column and, the immediate successors and number of feasible links are shown in the fourth and last columns respectively.

i \ *j*	A	B	C	D	E	F	G	H	I	J
A	-	1	1	0	0	0	0	0	0	0
B		-	0	1	0	0	0	0	0	0
C			-	0	1	0	0	0	0	0
D				-	0	1	0	0	0	0
E					-	1	0	0	0	0
F						-	1	1	1	
G							-	0	0	1
H								-	0	1
I									-	1
J										-

Total number of links = 12

Figure 3.6. Precedence matrix of the above network

For a chromosome to be feasible, the total number of feasible links (*l*) must be equal to the total number of precedence relationships (*P*). The chromosome shown in figure 3.5(a) satisfies this condition and therefore, it is a feasible chromosome. However, for the chromosome shown in figure3.5 (b) the number of feasible links (*l* = 10) is less than *P* (=12) and therefore, it is not a feasible chromosome.

Locus	e_i	Elements	Successors (e_j)	Number of feasible links
1	A	***CB***EDFGHIJ	BC	2
2	C	B***E***DFGHIJ	E	1
3	B	E***D***FGHIJ	D	1
4	E	D***F***GHIJ	F	1
5	D	***F***GHIJ	F	1
6	F	***GHI***J	GHI	3
7	G	HI***J***	J	1
8	H	I***J***	J	1
9	I	***J***	J	1
10	J	-	-	0
			Total	**12**

Table 3.1. Feasible links of chromosome 3.5(a)

Locus	e_i	Elements	Successors (e_j)	Number of feasible links
1	E	ACB*F*DGHIJ	F	1
2	A	***CB***FDGHIJ	BC	2
3	C	BFDGHIJ	E	0
4	B	F***D***GHIJ	D	1
5	F	D***GHI***J	GHI	3
6	D	GHIJ	F	0
7	G	HI***J***	J	1
8	H	I***J***	J	1
9	I	***J***	J	1
10	J	-	-	0
			Total	**10**

Table 3.2. Feasible links of chromosome 3.5(b)

For infeasible chromosomes, the number of feasible links (l) can be used design the fitness function (f_i) and, one having higher number of links receives higher fitness value than that of a lower number of links. Since the number of feasible links are less than or equal n, the fitness generated by the function always lies between one and n. This value is extremely low when compared to the fitness of feasible solutions. Therefore, a problem dependent constant is integrated to overcome this problem. So the complete fitness function for infeasible solutions is expressed as follows:

$$f_i = n^3\beta l \qquad \forall\ l < P \tag{3.1}$$

Where n is the number of elements in the problem and β is a test problem constant which will be defined in the next section.

3.2.2 FITNESS FUNCTION DESIGN FOR FEASIBLE CHROMOSOMES

The issue of comparing two feasible chromosomes is complicated. The number of feasible links cannot be used as the fitness function, because all feasible chromosomes generate the same fitness value, which is equal to n. Therefore, a different approach should be used to compare two feasible chromosomes. The proposed fitness function for feasible chromosomes (f_f) consists of three basic components X, Y, Z and is expressed as follows.

$$f_f = X + Y + Z \tag{3.2}$$

In order to generate higher fitness values for feasible frontloaded solutions than fully frontloaded infeasible solutions, terms X, Y and Z are designed as $\max(X) > \max(Y) > \max(Z)$.

3.2.2.1 THE COMPONENT X

This evaluates the feasibility of chromosomes. Feasible chromosomes receive higher fitness values than that of infeasible ones and thereby clearly distinguish them from infeasible chromosomes.

$$X = n^3 l\beta \tag{3.3}$$

$$\alpha = nCR^{m^*+1}$$

$$\beta = \frac{\alpha}{R}$$

Where l is the number of feasible links in the chromosome, n is the number of elements (tasks), C is the cycle time, α and β is constants that depend on the test problem and the cycle time. R is the front-loading constant described in section 3.2.2.4 and m^* is the theoretical minimum number of workstations.

3.2.2.2 THE COMPONENT Y

This takes into account the gradual (progressive) filling of workstations and preserves filled stations during the evolutionary process, and can be stated as:

$$Y = n^2\beta\frac{\sum_{j=1}^{K}\left(\frac{S_j}{C}\right)}{m - m^* + 1} \tag{3.4}$$

S_j = workstation time C = Cycle time j = workstation m = number of workstations

At the beginning of the evaluation, K=1 and if (S_j/C) = 1 or generation number (g) equals to G_w*K, the value of K is incremented by one unit. Where G_w is the number of generations permitted for a workstation and, is obtained by dividing the total number of generations (G) by the number of workstations in the initial heuristic solution.

3.2.2.3. THE COMPONENT *Z*

This is the key component of the proposed fitness function, which is based on the front-loading theorem. It forces the algorithm to shift elements from later to early workstations to accomplish fully filled workstations, and is mathematically expressed as follows:

$$Z = \alpha \frac{\sum_{j=1}^{m} \left(\frac{S_j}{C} \right)^k R^{m-j}}{mR^{m+1}} = \alpha \frac{\sum_{j=1}^{m} \left(\frac{S_j}{C} \right)^k R^{1-j}}{m} \tag{3.5}$$

Where *m* is the number of workstations, *C* is the cycle time, S_j is the workstation time and *k* is constant and the best choice of *k* will be covered later. The other constants are as defined before.

The front-loading concept is integrated in to the component *Z* by designing the fitness function to assign higher fitness values for early packed workstations than for late packed (R^{1-j} , where j=1,2,...*m*)). A fully packed first workstation ($S_1 = C$) will received the highest fitness weight (R^0) and this maximum is scaled exponentially to give higher weight to early workstations and lower values to later workstations (R^0, R^{-1}, R^{-2},......, R^{1-m}). The value *R* controls the degree of scaling, and it should be selected appropriately, which will be dealt with later in this section. Suffix *k* in the fitness function provides a family of fitness curves, but the following analysis verifies that when *k* equals to unity, shifting of an element to an early from latter workstation increments the fitness value to support front-loading (property 1). Furthermore, the decreasing number of workstations (*m*) increases fitness value (property 2, proof given in Appendix C)

Figure 3.7 shows a typical line balancing solution in the g^{th} generation. It consists of *m* workstations with station times S_a and S_b for workstations *a* and *b* respectively. Assume in the next generation (*g*+1), genetic operators have modified the chromosome representing the above solution by shifting a unit element from workstation *b* to workstation *a*. Z_g and Z_{g+1} give the corresponding fitness values before and after reproduction respectively.

$$Z = E\sum_{j=1}^{m}\left(\frac{S_j}{C}\right)^k R^{m-j} \tag{3.6}$$

$$\text{Where } E = \frac{\alpha}{mR^{m+1}}$$

From equation 3.6, fitness (Z) of the chromosome in the g^{th} generation is given as

$$Z_g = E\left(P + \left(\frac{S_a}{C}\right)^k R^{m-a} + Q + \left(\frac{S_b}{C}\right)^k R^{m-b} + T\right)$$

$$\text{Where } P = \sum_{i=1}^{x-(a+1)}\left(\frac{S_i}{C}\right)^k R^{m-i}, Q = \sum_{i=a+1}^{b-1}\left(\frac{S_i}{C}\right)^k R^{m-i}, \ T = \sum_{i=b+1}^{m}\left(\frac{S_i}{C}\right)^k R^{m-i}$$

After crossover and mutation operations, in the ($g+1$)th generation, the fitness of the chromosome is given by,

$$Z_{g+1} = E\left(P + \left(\frac{S_a+1}{C}\right)^k R^{m-a} + Q + \left(\frac{S_b-1}{C}\right)^k R^{m-b} + R \right)$$

Figure 3.7. Workstation time distribution

Therefore, change of fitness component *(ΔZ)* that is,

$$\Delta Z = Z_{g+1} - Z_g$$

$$\Delta Z = E\left(\left(\frac{S_a+1}{C}\right)^k R^{m-a} + \left(\frac{S_b-1}{C}\right)^k R^{m-b} - \left(\frac{S_a}{C}\right)^k R^{m-a} - \left(\frac{S_b}{C}\right)^k R^{m-b} \right)$$

$$\Delta Z = \frac{ER^m}{C}\left[\frac{1}{R^a}\left\{(S_a+1)^k - S_a^k\right\} + \frac{1}{R^b}\left\{(S_b-1)^k - S_b^k\right\}\right] \qquad (3.7)$$

Similarly, it can be shown that change of fitness component (ΔY) due to transferring elements from workstation b to a equals to zero and will have no effect on fitness change. Equally, the change of fitness component (ΔX) is zero too, because it is a constant for all feasible chromosomes and does not has any influence on transferring.

Therefore, total change of fitness is as follows:

$$\Delta f_f = \Delta Z \qquad (3.8)$$

It can be seen from the equation 3.7 that, ΔZ is a function of k and to find the optimum value, lets consider three k values including 1,2 and 0.5.

(a) Case 1

Substituting k =1 in equation (3.7);

$$\Delta f_f = \frac{ER^m}{C}\left[\frac{1}{R^a} - \frac{1}{R^b}\right] \qquad (3.9)$$

ΔZ is positive for all values $b > a$ subject to the condition that R>1. Therefore when k =1; shifting an element to an early workstation results in an increase in fitness component

(b) Case 2

Substituting k =2 in equation 3.7;

$$\Delta f_f = \frac{ER'''}{C^2}\left[\frac{1}{R^a}\left\{(S_a+1)^2 - S_a^2\right\} + \frac{1}{R^b}\left\{(S_b-1)^2 - S_b^2\right\}\right]$$

$$= \frac{ER'''}{C^2}\left[\frac{1}{R^a}\left(S_a^2 + 2S_a + 1 - S_a\right) + \frac{1}{R^b}\left(S_b^2 - 2S_a + 1 - S_b\right)\right]$$

$$= \frac{ER'''}{C^2}\left[\frac{1}{R^a}(2S_a + 1) + \frac{1}{R^b}(1 - 2S_b)\right] \qquad (3.10)$$

It can seen from the equation 3.10 that Δf_f is positive only if the following condition is satisfied.

$$\frac{2S_a + 1}{R^a} + \frac{1 - 2S_b}{R^b} > 0$$

This occurs only when $R^{b-a} > \frac{2S_b - 1}{2S_a + 1}$, and this condition is not true for all S_a and S_b. therefore, when k = 2, the proposed fitness function may violate the front-loading concept. Similarly, it can be proved that higher orders of k will introduce more constraints to the fitness function.

(c) Case3

Substituting k = 0.5 in equation 3.7,

$$\Delta Z = \frac{ER'''}{C}\left[\frac{1}{R^a}\left\{(S_a+1)^{\frac{1}{2}} - S_a^{\frac{1}{2}}\right\} + \frac{1}{R^b}\left\{(S_b-1)^{\frac{1}{2}} - S_b^{\frac{1}{2}}\right\}\right]$$

$$= \frac{ER'''}{C}\left[\frac{\sqrt{S_a}}{R^a}\left\{\left(1 + \frac{1}{S_a}\right)^{\frac{1}{2}} - 1\right\} + \frac{\sqrt{S_b}}{R^b}\left\{\left(1 - \frac{1}{S_b}\right)^{\frac{1}{2}} - 1\right\}\right]$$

$\left(1+\frac{1}{S_a}\right)^{\frac{1}{2}}$ and $\left(1-\frac{1}{S_b}\right)^{\frac{1}{2}}$ are infinite series when $\left(\frac{1}{S_a}\right)^2 < 1$ and $\left(\frac{1}{S_b}\right)^2 < 1$ respectively. Expanding and substituting in the above equation

$$= \frac{ER^m}{C}\left[\frac{1}{2}\left(\frac{1}{R^a S_a^{\frac{1}{2}}} - \frac{1}{R^b S_b^{\frac{1}{2}}}\right) - \frac{1}{8}\left(\frac{1}{R^a S_a^{\frac{3}{2}}} - \frac{1}{R^b S_b^{\frac{3}{2}}}\right) + \frac{1}{16}\left(\frac{1}{R^a S_a^{\frac{5}{2}}} - \frac{1}{R^b S_b^{\frac{5}{2}}}\right) - \ldots\ldots\right] \quad (3.11)$$

It can be seen that the right hand side of the equation 3.11 consists of both positive and negative terms and therefore ΔZ may not be positive for every case.

Based on the above finding the component Z that supporting front loading concept can be expressed as follows.

$$Z = \alpha \frac{\sum_{j=1}^{m}\left(\frac{S_j}{C}\right) R^{m-j}}{mR^{m+1}} \quad (3.12)$$

3.2.2.4 DETERMINATION OF FRONT-LOADING CONSTANT (*R*)

The constant *R* plays a vital role in the new fitness function and it must be determined cautiously. It was shown that for front loading, *R* must be greater than unity (equation 3.9). Higher values of *R* will increase station weights exponentially and decrease the weight difference among latter workstations, and involves very large numbers. On the other hand, lower values reduce the weight difference between front and back workstations. Figure 3.8(a) and figure 3.8(b) show weights for a twenty workstation solution for $R = 5.0$ and $R = 1.25$ respectively.

In order to transfer elements from latter workstations to early workstations, the change of fitness (Δf_f) must be significant. In a problem having *m* workstations, there will be ($m^2 - m$) number of possible ways to shift elements among workstations, and all of these possibilities are shown in figure 3.9.

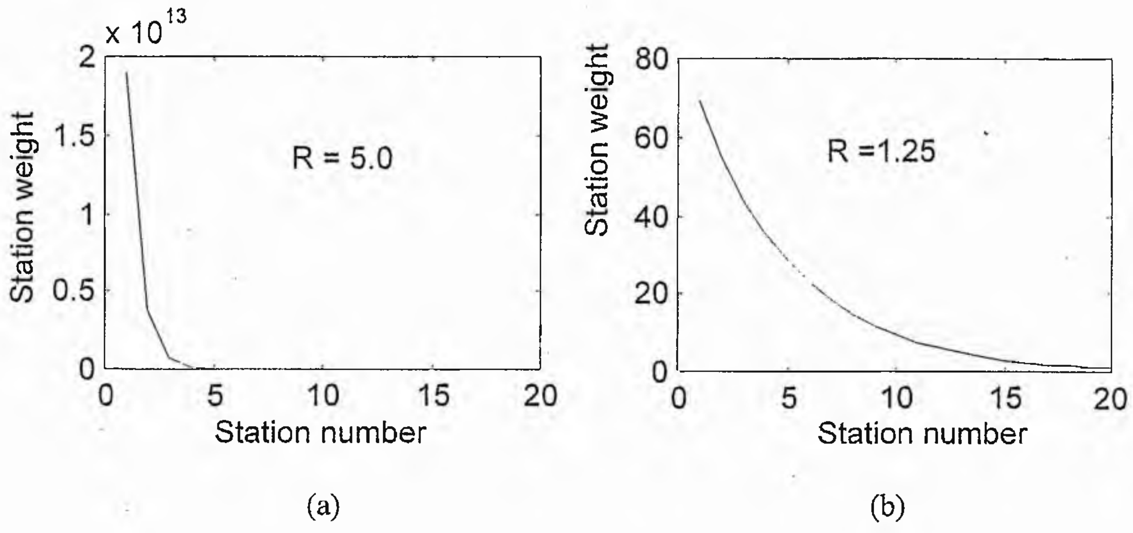


Figure 3.8. Station weights variation with station number (a) R = 0.5 (b) R = 1.25

In fact, moving a unit element from workstation *m* to *m-1* results the minimum change of fitness. This value can be calculated from the equation 3.9.

$$\Delta f_{f(m-1\to m)} = \frac{ER^m}{C}\left[\frac{1}{R^{m-1}} - \frac{1}{R^m}\right] = \frac{E}{C}(R-1)$$

Substituting $E = \dfrac{\alpha}{mR^{m+1}}$ and $\alpha = nCR^{m^*+1}$ in the above equation

$$\Delta f_{f(m-1\to m)} = \frac{n}{mR^y}(R-1)$$

Where $y = (m^*\text{-}m)$

This value (i.e., $\Delta f_{f(m\to\, m-1)}$) must be significant (say greater than some value θ) to force the algorithm to perform better forward loading. Therefore,

$$\Delta f_{\min} = \Delta f_{f(m-1\to m)} = \Delta f(R,\theta) = \frac{n}{mR^y}(R-1) \geq \theta$$

$$\Delta f_{\min}(R,\theta) = \frac{n}{mR^{y}}(R-1) - \theta \geq 0 \tag{3.13}$$

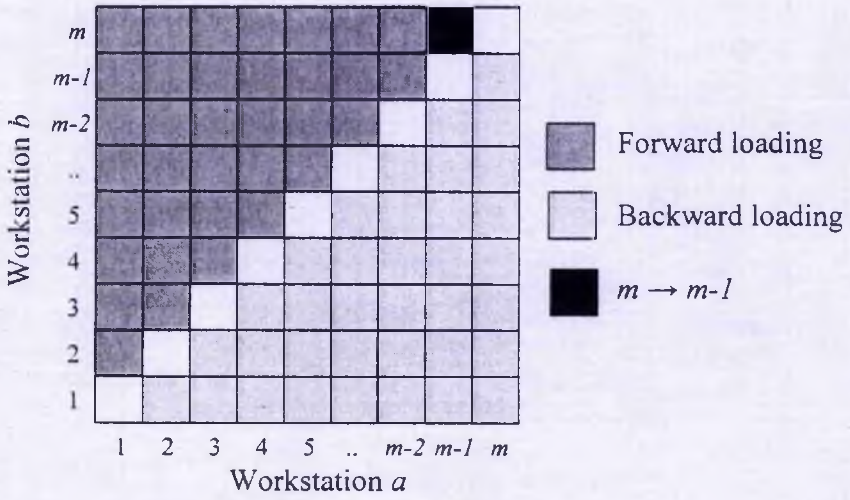


Figure 3.9. Element transferring possibilities

The above inequality (equation 3.13) gives R values for minimum Δf_f to greater than a particular θ. For example let's consider a problem where n = 45, m = 10, $y = 2$ and θ=1 (KW_{45}^{69}, Chapter 4). Substituting these values in equation 3.13 gives,

$$\Delta f_{\min}(R) = \frac{4.5}{R^2}(R-1) - 1 \geq 0 \tag{3.14}$$

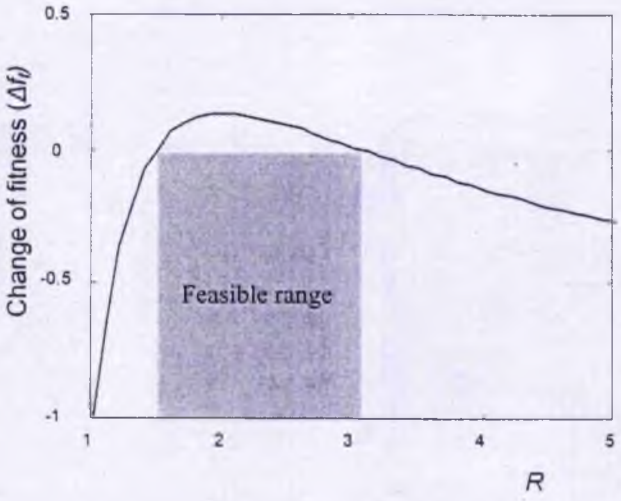


Figure 3.10. Minimum fitness change against R

Figure 3.10 shows the minimum change of fitness (that is $\Delta f_{\min}$) against R. It can be seen from the graph that between 1.5 and 3.0 $\Delta f_{\min}$ is positive and satisfies equation 3.14, implying the minimum change of fitness is greater than θ (=1). This determine the best range of R and therefore,

$$1.5 < R < 3.0$$

The change of fitness due to shifting a unit element between any two workstations can be calculated using equation 3.9 and figure 3.11 displays the change for all possible cases for the above problem having 10 workstations with R =2.

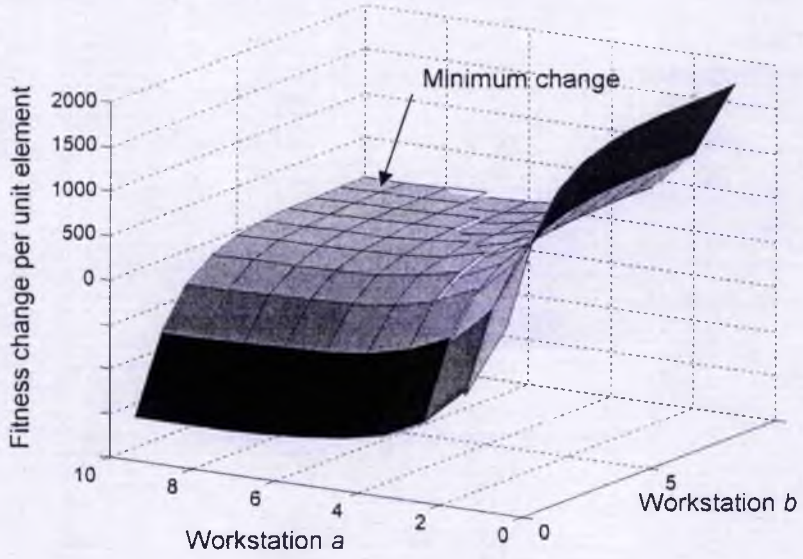


Figure 3.11. Fitness change per unit element transferring between workstations *a* and *b*.

Therefore, the complete overall global fitness function (*FF*) can be express as follows and it consists of f_f and f_i.

$$FF = \begin{cases} n^3 l \beta \\ n^3 l \beta + n^2 \beta \dfrac{\sum_{j=1}^{K} \left(\dfrac{S_j}{C} \right)}{m - m^* + 1} + \alpha \dfrac{\sum_{j=1}^{m} \left(\dfrac{S_j}{C} \right)^{..} R^{m-j}}{m R^{m+1}} & l = P \end{cases} \tag{3.15}$$

Where, l is the number of feasible links in the chromosome, and P is the total number of precedence constraints (links) in the problem.

3.3 INITIAL POPULATION

At the start of evolution, a Genetic Algorithm requires an initial population. A good initial population will certainly increase the performance of the algorithm and speed up its convergence. This type of population is known as a *well-seeded* population and the proposed model can generate an initial population in three ways.

The first method consists of generating *n* permutations. This population is known as bin-packing population, and there is no guarantee that the generated chromosomes will be feasible.

The second method generates feasible chromosomes by the random task assignment technique. As the name implies, this technique assigns elements to workstations randomly from a feasible list and the list is constructed using the precedence matrix described in chapter 2. The procedure can be repeated to generate different solutions and is ideal for the initialisation process. The steps involving in the algorithm are as follows:

Step1. Construct list A, showing all work elements in one column and the total number of elements that immediately precede each element in an adjacent column.

Step2. Construct list B, showing all elements from list A that have no immediate predecessors.

Step3. Select at random one of the elements from list B. The computer is programmed to perform this random selection process. The only constraint is that the element selected must not cause the cycle time to be exceeded.

Step4. Eliminate the element selected in step 3 from the lists A and B and update both lists, if necessary. Updating may be needed because the selected element was probably an immediate predecessor for some other element(s). Hence, there may be changes in the number of immediate predecessors for certain elements in list A; and there may be some new elements having no immediate predecessors that should be added to list B.

Step5. Again select one of the elements from list B that is feasible for cycle time.

Step6. Repeat steps 4 and 5 until all the elements have been allocated to stations within the cycle time constraint.

The third method employs a priori knowledge about the given optimisation problem. Previously published deterministic algorithms include COMSOAL, the Hoffman precedence matrix procedure and Rank Positional Weight (RPW) technique. These methods plus the above two methods are employed to generate an initial population and is the default method for the model. Since the Genetic Algorithm starts the optimisation with a set of approximately known solutions, convergence is quick and consumes less CPU time.

3.3.1 POPULATION SIZE

The population size has direct effect on the convergence of the algorithm and should be selected carefully. General wisdom dictates that a larger population will work more slowly, but will eventually achieve better solution than a smaller population. However, experience indicates, that this rule of thumb is not always true, and that the most effective population size is dependent on the problem being solved, the representation used, and the operations manipulating the representation. Suresh et al (1996) showed that a population of 30-50 was effective for Genetic Algorithm assembly line balancing problems and, in this model it was set to 40 chromosomes.

3.4 GENETIC OPERATORS

3.4.1 SELECTION

Selection is the process of choosing chromosomes for the next generation from the current generation. A number of selection schemes have been reported in the literature and most have not addressed the two common problems: high selective pressure and loss of population diversity. Population diversity is representing a variety of chromosomes of the population that were selected during the selection phase, whereas, selective pressure is the probability of the best chromosome being selected compared to the average probability of selection of all chromosomes. If selective pressure is too high, the population diversity decreases and this could result in a premature convergence, on the other hand weak selective pressure makes the search ineffective.

The standard rank-based selection technique, in which the population is sorted according to the fitness values depending only on its rank and not on the actual fitness value, is used in the new model with modifications to overcome the above two main problems. Figure 3.12 illustrates the modified rank-based selection scheme. First, once and for all, a population of N_{POPF} chromosomes (*feeding pool*) consisting of 85% feasible chromosomes and 15% of bin packing chromosomes are generated at the beginning of the evolution process. The value of N_{POPF} depends upon the number of elements (n) in the problem and the following values are recommended for better performance. The feasible chromosomes are generated by the random task assignment technique and the bin packing chromosomes by random permutation.

Number of elements (n)	Feeding population size (N_{POPF})
n <20	25
$20 \leq n \leq 50$	50
$50 > n$	100

Table 3.3. Feeding population sizes

Secondly, all the chromosomes in the current population are ranked according to their fitness values and the first N_b (user defined) chromosomes are selected to create a *selection pool*. Then, choosing chromosomes from the selection pool and performing crossover and mutation generates the rest of the new population. If any chosen chromosomes are identical, the currently selected pair will be discarded and a new pair of chromosomes will be selected and this will continue for maximum user specified times (n_i). If the technique is still unable to find two different chromosomes, the selected chromosomes are copied to the new population undergoing high mutation.

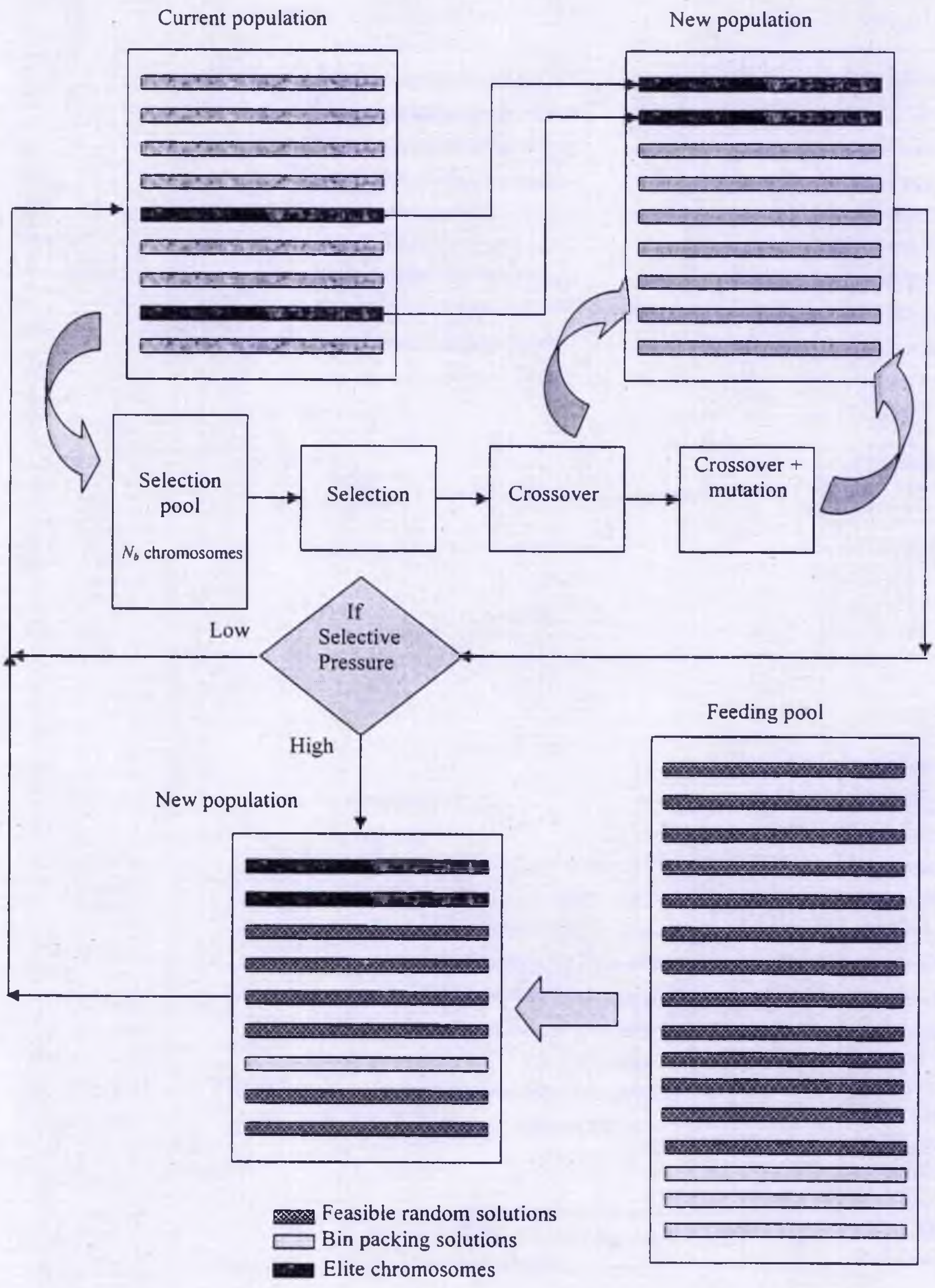


Figure 3.12. Modified rank based selection scheme

After constructing the new population, the number of identical chromosomes in the selection pool is counted and if they are greater than the specified value, all the chromosomes excluding the elite chromosomes in the new population are replaced with randomly selected chromosomes from the feeding pool. This reduces high selective pressure and adds population diversity.

3.4.2 CROSSOVER

The Crossover is a genetic operator that combines two chromosomes to create a new chromosome. The idea behind crossover is that new chromosome(s) may be better than both the mating chromosomes if it takes the best characteristics from each of the chromosome. Several crossover techniques have been developed since 1975, and they are reviewed in the literature survey (Rubinovits and Levitin, 1995). Falkennaure (Chambers, 1999, p.67) writing on application of Genetic Algorithms to real-world problems stated that:

> "The problem is that we never really know which part(s) of a good solution are the ones that make it a good solution, because we only have a measure of worth of the whole of it (the objective function). The parts must thus be tested"

None of the crossover techniques developed so far has addressed this issue. The proposed new techniques, Fixed Boundary Moving Crossover Point (FBMCP) and Variable Boundary Moving Crossover Point (VBMCP) mainly focus on the above issue and transferring best attributes of both parents to their offspring as the evolution progresses. The proposed Genetic Algorithm model consists of six standard crossover techniques described in Chapter 2 plus the two new crossover techniques with ability to adjust crossover probability (P_c) to suit clarity requirement.

3.4.2.1 FIXED BOUNDARY MOVING CROSSOVER POINT (FBMCP)

In FBMCP technique, a chromosome is divided equally in to a number of spans (s_n), depending upon a user defined crossover span-overlapping index (I_o) and crossover span size (c_s). The crossover point is selected randomly within the left (c_L) and right (c_R) locus of the specified span and the classic single point crossover technique is then applied to generate offspring. Both loci (boundaries) are incremented in a defined way at set generation intervals as the evolution progresses. The following illustrative example (figure 3.13) shows its application and the corresponding parameters are given below:

1. *Crossover overlapping index (I_o)*: This index determines the number of overlapping genes between two adjacent crossover spans. Overlap is very essential to explore the whole chromosome in the reproduction process. Without this some of the genes may not be selected at all, during the evolution especially at the margins.

2. *Crossover span (c_s)*: is the number of elements for which the total work content is less than or equal to the cycle time. This is determined by, arranging all the task elements in ascending order of their task times, and then calculating the cumulative task time. Figure 3.13 shows a typical cumulative task time curve where the abscissa corresponds to the cycle time given the crossover span. It is the size of the crossover zone and depends on the cycle time and the task time distribution.

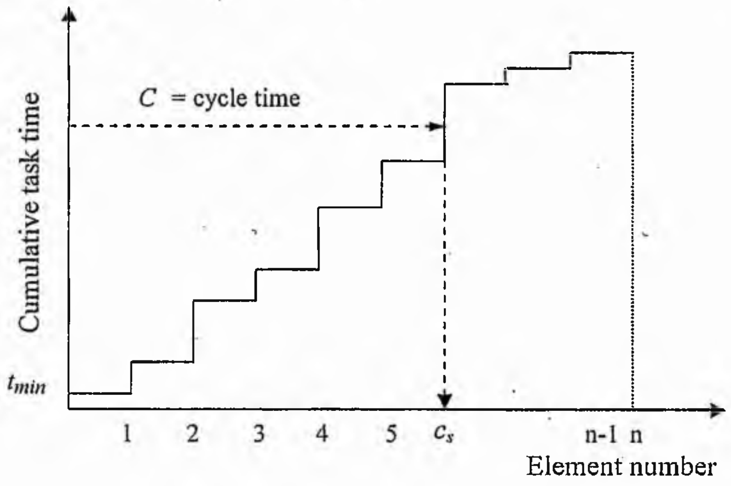


Figure 3.13. Cumulative task time curve

3. *Number of over-lapping genes* (g_n) is given by

$$g_n = \left[\frac{c_s}{I_o}\right]^+ \tag{3.16}$$

Where $[x]^+$ is the smallest integer larger or equal to x.

4. *Number of spans (s_n):* can be calculated by the following formula:

$$s_n = \left[\frac{n-g_n}{c_s-g_n}+1\right]^+ \tag{3.17}$$

Where n is the number of elements.

5. *Number of generations per crossover span (G_D):* is a test problem constant,

$$G_D = \left|\frac{G}{s_n}\right|^+ \tag{3.18}$$

Where G is the total number of generations.

6. *Span index* (s_i): which identifies the span number and determines both left and right loci of the crossover zone. Initially, s_i is equals to one and, if number of generations is equal to the product of G_F and s_i, then it is incremented by one unit.

Where $G_F = \left|\frac{G}{s_n}\right|^+$

G is the total number of generations and s_n is the number of crossover spans.

7. *Left crossover locus (c_L)*

$$(c_L) = \begin{cases} (s_i-1)(c_s-g_n) & c_L \le n\text{-}c_s, \\ n\text{-}c_s & c_L > n\text{-}c_s, \end{cases} \tag{3.19}$$

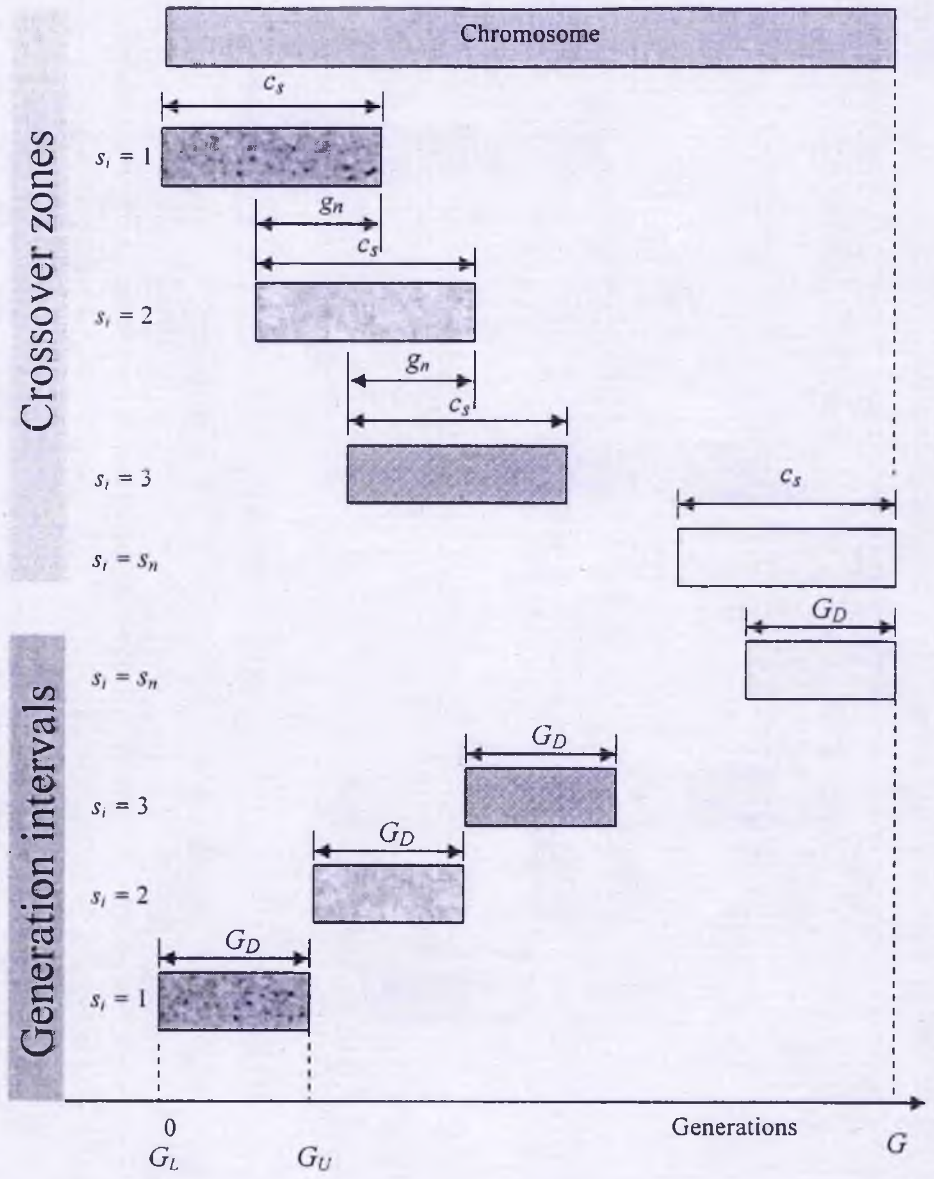


Figure 3.14. Moving crossover zones and generation class intervals (for FBMCP)

8. *Right crossover locus* (C_R)

$$(c_R) = \begin{cases} s_i c_s - (s_i - 1) g_n & c_R \leq n \\ n & c_R > n \end{cases} \tag{3.20}$$

Where s_i is Span index. ($1 \leq s_i \leq s_n$) , and n is the number of elements in the problem

9. *Generation class interval* $[G_L\ G_U]$: is the range of generations where a particular crossover span is active. This normally represented by two boundaries: the lower (G_L) and upper (G_U) generation boundaries respectively and are determined as follows:

$$G_L = G_D(s_i - 1) + 1 \tag{3.21}$$

$$G_U = G_D s_i \tag{3.22}$$

Figure 3.14 illustrated the moving crossover span concept and the variable generation intervals for different crossover zones.

3.4.2.2 VARIABLE BOUNDARY MOVING CROSSOVER POINT (VBMCP)

The VBMCP technique is a more advanced version of the fixed boundary moving crossover technique. Unlike FBMCP method, the left crossover boundary (c_L) is dynamically determined by taking into account the number of elements in the previous fully filled workstation(s). In the case where, a fully filled workstation cannot be achieved, after a user-defined number of generations (G_D) the lower left boundary is incremented, and depending upon the number of elements in the previous workstation (equation 3.24) and crossover-overlapping index (I_o). The crossover span (c_s) is fixed in this technique and calculated by considering the cycle time and the task element times. Once a crossover span is selected, a random point is selected within the boundaries of the span and the single point crossover operation is employed to generate offspring. The VBMCP technique can be applied with any fitness function, but good results can be achieved with the front-loading fitness function. This technique is superior to the FBMCP in the way that it changes the boundary point when workstations are fully filled, leading to faster convergence. The following illustrative example (figure 3.15) shows its application and the corresponding parameters are given below.

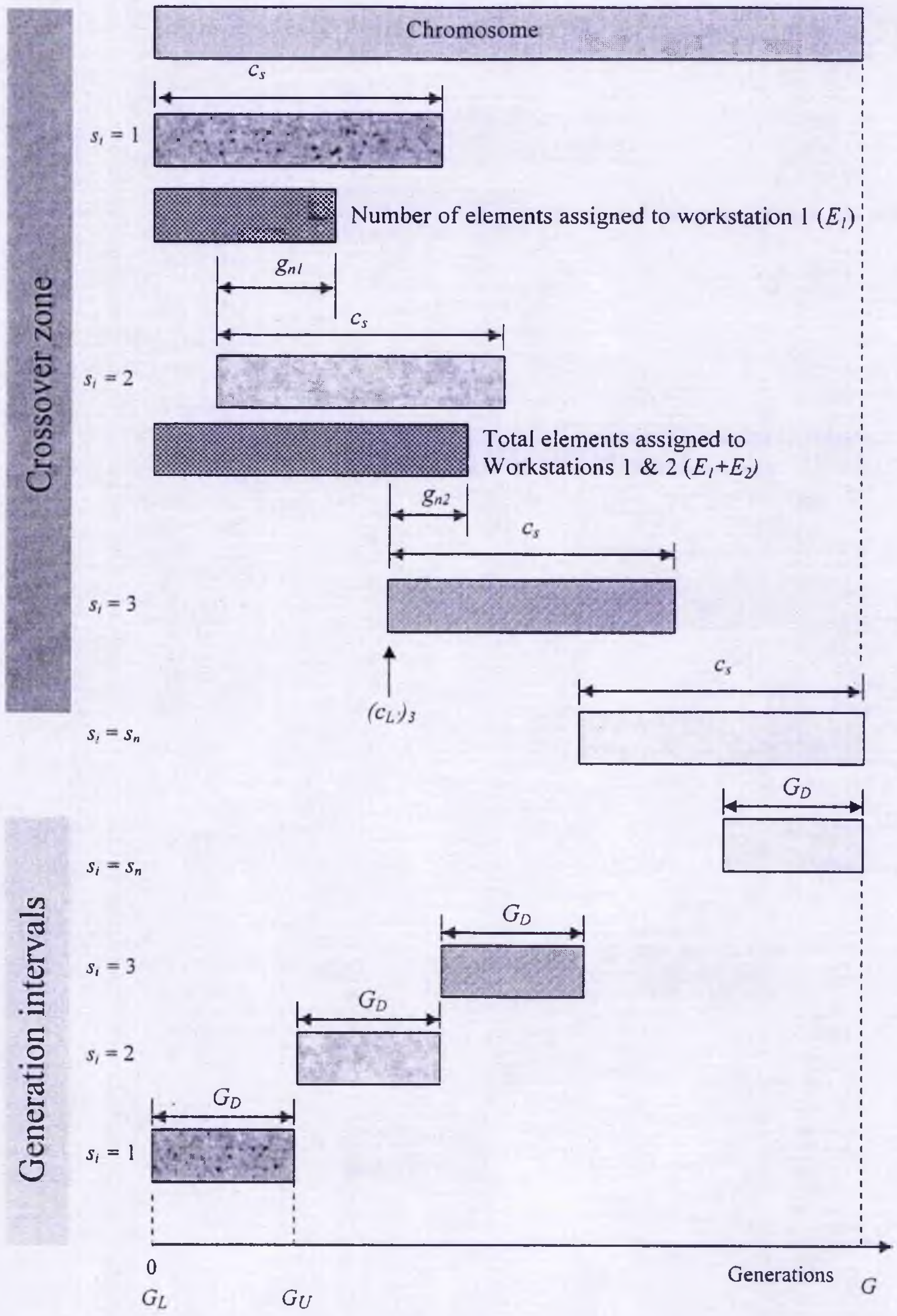


Figure 3.15. Crossover boundaries and generation intervals (for VBMCP)

1. *Number of over-lapping genes* ($g_{n_{si}}$)

$$g_{n_{si}} = \left\lceil \frac{E_{si-1}}{I_o} \right\rceil \qquad s_i>1 \tag{3.23}$$

Where I_o is the crossover overlapping index and E_{si} is the number of genes in the s_i workstation.

2. *Variable left crossover boundary* ($c_{L'}$): is the lower crossover boundary depending upon the number of elements in the previous workstations (> 1) and span index $s_{i'}$, and is defined as follows:

$$(c_{L'})_{si} = \begin{cases} 0 & s_i = 1 \ \& \ c_{L'} < n - c_s \\ \sum_{j=1}^{s_i-1} E_j - g_{n_{si}} & s_i > 1 \ \& \ c_{L'} < n - c_s \\ n - c_s & s_i > 1 \ \& \ c_{L'} \geq n - c_s \end{cases} \tag{3.24}$$

Where E_j is the total number of elements in the j^{th} workstation and n is the total number of elements in the problem.

3. *Variable right crossover boundary* ($c_{R'}$)

$$(c_{R'})_{si} = \begin{cases} (c_{L'})_{si} + c_s & \text{if } (c_{R'})_{si} < n \\ n & \text{if } (c_{R'})_{si} \geq n \end{cases} \tag{3.25}$$

3.4.3 MUTATION

Mutation is a genetic operator that alters one or more gene values in a chromosome from its initial state. This can result in an entirely new gene added to the gene pool. With these new gene combinations, the Genetic Algorithm may be able to arrive at a better solution than was previously possible. Only a few mutation techniques have been published for the assembly line balancing problem and they are reviewed in Chapter 2. The proposed model consists of four new mutation techniques plus previously published standard techniques and the mutation probability (p_m) is kept at a low value such as 0.01.

All four techniques are based on the moving mutation zone approach where mutation points are selected randomly from the corresponding zone. This approach lets the algorithm scan more feasible chromosomes and converge faster. The left and right boundaries of the mutation zone (m_L, m_R) are varied with generation class interval [G_L, G_U], as in the FBMCP and VBMCP techniques, keeping the mutation span size (m_s) constant. This is a problem dependent constant and equals to b times c_s (Where b is a constant and can be fixed by the user).

3.4.3.1 FIXED BOUNDARY ADJACENT MUTATION (FBAM):

The concept behind this is similar to the FBMCP technique. As the name applies, N_m number of mutation points is selected randomly within the corresponding mutation zone and they are swapped with the adjacent genes to their right. If the selected point lies on the right hand boundary of the zone, then it is swapped with the gene to its left. Figures 3.16 (a) and (b) illustrate the mutation operation before and after respectively

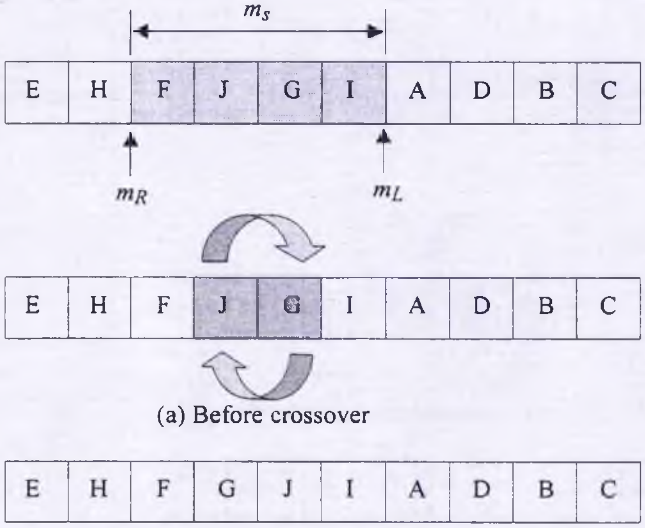


Figure 3.16. Adjacent mutation operation

3.4.3.2 FIXED BOUNDARY RANDOM MUTATION (FBRM):

This technique is similar to the above; one or more pairs of genes are selected randomly within the mutation span and they are swapped to produce new offspring.

3.4.3.3 VARIABLE BOUNDARY ADJACENT MUTATION (VBAM):

In this procedure, the lower or left boundary (m_L) of the mutation span is changed according to the filling of workstations. This allows the algorithm to explore the whole chromosome more rapidly than FBRM technique. Mutation points are selected randomly and swapped as for FBAM

Mutation technique	Mutation span boundaries	
	Left boundary (m_L)	Right boundary (m_R)
FBAM FBRM	$(s_i - 1)\left(m_s - \left\lvert \frac{m_s}{I_o} \right\rvert^{+}\right)$ (3.26)	$s_i m_s - (s_i - 1)\left\lvert \frac{m_s}{I_o} \right\rvert^{+}$ (3.27)
VBAM VBRM	$\begin{cases} 0 & s_i = 1 \\ \sum_{j=1}^{s_i-1} E_j - \left\lvert \frac{E_{si-1}}{I_0} \right\rvert^{+} & s_i > 1 \end{cases}$ (3.28)	$\sum_{j=1}^{s_i-1} E_j - \left\lvert \frac{E_{si-1}}{I_0} \right\rvert^{+} + m_s$ (3.29)

For all cases if $m_L > n$ then $m_L = n$ and $m_R = n - c_s$

Table 3.4. Lower and upper mutation span boundaries

Mutation technique	Generation class boundaries	
	Lower boundary (G_L)	Upper boundary (G_U)
FSAM FSRM	$\left\lvert \frac{G}{s_n} \right\rvert^{+} (s_i - 1) + 1$ (3.30)	$\left\lvert \frac{G}{s_n} \right\rvert^{+} s_i$ (3.31)
MSAM MSRM	$G_F (s_i - 1) + 1$ (3.32)	$G_F s_i$ (3.33)

Table 3.5. Lower and upper generation class boundaries

3.4.3.4 VARIABLE BOUNDARY RANDOM MUTATION (VBRM).

This technique is similar to the VBAM in determining the mutation boundaries and similar to the FBRM in mutation operation. Left and right mutation boundaries (m_L, m_R) and lower and upper generation class boundaries (G_L,G_U) of mutation zones for the above four techniques are shown in tables 3.4 and 3.5.

3.5 REPAIR TECHNIQUE

Applying standard genetic operators in the assembly line balancing problem can rarely ensure offspring feasibility. Duplication of genes in offspring is the main cause of infeasibility. Figure 3.17 illustrates this problem using two feasible chromosomes generated from the problem in figure 3.5. After a crossover operation, genes B and C are duplicated in child 1 and 2 respectively whilst genes C and B are missing in child1 and 2 respectively, violating the assembly line design concept.

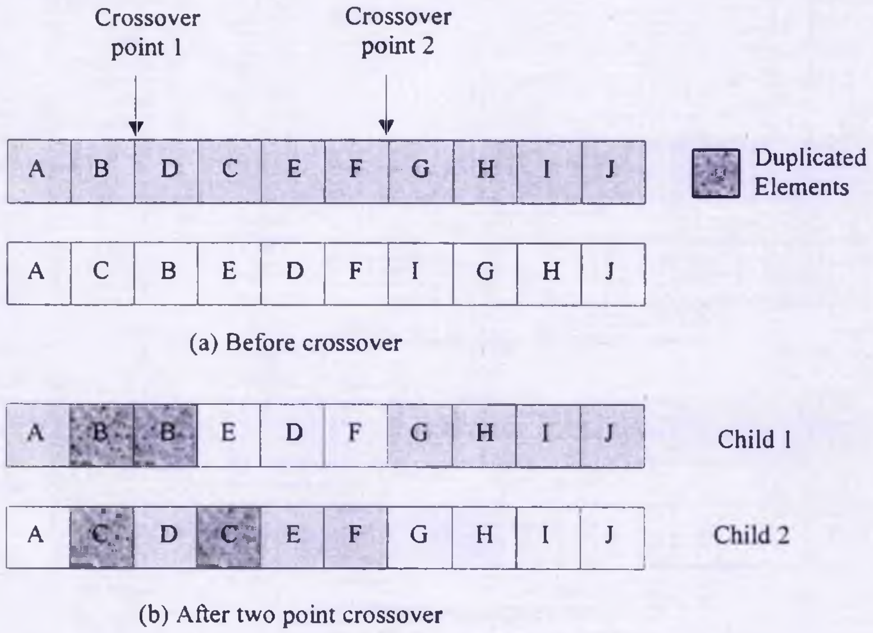


Figure 3.17. Duplicating elements after crossover operation

3.5.1 RANDOM BASED REPAIR TECHNIQUE

This technique removes all the duplicated elements from both offspring and creates a gene pool. Then, missing genes are selected from the pool and reassigned randomly by the following procedure:

1. Remove duplicated genes from both chromosomes and prepare a list. The number of genes in the list is equal to N_d.
2. Using a random number generator, generate a random integer between one and N_d and select the corresponding gene from the list and reassign to the first empty position towards the left.
3. Remove the assigned gene from the list and update the value N_d.
4. Go to step 2, and repeat the procedure until all the duplicated genes are reassigned.

3.5.2 ORDER BASED REPAIR TECHNIQUE

In the order based repair technique, the genes are reassigned to empty spaces based on their order in the precedence diagram. For example {1, 2, 3, 4...} or {A, B, C, D...}. Assigning early elements in the precedence diagram first tends to increase the feasibility of the chromosome and the complete procedure is described as follows:

1. Remove duplicated genes from both chromosomes and prepare a list. The genes in the list are arranged in the order that they appear in the precedence diagram.
2. Select the first gene from the list and reassigned to the first empty position in the chromosome towards the left.
3. Remove the assigned gene from the list and update the list.
4. Go to step 2, and repeat the procedure until all the duplicating genes are assigned.

3.5.3 RANK POSITIONAL BASED REPAIR TECHNIQUE

This technique is similar to the above technique except for step 1. The element list is arranged according to their positional weights, instead of their order of appearance. The positional weight (pw_k) of the k^{th} element (gene) is calculated using the formula below:

$$pw_k = t_k + \sum_{h \in F_k^*} t_h \qquad (3.34)$$

Where F_k^* consists a set of immediate transitive followers of task k.

Once the positional weights are calculated the list is rearranged according to the ranks of the positional weights. The highest rank first and so on. The rest of the procedure is same as the order based repair technique (steps 2-4).

3.6 ELITISM

The convergence of generic algorithms is one of the most challenging theoretical issues in the evolutionary computation area. Several researchers explored this problem from different perspectives. Recently Rudolph (1994) proved that the classical Genetic Algorithm never converges to a global optimum, but a modified version, which maintains the best chromosomes in the population, does. This is because when creating new populations by crossover and mutation there is a good chance that the best chromosome may be lost.

Elitism is the name of a method which first copies the best chromosome (or a few best chromosomes) to the new population. The rest of the population is generated in a classical way. Elitism very rapidly increases the performance of the Genetic Algorithm, because it prevents losing the best-found solution. In the genetic model, the user can define the number of elite chromosomes (N_e) in the population. It is given as a percentage of the total population.

3.7 TERMINATION

Termination is the criterion by which the Genetic Algorithm decides whether to continue searching or stop the search. Two termination criteria are used in the proposed model. The default criterion is the theoretical minimum number of workstations and, if this condition is not satisfied, then the evolution will stop when the user-defined maximum number of generations has been run.

3.8 COMPLETE ALGORITHM

The classic Genetic Algorithm described by Holland (1975) is what has become known as the generational Genetic Algorithm. The proposed algorithm is based on this concept and it keeps two populations all the time, the current and the future (new). Each iteration (generation) proceeds by constructing the new population, and finally the one just constructed becomes the current one in the next generation.

The chromosomes in the new generation come from three sources: some are the offspring of recombination (i.e. crossover); others are products of mutation of the chromosomes in the current population. The rest are simply copied unchanged from the current population (elitism). Mutation is normally applied after crossover selecting random chromosomes from the new population.

The overall fitness function (equation 3.15) is used for chromosome evaluation and then, they are sorted for the rank-based selection. The best 75% of the population is used as the selection pool and chromosomes are selected from the pool randomly for genetic operations.

After crossover, the new chromosomes are examined for any duplication of elements. These chromosomes are then rectified by a repair technique. In case of identical parents, the algorithm will search for new parents for a fixed number of times and afterwards they are copied to the new population undergoing heavy mutation. The number of mutated pairs is three times the normal mutation. When the termination criterion is met, the best chromosome is selected from the current population and it is decoded to obtain the solution.

Row vectors and matrices are used to represent chromosomes and populations respectively. The high level, user-friendly programming environment MATLAB is used for programming. Its built-in functions to handle matrix algebra provided the power for large-scale number crunching and permits writing source code (m. files) that can be directly executed in the MATLAB workspace. Easy to build, data visualization, menu driven GUI interface modules are used for interaction. The flow chart of the proposed GA line-balancing simulation model with the above features is shown in figure 3.18 and the corresponding steps are given below.

1. Start.
2. Read task time and precedence data.
3. Input Genetic Algorithm parameters.
4. Generate initial population.
5. Select fitness function.
6. Evaluate each chromosome in the population using the selected fitness function.
7. If the termination criterion is met, go to step 8, otherwise go to step 11.
8. Select the best chromosome(s) from the population.
9. Decoding.
10. Display solution and other graphical representations and stop.
11. Sort the chromosomes in the population according to their fitness values.
12. Copy best chromosome(s) to the next population.
13. Select two chromosomes (parents) for genetic operations.
14. If the selected chromosomes are identical, discard them and go to step 13, otherwise go to step15.
15. Apply crossover operation to the selected chromosomes and generate new offspring.
16. If the offspring contain duplicate element(s) go to step 17, otherwise go to step 18.
17. Remove duplicate element(s) using the repair technique.
18. Apply mutation
19. Copy chromosomes to the new population.
20. If the number of chromosomes in the new population is less than N_{pop} go to step 13 and repeat steps 13 to 19 until number of chromosomes equals $N_{pop.}$ Go step 6 and replace the current population with the one generated in step 19.

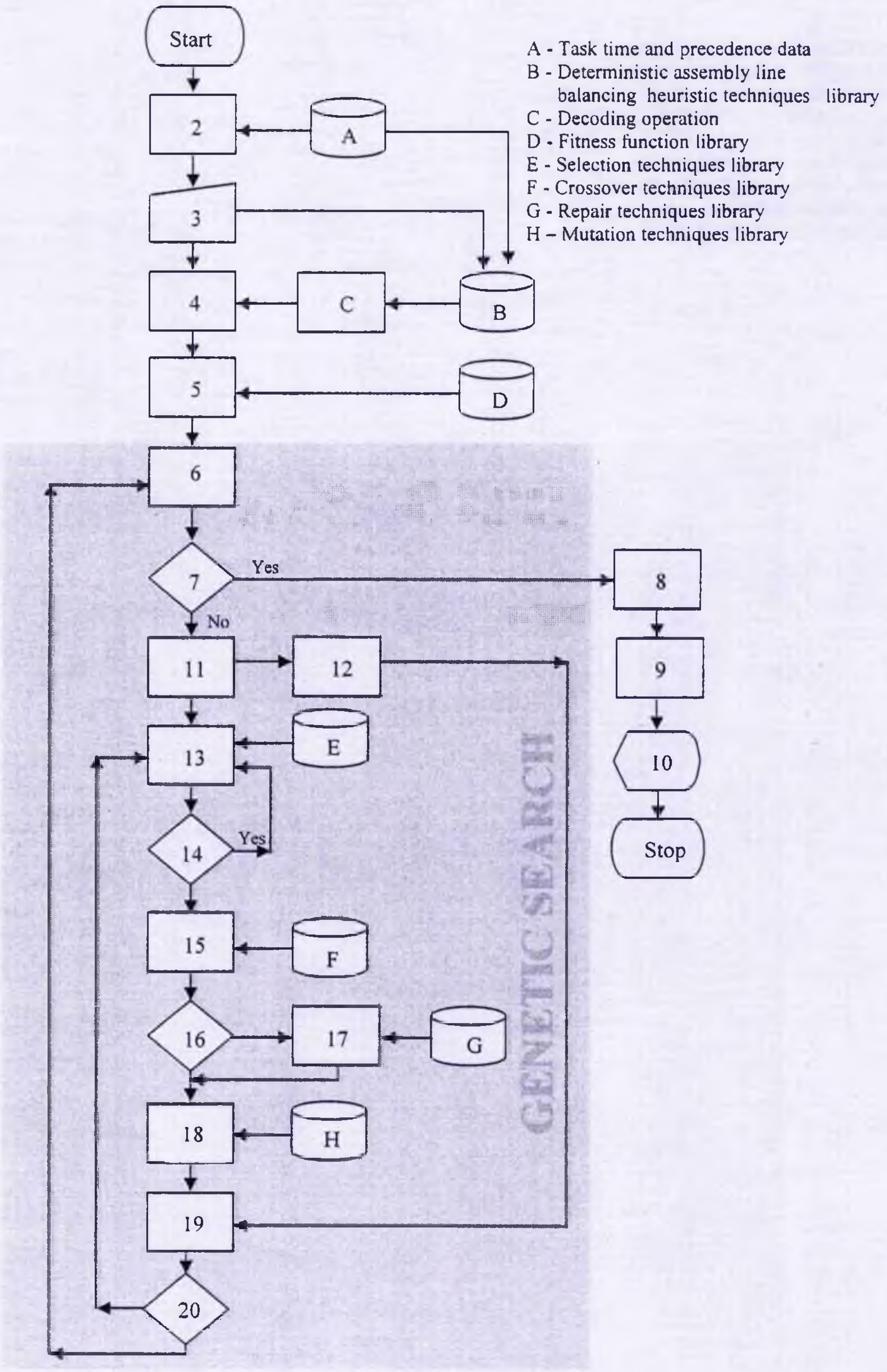

Start
2
A
3
4
C
B
5
D
6
7
Yes
No
8
9
10
Stop
11
12
13
E
14
Yes
15
F
16
17
G
18
H
19
20
GENETIC SEARCH
A - Task time and precedence data
B - Deterministic assembly line balancing heuristic techniques library
C - Decoding operation
D - Fitness function library
E - Selection techniques library
F - Crossover techniques library
G - Repair techniques library
H – Mutation techniques library

Figure 3.18. Flow chart of the Genetic Algorithm line-balancing model

Some of the basic features associate with software design and development are described below:

1. *Loading and saving data*: the software starts with the reading of two text files, namely precedence data and task time data written in two separate directories:
   D:\data\precedence\problemID.txt and D:\data\tasktime\problemID.txt respectively. After reading these files, they are assigned to two matrices [P] and [T] respectively. The output of the algorithm, that is all the data and variables created during computation are saved in MATLAB binary format in to the file (D:\gamodel\problemID\results.mat).

2. *Input Data*: the following test problem data and Genetic Algorithm control parameters are entered manually.
   a. Cycle time
   b. Number of evolutions
   c. Population size
   d. Initial heuristic solution techniques
   e. Crossover and mutation techniques and their probabilities.
   f. Optimisation criterion
   g. Fitness function model
   h. Selection method

3. *Coding and initialisation*: chromosomes are assigned to row vectors, and these vectors collects together constitutes the initial population matrix. Two population matrices are maintained in the model. The current population [G] and next population [GN] respectively. During the evolution new offspring are copied into the matrix [GN] and after each generation, matrix [G] is replaced by [GN] and the entries of matrix [GN] are cleared to make room for new offspring.

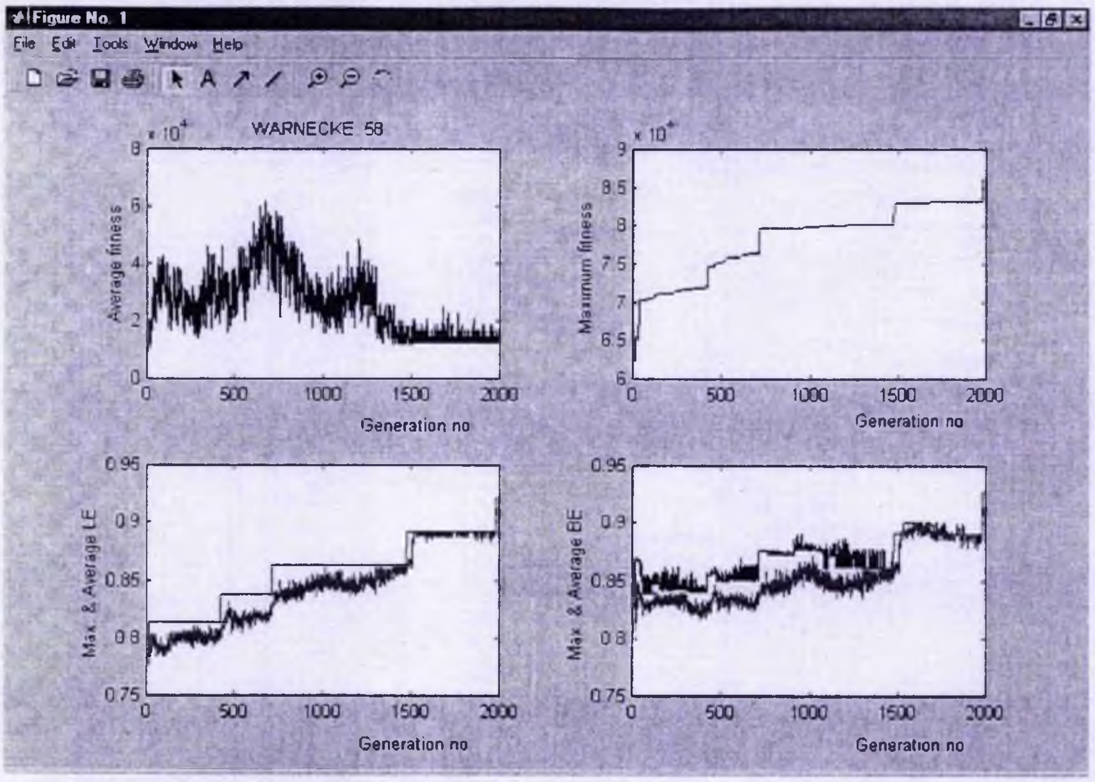
4. *Mathematical Procedures:* these procedures in the Genetic Algorithm model include coding, fitness evaluation, crossover, mutation, selection

and stopping procedures. They are normally accomplished using random number generators, sorting algorithms and several other built-in functions such as exponential and algorithmic functions. The normal distribution is used to generate random numbers

5. *Numerical values:* Comparing two feasible chromosomes might require the comparison of two fitness values which are very close to each other. Real numbers with 16 digits plus exponent are used to differentiate chromosomes.

6. *Data output:* the powerful Graphic User Interface (GUI) properties of MATLAB provide data visualization. The line balancing solution, element assignments, problem complexities and line balancing measures are tabulated and displayed in the MATLAB *command* window. All the relevant graphs, bar charts and statistical plots are displayed in *figure windows*.

A graphical and text output of the model for the 58–element Warnecke (WA_{58}^{56}) problem (cycle time =56) are shown in figure 3.19.



LE = Line Efficiency   BE = Balance Efficiency

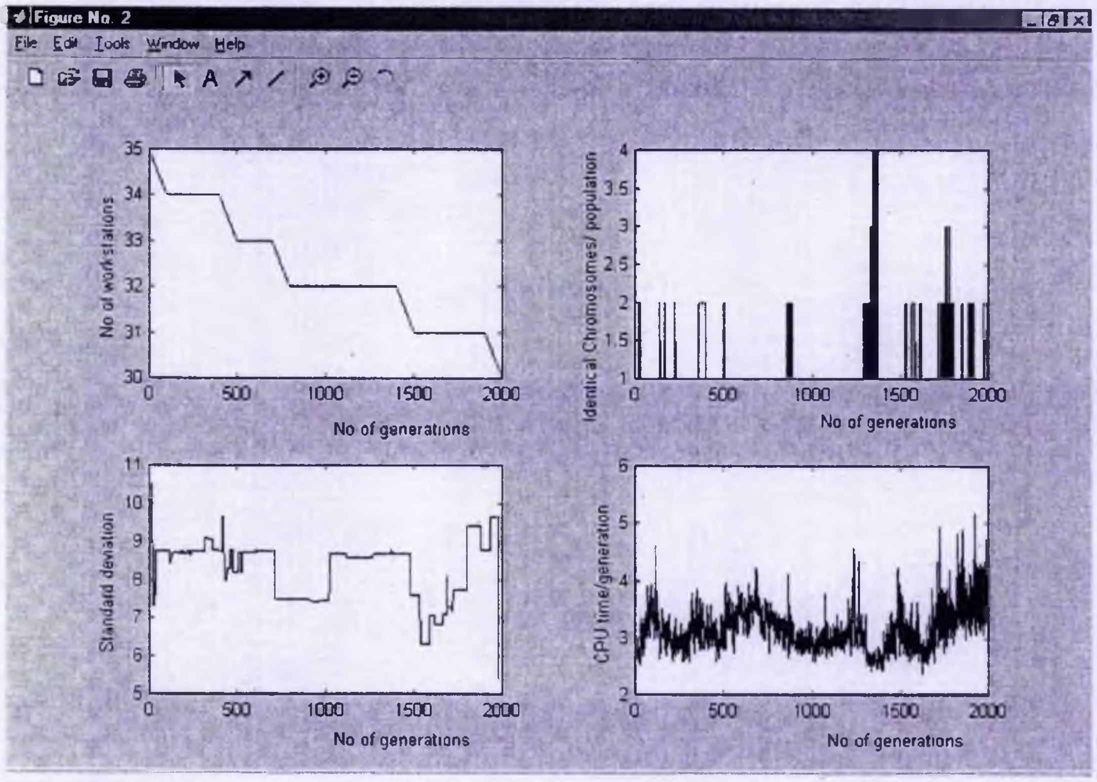
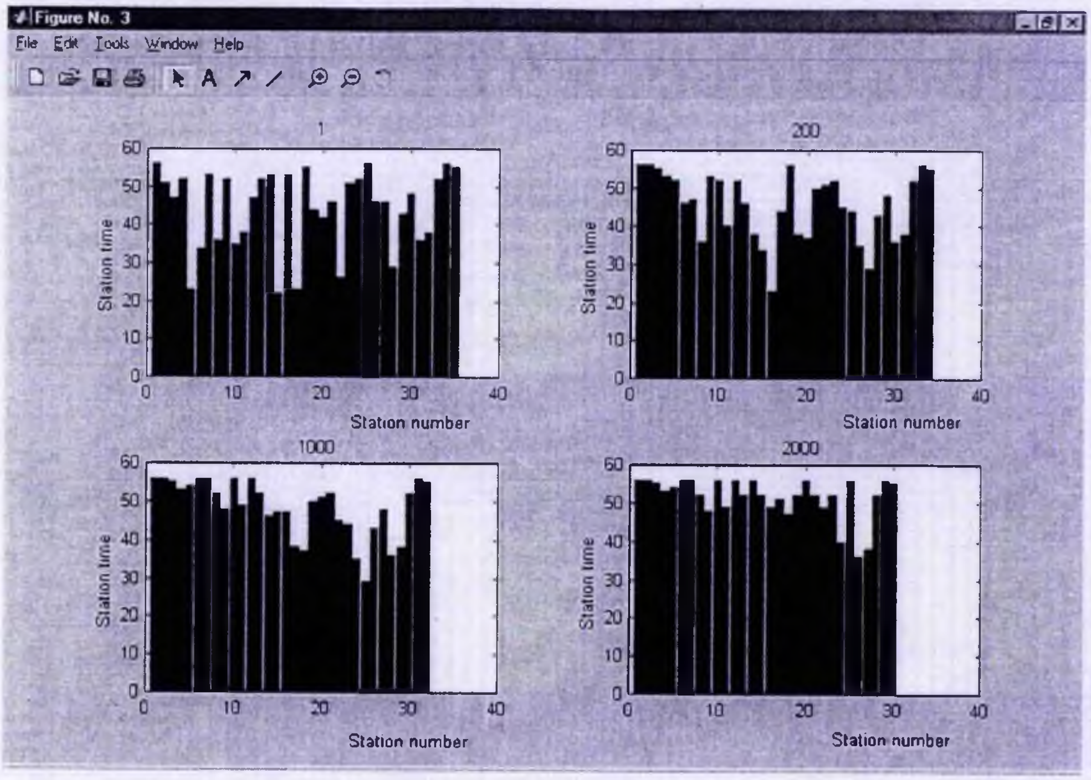


Figure 3.19 Genetic Algorithm line balancing analysis (graphical)



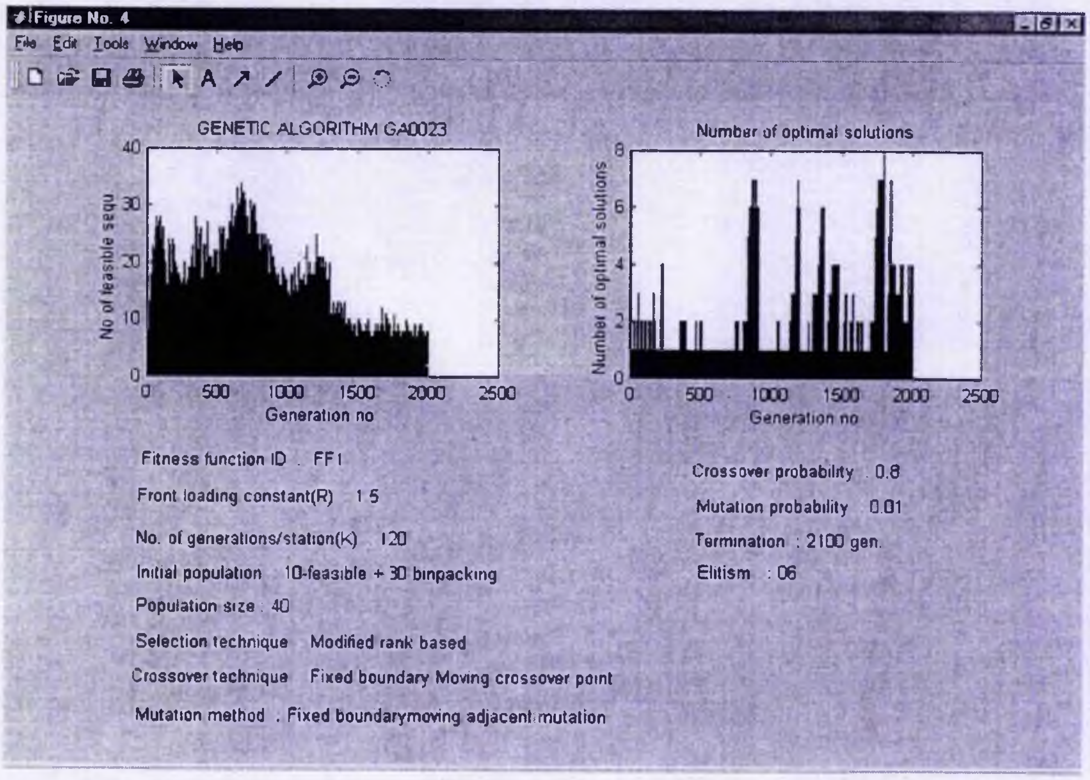


Figure No. 4
File Edit Tools Window Help
GENETIC ALGORITHM GA0023
No of feasible sequ
Generation no
Number of optimal solutions
Number of optimal solutions
Generation no

Fitness function ID : FF1

Front loading constant(R) : 1.5

No. of generations/station(K) : 120

Initial population : 10-feasible + 30 binpacking

Population size : 40

Selection technique : Modified rank based

Crossover technique : Fixed boundary Moving crossover point

Mutation method : Fixed boundarymoving adjacent mutation

Crossover probability : 0.8

Mutation probability : 0.01

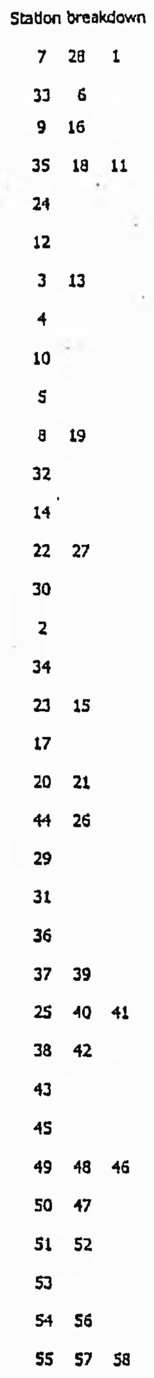
Termination : 2100 gen.

Elitism : 06

Figure 3.19 Genetic Algorithm line balancing analysis (graphical)

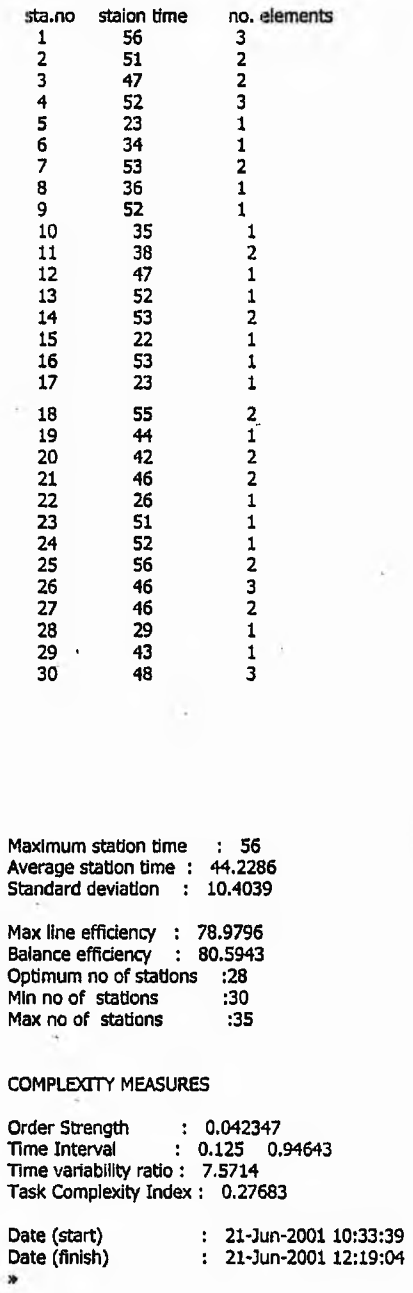
```
*****************************************************
  GENETIC ALGORITHM LINE BALANCING SIMULATOR
*****************************************************

PROBLEM SPECIFICATIONS

Problem ID       :  WARNECKE  58
Cycle time   :56

FITNESS FUNCTION SPECIFICATIONS

Fitness function ID  :  FF1
Front loading constant(R)   : 1.5

INITIAL POPULATION

Random task assignment method
Number of random solutions  : 01
Population size  : 40

GENETIC OPERATORS

SELECTION

Selection technique  : Modified rank based method
Selective pressure   : 30%
Selection population size : 100

CROSSOVER

Crossover technique  : Fixed boundary moving crossover point
Crossove probability  :  80%

MUTATION

Mutation technique  : Fixed Boundary moving adjacent mutation
Mutation probaility  : 1%
Repair technique(1/2/3)   : 1

TERMINATION
Number of generations : 2000


RESULTS

NO EVOLUTIONS  LE     BE      CPU    NO STATIONS

      0.00     0.79    0.82     0.00    35.00
    100.00    81.30   83.86     3.24    34.00
    200.00    81.30   84.12     3.08    34.00
    300.00    81.30   83.99     2.91    34.00
    400.00    81.30   84.05     3.02    34.00
    500.00    83.77   84.91     3.24    33.00
    600.00    83.77   85.04     3.35    33.00
    700.00    83.77   84.91     3.73    33.00
    800.00    86.38   87.31     3.29    32.00
    900.00    86.38   87.31     3.07    32.00
   1000.00    86.38   87.56     3.02    32.00
   1100.00    86.38   86.11     2.86    32.00
   1200.00    86.38   86.24     3.02    32.00
   1300.00    86.38   86.11     2.97    32.00
   1400.00    86.38   86.11     2.58    32.00
   1500.00    89.17   89.12     2.97    31.00
   1600.00    89.17   90.03     3.03    31.00
   1700.00    89.17   88.98     3.40    31.00
   1800.00    89.17   88.98     3.19    31.00
   1900.00    89.17   88.97     4.07    31.00
   2000.00    92.14   92.58     3.79    30.00
```

Figure 3.20. GA line balancing analysis (text output)



```
Station breakdown

 7  28  1

33   6
 9  16

35  18  11

24

12

 3  13

 4

10

 5

 8  19

32

14

22  27

30

 2

34

23  15

17

20  21

44  26

29

31

36

37  39

25  40  41

38  42

43

45

49  48  46

50  47

51  52

53

54  56

55  57  58
```

Figure 3.20. GA line balancing analysis (text output)



```
sta.no   staion time   no. elements
 1        56            3
 2        51            2
 3        47            2
 4        52            3
 5        23            1
 6        34            1
 7        53            2
 8        36            1
 9        52            1
 10        35            1
 11        38            2
 12        47            1
 13        52            1
 14        53            2
 15        22            1
 16        53            1
 17        23            1
 18        55            2
 19        44            1
 20        42            2
 21        46            2
 22        26            1
 23        51            1
 24        52            1
 25        56            2
 26        46            3
 27        46            2
 28        29            1
 29        43            1
 30        48            3


Maximum station time    :  56
Average station time  :  44.2286
Standard deviation    :  10.4039

Max line efficiency   :  78.9796
Balance efficiency    :  80.5943
Optimum no of stations    :28
Min no of  stations        :30
Max no of  stations        :35

COMPLEXITY MEASURES

Order Strength        :  0.042347
Time Interval         :  0.125    0.94643
Time variability ratio :  7.5714
Task Complexity Index :   0.27683

Date (start)            :  21-Jun-2001 10:33:39
Date (finish)           :  21-Jun-2001 12:19:04
»
```

Figure 3.20. GA line balancing analysis (text output)

CHAPTER

TEST PROGRAMME FOR GA LINE BALANCING MODEL

The proposed genetic algorithm model consists of four new approaches related to fitness function, dynamic crossover and mutation techniques, repair method and a modified rank based selection scheme. Each of these components has its own control parameters and, they have significant impacts on the performance of the model. These new components are designed to outperform the existing line balancing genetic algorithm models, and to support this hypothesis a series of test programmes have been prepared using six previously published benchmark problems. This chapter explains the logic behind the test and the experiments proposed for evaluation of the above new components.

4.1 TEST CASES

The selected problems represent a spread of problem sizes ranging from 58 to 297 elements including the largest published test problem in the literature. The cycle time of each problem was selected to allow a theoretical minimum number of workstations (m^*) equals to twenty-five. This is because, firstly, equal numbers of workstations show the same front-loading characteristics on each problem and, secondly, these cycle times represent hard problems and exhibit a significant difference in the number of workstations between the initial and the final solution. This can be used to assess the algorithm's capacity for improving the quality of solutions. Table 4.1 describes the characteristics of the selected precedence networks by specifying the original reference, the number of tasks, the minimal and maximal task time as well as the sum of task times, in each case. Two single-valued complexity measures for precedence networks including Order Strength (equation 2.2) and Time Variability ratio (equation 2.3) are shown. The task time distributions of the problems are shown in Appendix D. These characteristics show that the selected problems cover a reasonably good spectrum without any bias.

In this research, the following coding system was used for problem identification (figure 4.1). Two upper case letters and a subscript indicate the problem reference and the number of elements in the problem respectively. A superscript denotes the cycle time of the particular test case.

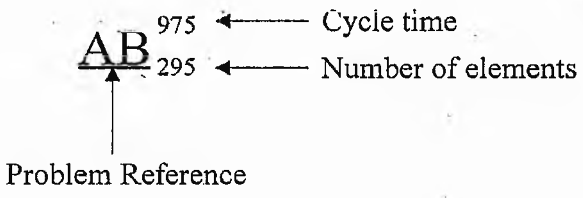


Figure 4.1. Test case coding system

In addition to the above problems, 48 challenging test cases of Scholl's data set (Scholl, 1999) containing 12 precedence networks, including the well known 45-element Kilbridge and Wester problem with four cycle time assignments ($KW_{45}^{69}, KW_{45}^{92}, KW_{45}^{138}$ and KW_{45}^{184}), which have been used in the literature to evaluate and compare algorithms, were balanced using the model to assess its overall performance. The characteristics of the additional test problems and their task time distributions are also shown in Appendix D.

The precedence network details and corresponding task times were down loaded from the following World Wide Web site.

(http://www.bwl.tudarmatadt.de/bwl3/forsch/projekte/alb/albdata.htm).

ID	Reference	*n*	*C*	Min. task time	Max. task time	Order Strength	Time variability ratio
SH_{297}^{2787}	Scholl (1993)	297	2787	5	1386	58.2	277.2
BA_{148}^{170}	Bartholdi (1993) modified	148	170	1	83	25.8	83.0
AR_{111}^{6269}	Arcus (1963)	111	6269	10	5689	40.4	568.9
MU_{94}^{176}	Mukherjee and Basu (1964)	94	176	8	171	44.8	21.4
LU_{89}^{20}	Lutz (1974) modified	89	20	1	10	77.6	10.0
WA_{58}^{65}	Warneke (1,971) modified	58	65	7	53	59.1	7.6

Table 4.1. Test problem specifications

4.2 HARDWARE AND SOFTWARE ENVIRONMENT AND EVALUATION CRITERIA

In order to perform fair comparisons of different genetic algorithm models, the influence of the programming language, the way of coding or the data structures have to be eliminated. Therefore, all algorithms are coded using the MATLAB (version 5.3) high-level language (MATLAB, 1998) and the same data structure was used for all the compared algorithms. All computations described in this chapter were performed on a single IBM-compatible personal computer in order to achieve comparable results. The specifications of the used computer system are given below.

1. Central processing unit: Intel Pentium III
2. Processing speed: 550 MHz
3. Available memory: 64MB RAM

The proposed experiments were carried out with the selected six test cases varying only one parameter, to examine its true impact on the performance. In all the experiments the following genetic algorithm parameters were maintained at the values shown in table 4.2 unless otherwise specifically stated.

Control parameter	Values and techniques
Fitness function	*FF_1 (front loading)*
Population size	40
Initial population	10 random task assignment solutions plus 30 bin-packing solutions
Selection criterion	Modified rank-based selection scheme
Selection pool size	20 chromosomes
Crossover method	Variable boundary moving crossover technique
Crossover rate	0.80
Repair criterion	Rank positional based technique
Mutation technique	Variable boundary moving mutation technique
Mutation rate	0.01
Number of elite chromosomes	06
Termination criterion (Number generations)	Number of generations (3000)

able 4.2. GA control parameters

The following performance measures and data were recorded at each one hundredth interval up to three thousand generations for evaluation.

1. Maximum and average line efficiency
2. Maximum and average balance efficiency
3. Maximum and average population fitness
4. Number of feasible solutions in each generation
5. Optimal number of solutions in each generation
6. Processing time per generation
7. Population selective pressure

These data sets were plotted against number of generations plus average Line and Balance Efficiency, average increase above the theoretical minimum number of workstations were used for evaluation and comparison of performances. Finally, the quality of solutions obtained by the model was compared against the following heuristic techniques.

1. Rank positional weight technique (Helgeson and Birnie, 1961).
2. Hoffman precedence matrix procedure (Hoffmann, 1963).
3. COMSOAL (Arcus, 1966).
4. Modified Hoffman technique (Thilakawardana et al, 2002)

4.3 MODEL CONTROL PARAMETERS

4.3.1 FITNESS FUNCTION COMPARISON

The novel front-loading fitness function is the key component of the new model. It was designed using the front-loading concept (described in Chapter 3) to outperform existing fitness models and have proved theoretically that it yields the optimum solution or theoretical minimum number of workstations in a multi solution environment. In order to support this hypothesis, eight fitness function models including seven previously published plus one based purely on balance efficiency were compared against the proposed model.

The original fitness functions are listed in table 2.5. Depending on the fitness function, a genetic algorithm model becomes a maximization or minimization problem. Four of the selected fitness functions including the front-loading fitness function fall into the maximization category and the others into the minimization category. The minimization models were modified into maximization models as shown in table 4.3 to make the comparison fair and meaningful.

Each modified overall fitness function ($FF_{2...9}$) consists of parts for evaluating feasible and infeasible chromosomes. If the number of feasible precedence links in a chromosome is less than the total number of precedence links (P), the chromosome is *infeasible* and, the number of feasible precedence links (l) becomes the overall fitness function. On the other hand, if the total number of feasible links equals the total number of precedence constraints (i.e., $l = P$) the chromosome is a feasible chromosome and, therefore, the overall fitness function is given by $f_i + (f_f)_k$; where f_i is the fitness function evaluating infeasible chromosomes and $(f_f)_k$ is the secondary fitness function evaluating feasible chromosomes and generally different from model to mode. Therefore, the complete overall fitness function is given in equation 4.1.

$$FF_k = \begin{cases} f_i & l < P \\ f_i + (f_f)_k & l = P \end{cases} \tag{4.1}$$

Where $f_i = l$

4.3.1.1 THE INFLUENCE OF FITNESS FUNCTION PARAMETERS

It can be seen from the equations 3.8 and 3.9, that the change of fitness (ΔZ) is exclusively dependent on the front-loading constant R. A high value of R decreases the change of fitness, particularly transferring elements among latter workstations in problems with a large number of workstations. Low front-loading constants do not create enough fitness change however to force the algorithm to perform forward loading, consequently this constant should be selected carefully by examining the approximate number of workstation in the final balance. To examine this point the proposed fitness function (Equation 3.15) was run with five different front-loading constants (1.5, 2.0, 10, 20 and 40) to examine its effect on forward loading performance.

The number of generations permitted per workstation (G) is the second control parameter, which has a major impact on the performance and the convergence. Allocating more generations per workstation will allow the algorithm to search for better workstation assignments, but an excessive number of generations per workstation would increase the computational time unnecessarily. Five different levels of G (40, 80,120,160 and 200) were examined by terminating simulations at 1000, 2000, 3000, 4000, and 5000 generations. The influence of different levels of G

on average CPU time was also analysed to develop a relationship between the two parameters.

Fitness Function ID	Reference	Modified fitness function	
FF_2	Minagawa and Kakaz (1992) (*Line Efficiency*)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{\sum_{i=1}^{n} t_i}{mC} & l = P \end{cases}$	(4.2)
FF_3	Falkenauer & Dechmbre (1992)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{\sum_{j=1}^{m} \left(\dfrac{S_j}{C}\right)^2}{m} & l = P \end{cases}$	(4.3)
FF_4	Leu et al (1994)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{n}{\sqrt[2]{\dfrac{\sum_{j=1}^{m}(S_j - C)^2}{m} + \dfrac{\sum_{j=1}^{m}(C - S_j)}{m} + 1}} & l = P \end{cases}$	(4.4)
FF_5	Tsujimuya et al (1995)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{n}{\sqrt{\sum_{j=1}^{m}(C - S_j) + 1}} & l = P \end{cases}$	(4.5)
FF_6	Suresh et al (1996) (*Smoothness index*)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{n}{\sqrt{\sum_{j=1}^{m}(C - S_j)^2 + 1}} & l = P \end{cases}$	(4.6)
FF_7	Kim et al (1998) (*Mean absolute deviation*)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{n}{\dfrac{1}{m}\sum_{j=1}^{m}\sqrt{\lvert S_j - S_{mean}\rvert} + 1} & l = P \end{cases}$	(4.7)

Table 4.3. Modified fitness functions Where $f_i = l$

Fitness Function ID	Reference	Modified fitness function
FF_8	Sabuncuoglu et al (2000)	$\begin{cases} f_i & l < P \\ f_i + \dfrac{n}{\sqrt[2]{\dfrac{\sum_{j=1}^{m}(S_{\max}-S_j)^2}{m}} + \dfrac{\sum_{j=1}^{m}(S_{\max}-S_j)}{m} + 1} & l = P \end{cases}$
FF_9	*(BalanceEfficiency)*	$\begin{cases} f_i & l < P \\ f_i + n\left(1 - \dfrac{\sum_{j=1}^{m}\lvert S_j - S_{av}\rvert}{m \times S_{av}}\right) & l = P \end{cases}$ (4.9)

Where $f_i = l$

4.3.1.2 THE EFFECT OF THE PROBLEM COMPLEXITY ON FITNESS MODEL

Generally, cycle times, task time distributions and the structure of the precedence network govern the problem complexity, and have a major impact on the balancing problem. Previous studies (Talbot et al, 1986) showed that heuristic performances are (4.8) less affected by network structure, however it significantly varies with the magnitude of the cycle time. Cycle times close to the maximum task time and multiples of cycle times (2*C*, 3C....) are relatively hard to balance. Equally task time distributions positively skewed (figure 4.2(a)) towards the cycle time are harder than negatively skewed (figure 4.2(b)).

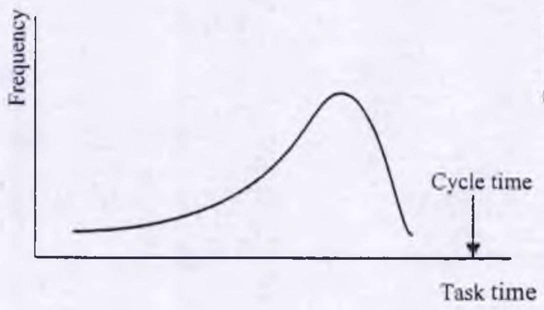


Figure 4.2(a). Positively skewed

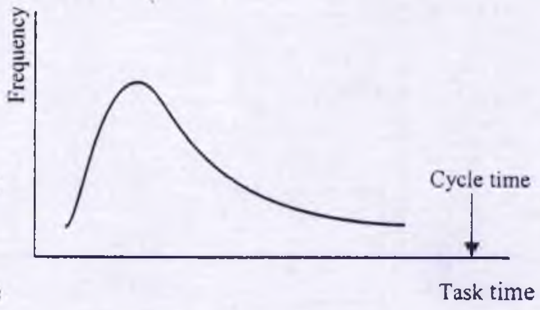


Figure 4.2(b). Negatively skewed

Moreover, precedence networks with low height to breadth ratio are generally considered as difficult problems. To find the real effect of cycle time on the proposed genetic algorithm model, a set of four-cycle times were selected from each problem consisting of both hard and easy problems. They were categorized according to Hoffmann's classification, that is, cycle times where the number of workstations (*m*) falling between the following limits is considered as hard problems, and the selected cycle times are shown in table 4.4.

$$\frac{\sum_{i=1}^{n} t_i}{2t_{\max}} \le m \le \frac{\sum_{i=1}^{n} t_i}{t_{\max}}$$

Where t_{max} is the maximum task time

Problem	Cycle time			
SH_{279}	14215	7035	4675	▨ 3500
BA_{148}	865	▨ 430	▨ 285	▨ 210
AR_{111}	30695	15195	▨ 10095	▨ 7560
MU_{94}	859	426	▨ 283	▨ 212
LU_{89}	98	50	35	▨ 25
WA^{9}_{58}	316	157	▨ 104	▨ 78

Table 4.4. Selected test cycle times

▨ Hard problems

The effect of problem complexity on the performance and the variation of computational time with the number of generations and problem size were also analysed.

4.3.2 THE INFLUENCE OF POPULATION SIZE AND INITIAL SOLUTIONS

The number of chromosomes in the population has a strong influence on the convergence of the algorithm and the computational time. A large population increases the computational time unnecessarily and, conversely, a small size of population would restrict the convergence and shows poor performance. To study this issue five populations having sizes 20, 40, 60, 80 and 100 were examined.

The characteristics of the initial population have a significant influence on the convergence. It was reported that a well seeded, or in other words, a starting population with good solutions, quickly converges to the global optimum compared to that of a poorly seeded population. Five sets of initial populations, consisting of a single solution generated by the following popular heuristic algorithms plus 39 bin-packing solutions were generated and studied.

1. Random task assignment heuristic (section3.3).
2. Rank positional weight heuristic (Helgeson and Birnie, 1961).
3. Hoffman precedence matrix procedure (Hoffmann, 1963).
4. COMSOAL (Arcus, 1966).
5. Bin packing

The default initial population of the model consists of solutions generated by the random task assignment technique and bin packing solutions. Five levels of random solutions per population (1, 5 10, 20, and 40) were examined to find the optimum ratio, keeping the population size to 40. Furthermore, the variation of CPU time with respect to population size was also studied to find the extra computational time needed due to increasing population size and also to establish a relation ship between CPU time and population time.

4.3.3 COMPARISON OF SELECTION TECHNIQUES

A number of selection techniques have been defined in the literature over the last twenty-five years and some of them experienced high selective pressure and loss of diversity as generations progress, mainly in latter part of the simulation. The number of identical chromosomes in the selection pool determines the selective pressure and the more the number of identical chromosomes the higher the selective pressure and the lower the population diversity. It also affects the selection process indirectly by consuming extra time on selecting different chromosomes for the mating process and, generally ends up with poor quality solutions.

The new selection scheme in the model was developed based on the rank-selection scheme. It was designed to overcome the above problems. In order to compare the performance of the new selection scheme, the following five previously published selection schemes including the original rank based were experimented with against the proposed technique.

1. Rank-based technique
2. Roulette wheel
3. Tournament selection
4. Random selection
5. Good and bad selection

The selective pressure of the original rank-based technique and the modified method were compared to confirm the effectiveness of the modified selection scheme.

In the modified selection scheme, the best twenty chromosomes were selected to create the selection pool and then the chromosomes were selected from the selection pool for mating. The size of the selection pool is very important for better performances. A small selection pool restricts the search to small neighbourhoods and would not allow the algorithm to explore the whole search domain and also results in high selective pressure. However, a large selection pool contains both superior and inferior chromosomes resulting in low quality solutions due to scattering. To study this issue four different sizes of selection pools containing 10, 20, 30 and 40 chromosomes were investigated to find the best selection pool size.

4.3.4 COMPARISON OF CROSSOVER TECHNIQUES

Crossover is the main reproductive operation in the model that generates new offspring. Transferring the best attributes of parent chromosomes in the assembly line balancing problem has been discussed in the literature and most of the published methods could not overcome this. However, the proposed Fixed Boundary and Variable Boundary Moving Crossover Point techniques (FBMCP, VBMCP) addressed this issue successfully. To verify this claim, the proposed techniques were tested against the following six published methods (section 2.2.3.6.2).

1. Two point crossover (TPC)
2. Single point crossover (SPC)
3. Order crossover (ORD)
4. Positional based crossover (POS)
5. Fragment crossover (FRG)
6. Uniform crossover (UNI)

The new crossover technique is not limited to the front-loading fitness function, it could be used with any fitness model developed for line balancing. To examine the validity of this statement, the fixed boundary moving crossover technique was experimented with two previously published fitness functions (*FF_3, FF_5*).

In both FBMCP and VBMCP techniques, the size of the crossover span (c_s) is constant. Small crossover spans would prevent transferring good attributes to offspring and, large crossover spans will carry both good and bad attributes of the parents to their offspring resulting in low quality solutions. Five different sizes of crossover spans ($0.5c_s$, $1c_s$, $1.5c_s$, $2c_s$, and $2.5c_s$) were studied to find the overall best crossover span ratio (i.e., crossover span size/c_s) for generating quality solutions.

4.3.5 THE EFFECT OF REPAIR TECHNIQUES

After the crossover operation, duplicate elements in the chromosomes violate the line balancing constraint assumptions. This problem has been addressed by several researchers proposing different repair procedures. Some of the procedures may consume extra computational time searching for feasible combinations. In this research the three repair procedures described in the previous chapter were studied. The number of feasible solutions in the population was found in each generation and the average number of feasible solutions created per generation was considered for comparison.

4.3.6 COMPARISON OF MUTATION TECHNIQUES

Mutation is the other genetic operatiön, which prevents the algorithm converging on a local optimum. This model includes two new mutation methods called Fixed Boundary Adjacent Mutation (FBAM) and Variable Boundary Adjacent Mutation (VBAM). The performances of the new techniques were compared with the following mutation techniques (section 3.3.3).

1. Random (Classic) Mutation (RAM)
2. Adjacent Mutation (ADM)
3. Fixed Boundary Random Mutation (FBRM)
4. Variable Boundary Random Mutation (VBRM)
5. Fixed Boundary Adjacent Mutation (FBAM)
6. Variable Boundary Adjacent Mutation (VBAM)

4.3.7 THE INFLUENCE OF ELITISM

Rudolph (1994) showed that classical genetic algorithm would never converge without elitism. The number of elite chromosomes copied to the new population has a great influence on the convergence. Alternatively, copying more elite chromosomes will decrease population diversity and leads the algorithm to settle in local optima. A set of four different elite chromosomes to total number of chromosomes ratios (0.025,0.25,0.50,and 0.75) are experimented with to find the best ratio giving a high performance.

4.4 TEST PROGRAMME SUMMARY

The complete summary of the test programme is presented in table 4.5. Sixty-five test experiments were conducted with the selected six problems to generate results for evaluation and they are presented and analysed in the next chapter to draw conclusions.

Control parameter	Number of experiments
Fitness functions	9
Front loading fitness function constant(R)	5
Generations per workstation (G)	5
Initial populations	5
Number of random solutions/population	5
Population size	5
Selection techniques	5
Crossover techniques	8
Crossover zone size	5
Mutation techniques	6
Repair techniques	3
Number of elite chromosomes per population	4
Total	65

Table 4.5. Test programme summary

4.5 SELECTION OF FACTORIAL EXPERIMENT PARAMETERS

The single factor design experiments, varying one experimental condition at a time repeated under the same conditions (section 4.3) has been applied in this research to identify and examine the leading parameter combinations.

To complete the experimental programme planned for this application of the Genetic Algorithm, single factor analysis is extended in this section to a multi-variable analysis. Experimentation with multi-variable analysis can make use of the factorial design approach, in which factors (significant major independent variables or parameters) are varied in levels or 'sub-divisions'. An experimental design with factors a t two l evels i s c alled two-factor f actorial e xperiment, t he t wo l evels b eing either quantitative or qualitative. Examples of qualitative factor variation can include 'high and low', and 'presence and absence'. A complete two-level replicate factorial analysis requires 2^k observations, generating for example the need for four thousand and ninety six tests per replicate in a 2^k factorial design (k =12) for the assembly line Genetic Algorithm review.

To manage an initial detailed examination in the application of the Genetic Algorithm to assembly lines,. the starting point is to separate and identify the significant factors and interactions. The literature helps to identify six significant parameters including the number of generations. Five parameters, fitness function, selection method, crossover and mutation techniques, initial population appear in the leading references as significant factors worthy of examination, (Dasgupta & Michalewicz(1997), Falkenauer (1998), Haput (1998)).

During the single parameter experimentation, the number of iterations was also found to generate interesting results (Chapter 5.2.2), with an identification of convergence of results at around three thousand generations. The effect of number of generations as a factorial analysis parameter is considered therefore worth examining and is added to the list of significant parameters, creating six in total. For analysis in section 5.10 section of the next chapter.

The remaining six out of twelve parameters examined at the single parameter analysis stage (front-loading constant, crossover and mutation span size, repair technique, the number of elite chromosomes and feasible solutions in the initial population) are problem specific and therefore have been held back from the factorial analysis, being identified in the further research section (Chapter 7).

4.5.1 Design of factorial experiment

The selections of appropriate levels for the six factorial parameters are based on highest and second highest results. Table 4.6 identifies parameters, levels and nomenclatures used in the remainder of this chapter.

A complete replicate of the 2^k factorial analysis requires would have required sixty four runs, examining six of the 63 degrees of freedom correspond to main effects plus fifteen degrees of freedom correspond to two-factor interactions. The remaining forty two degrees of freedom are associated with three-factor and higher interactions.

The single parameter investigation and review of factorial experiment design test cases supports the use however of a half-fraction of the 2^6 design, an approach consistent with the concepts of experimental design (Montgomery, 2001). The construction of the 2^{6-1} design is shown in table 4.7. The design was constructed by writing down the basic design having 32 runs (2^5 design in *A, B, C, D* and *E*), selecting *ABCDEF* as the generator, and then setting the levels of the sixth factor *F=ABCDE*. The defining relation for this design is *I = ABCDEF* and every main effect is aliased with a single five-factor interaction, and every two-factor interaction is aliased with a single four-factor interaction. Thus, the design is of resolution VI (the degree to which estimated main effects are aliased or confounded).

The WE_{58} problem (cycle time = 56) is selected for the experiments, exhibiting a suitable size and complexity of precedence relationship. This is supported by previous experimentation with this problem, which identified a considerable change in the number of solution stations (Chapter 3, figure 3.19).

Factor	Highest (+)	Next High (-)	Reference
Fitness function (*A*)	FF_1 (Front loading fitness function) [*a*]	FF_5 (Tsujimuya et al., 1995)	Chapter 5.1 (Figure 5.9)
Crossover technique (*B*)	Variable Boundary Moving Crossover Point (VBMCP) [*b*]	Uniform Crossover(UNI) Positional based crossover (POS)	Chapter 5.5.1 (Figure 5.38)
Selection method (*C*)	Modified Rank Based Selection method (MRBS) [*c*]	Rank-based selection method (RBS)	Chapter 5.4 (Figure 5.30)
Number of generations (*D*)	2000 [*d*]	6000	Chapter 5.2.2 (Figure 5.19)
Mutation technique (*E*)	Variable Boundary Adjacent Mutation technique (VBAM) [*e*]	Fixed Boundary Adjacent Mutation(FBAM)	Chapter 5.7.1 (Figure 5.43)
Population size (*F*)	40 [*f*]	60	Chapter 5.3.1 (Figure 5.26)

[*x*] Shows the treatment with factors *X*. at the high level

Table 4.6. Line Balancing GA Parameters and Levels

The analysis of variance (ANOVA) is used for data analysis and all terms with a *p*-value higher than 0.05 are rejected as their effects are negligible. With more than four parameters under investigation, the widely available MINITAB© software package is employed to support analysis of results.

Run	Basic Design						Treatment
	A	*B*	*C*	*D*	*E*	*F=ABCDE*	Combination
1	-	-	-	-	-	-	*l*
2	+	-	-	-	-	+	*af*
3	-	+	-	-	-	+	*bf*
4	+	+	-	-	-	-	*cb*
5	-	-	+	-	-	+	*c*
6	+	-	+	-	-	-	*ac*
7	-	+	+	-	-	-	*bc*
8	+	+	+	-	-	+	*abcf*
9	-	-	-	+	-	+	*df*
10	+	-	-	+	-	-	*ac*
11	-	+	-	+	-	-	*ad*
12	+	+	-	+	-	+	*abdf*
13	-	-	+	+	-	-	*cd*
14	+	-	+	+	-	+	*acdf*
15	-	+	+	+	-	+	*bcdf*
16	+	+	+	+	-	-	*abcd*
17	-	-	-	-	+	+	*ef*
18	+	-	-	-	+	-	*ae*
19	-	+	-	-	+	-	*be*
20	+	+	-	-	+	+	*abef*
21	-	-	+	-	+	-	*ce*
22	+	-	+	-	+	+	*acde*
23	-	+	+	-	+	+	*bcef*
24	+	+	+	-	+	-	*abce*
25	-	-	-	+	+	-	*de*
26	+	-	-	+	+	+	*adef*
27	-	+	-	+	+	+	*bdef*
28	+	+	-	+	+	-	*abde*
29	-	-	+	+	+	+	*cdef*
30	+	-	+	+	+	-	*acde*
31	-	+	+	+	+	-	*bcde*
32	+	+	+	+	+	+	*abcdef*

Table 4.7. A 2^{6-1} Design and treatment combination

CHAPTER 5

RESULTS EVALUATION

The experiments performed to evaluate the applicability and effectiveness of the proposed model for simple assembly line balancing problems are presented in this chapter. Comparison is made between the proposed genetic algorithm features with existing ones to confirm their superiority.

Thirteen test experiments have been conducted on 65 parameters and, the simulation results have been recorded at one hundred generation intervals up to three thousand generations generating total 11,700 test data (6 problems x 65 parameters x 30 generations), eventually, 48 previously published test problems were solved to show its applicability to a wide range of problems. The test results are presented in the form of tables and graphs in the order presented in Chapter 4. The results are discussed and specific comments are made after each evaluation.

An overall discussion of results and the final conclusions drawn on the genetic algorithm line-balancing model for solving generalized simple assembly line balancing problems are presented in the next chapter.

5.1 FITNESS FUNCTION COMPARISON

High line efficiency signifies positive achievement in line utilization and is the key indication of economic performance, with one hundred percent representing the best achievable. The average line efficiency performance of the nine fitness functions (FF_1,FF_2,...,FF_9) over the number of generations is shown in figures 5.1, 5.2, 5,3 and 5.4.

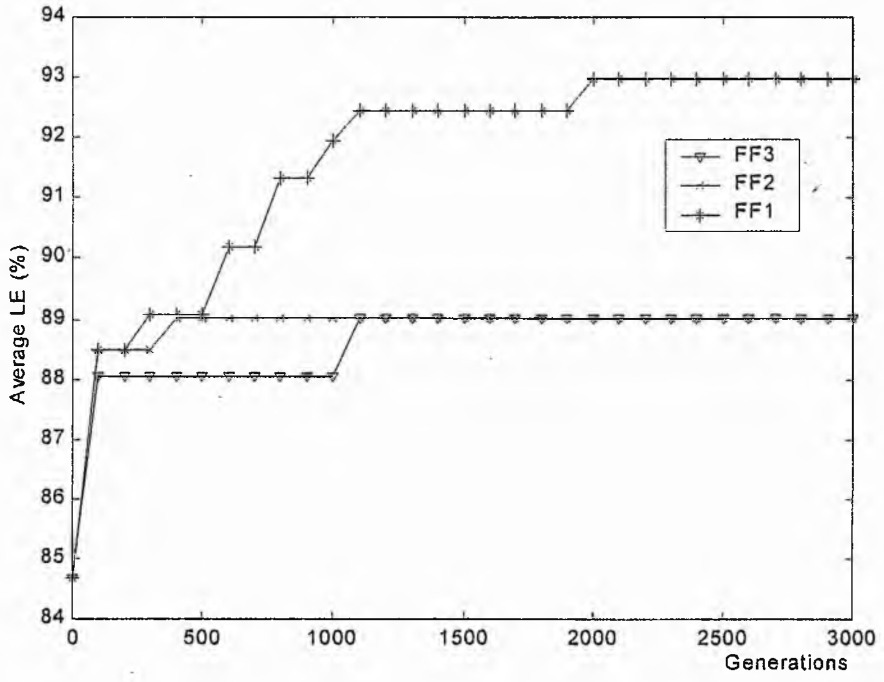


Figure 5.1. Average line efficiency performances of fitness functions FF_1, FF_2 and FF_3

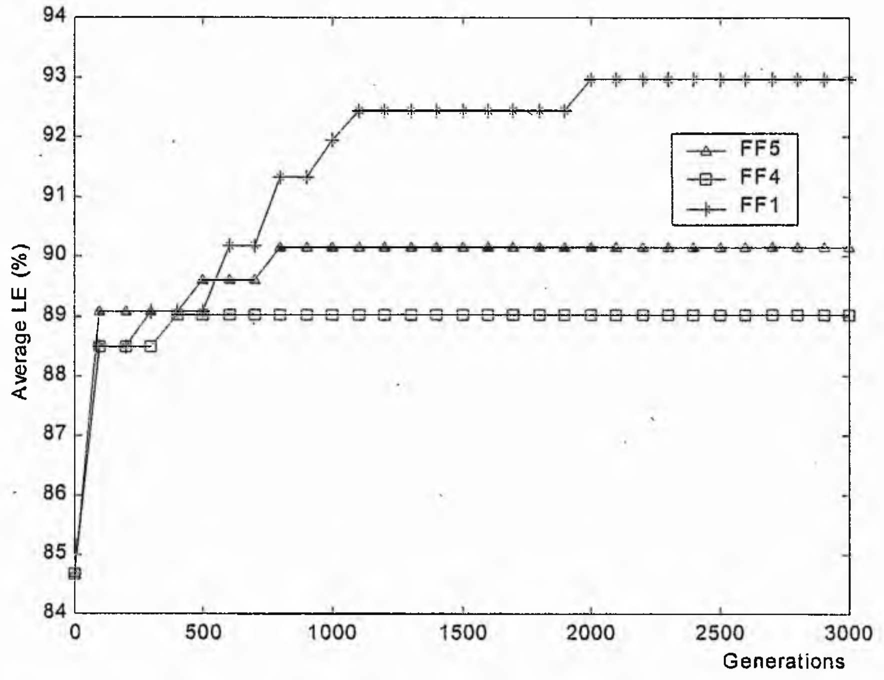


Figure 5.2. Average line efficiency performances of fitness functions FF_1, FF_4 and FF_5

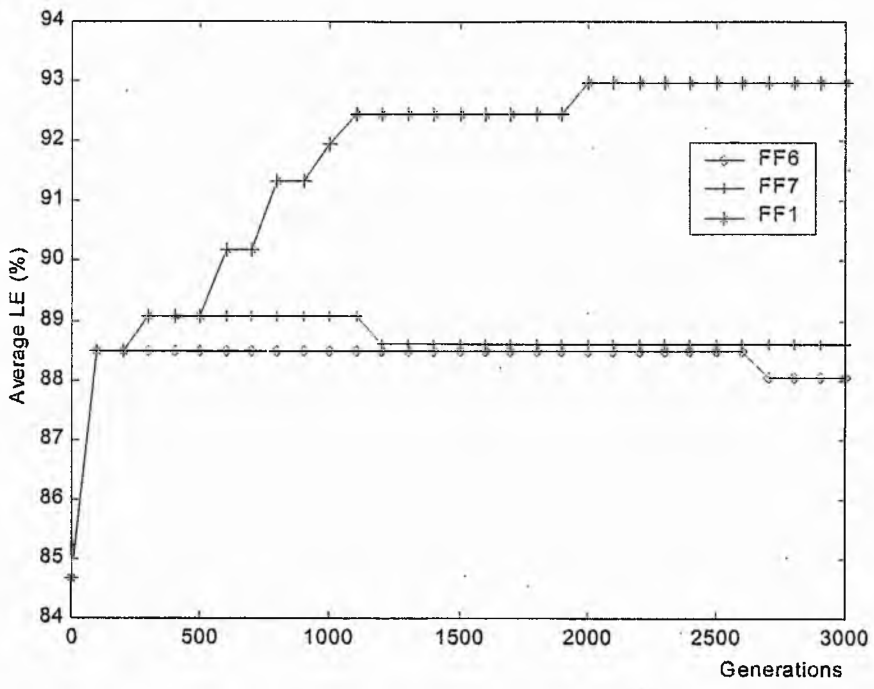


Figure 5.3. Average line efficiency performances of fitness functions FF_1, FF_6 and FF_7

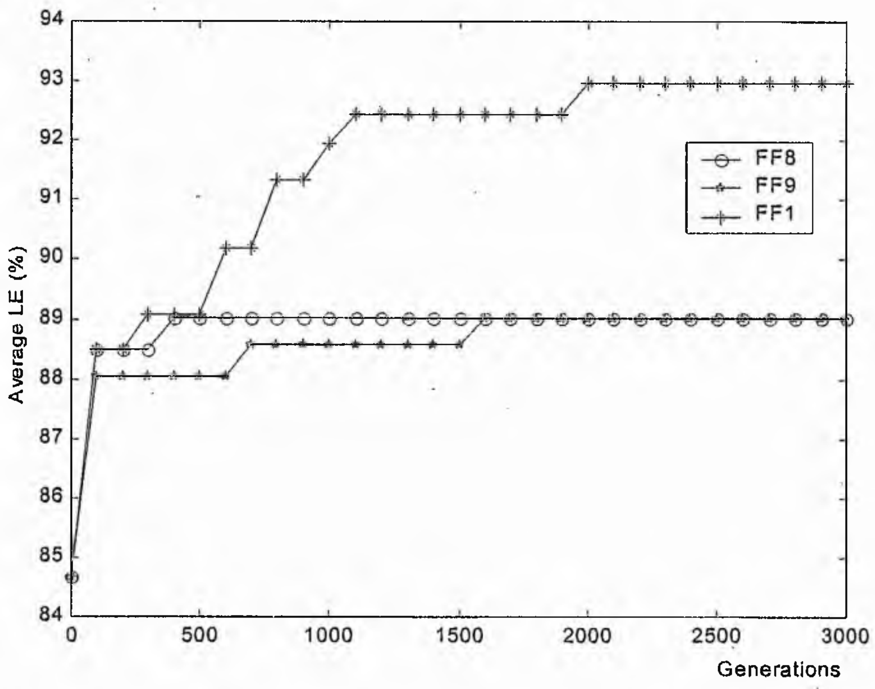


Figure 5.4. Average line efficiency performances of fitness functions FF_1, FF_8 and FF_9

All fitness functions have improved the initial solution as the evolution proceeds. In all cases, a significant increase of average line efficiency can be seen over the first one hundred generations. The Fitness function FF_1 showed the highest average line efficiency, and its increase was substantial mainly during first the 1,200 generations. However, the improvement was gradual thereafter, and finally reached just under 93%. The second best performance was shown by FF_5 (just over 90%), and there was hardly any increase in average line efficiency beyond 750 generations. Fitness functions FF_2, FF_3, FF_4, FF_8 and FF_9 all converged to the same value (89^+%) after initial steep increases.

The fitness functions FF_6 and FF_7 improved the initial solution over 100 and 200 generations respectively but subsequently decrease the quality of the solution. One possible explanation is both these fitness functions are trying to achieve the primary line-balancing objective (i.e., minimizing the number of workstations) by reducing the station time variation.

Graphs shown in figures 5.5, 5.6, 5.7, and 5.8 display the average balance efficiency performances. Balance efficiency is representative of the distribution of workload with consequent personnel satisfaction combined with increased opportunities for greater output, and is considered as the secondary objective of line balancing.

All fitness functions showed a marked increase in average balance efficiency over the first 50-100 generations. Fitness function FF_7 showed the highest performance and FF_9, which is based on a pure balance efficiency definition, shows the second best. Fitness functions, FF_2, FF_5, FF_7, FF_8 and FF_9 displayed similar characteristics and after steep initial increases, they all showed gradual increases in average balance efficiency up to 2000 generations. However, beyond 2,500 generations, the increase was marginal in FF_8 and FF_9 and the others remained constant.

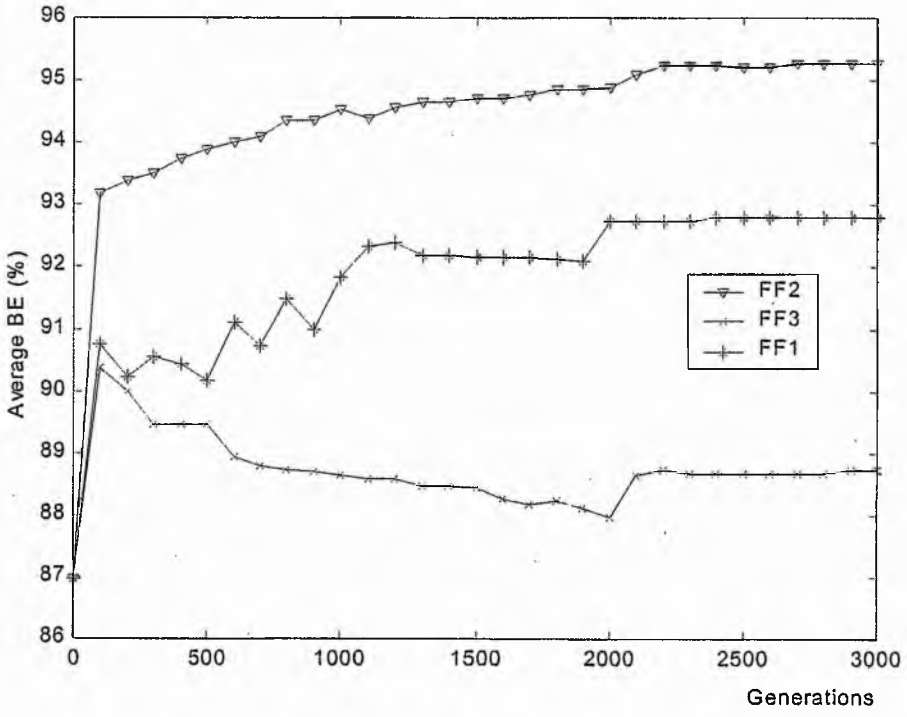


Figure 5.5. Average balance efficiency performances of fitness functions FF_1, FF_2 and FF_3

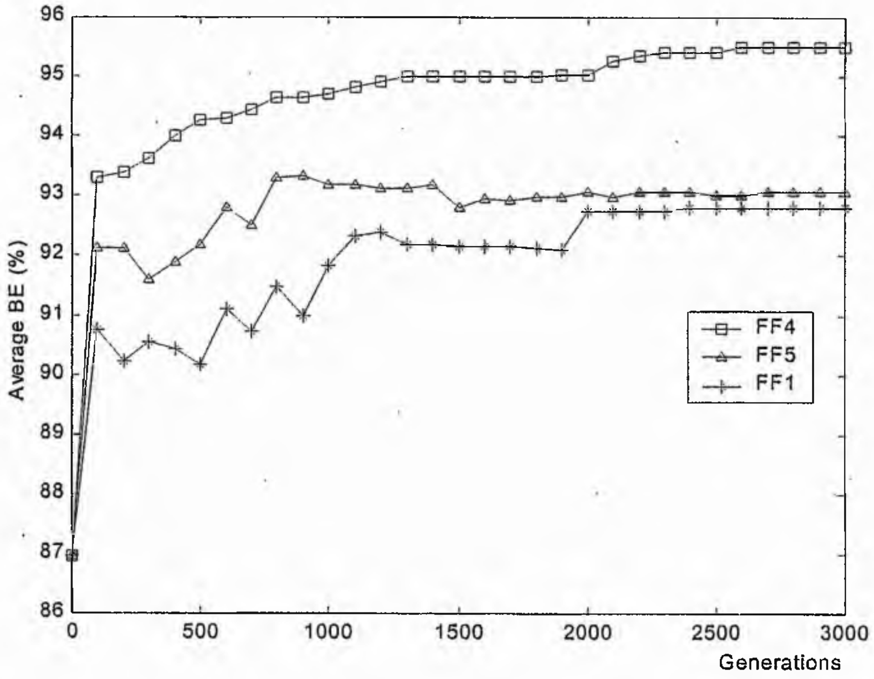


Figure 5.6. Average balance efficiency performances of fitness functions FF_1, FF_4 and FF_5

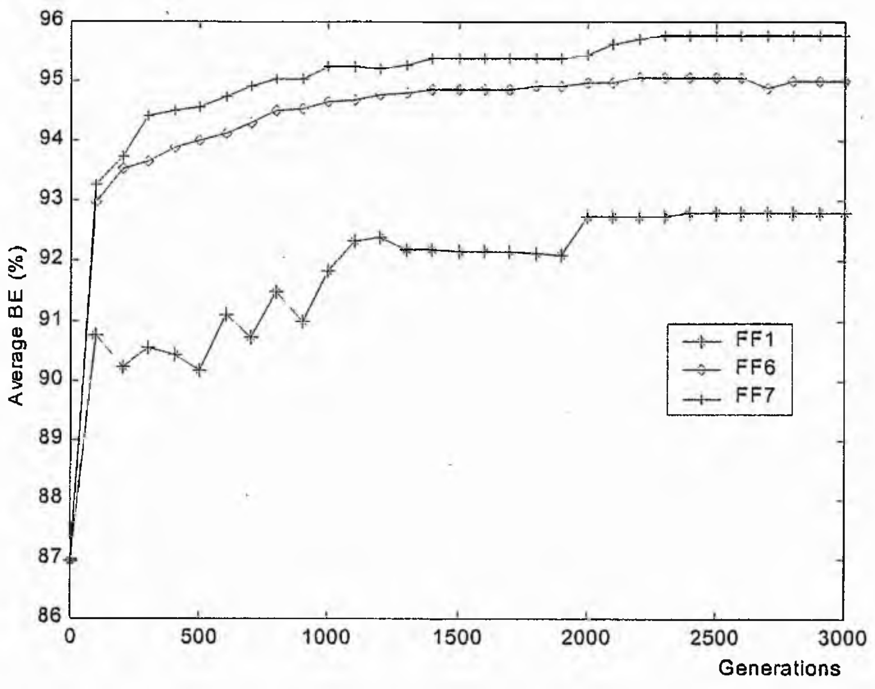


Figure 5.7. Average balance efficiency performances of fitness functions FF_1, FF_6 and FF_7

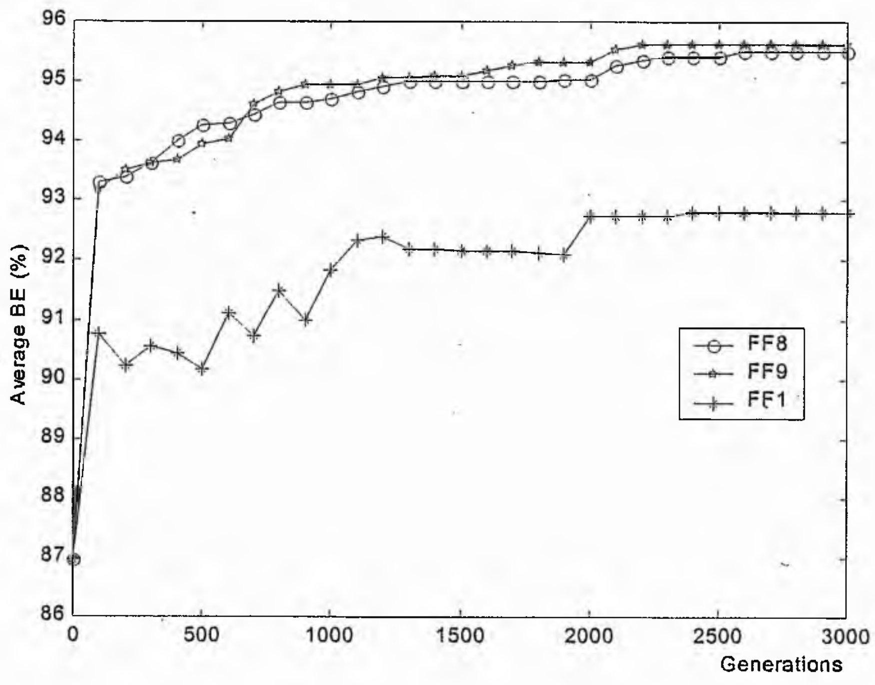


Figure 5.8. Average balance efficiency performances of fitness functions FF_1, FF_8 and FF_9

Fitness function FF_1, which is based on the front-loading concept generally showed low balance efficiency. After an initial boost in average balance efficiency, there was a gradual increase with ± 1% fluctuation (between 50-2,000 generations). This is mainly because the front loading process forces the algorithm to assign elements to early workstations if possible, leading to significant variation of workload particularly in the last few workstations. The improvement after 2,500 generations was negligible. The lowest balance efficiency was obtained by fitness function FF_3 and it showed gradual decrease in balance efficiency after 50 generations.

A summary of average line efficiency obtained by the fitness functions is shown in figure 5.9. As can be seen from the bar chart, the proposed front-loading fitness function showed the highest of 92.96%. Fitness function FF_5 developed by Tsujimura et al (1995) exhibited the best among the published fitness function models (90.16%). The lowest average line efficiency was obtained by FF_7, which is based on mean absolute deviation. Thus, findings indicate that the proposed fitness model outperformed selected the published fitness models with an average improvement of 2.80% more than FF_5 (second best).

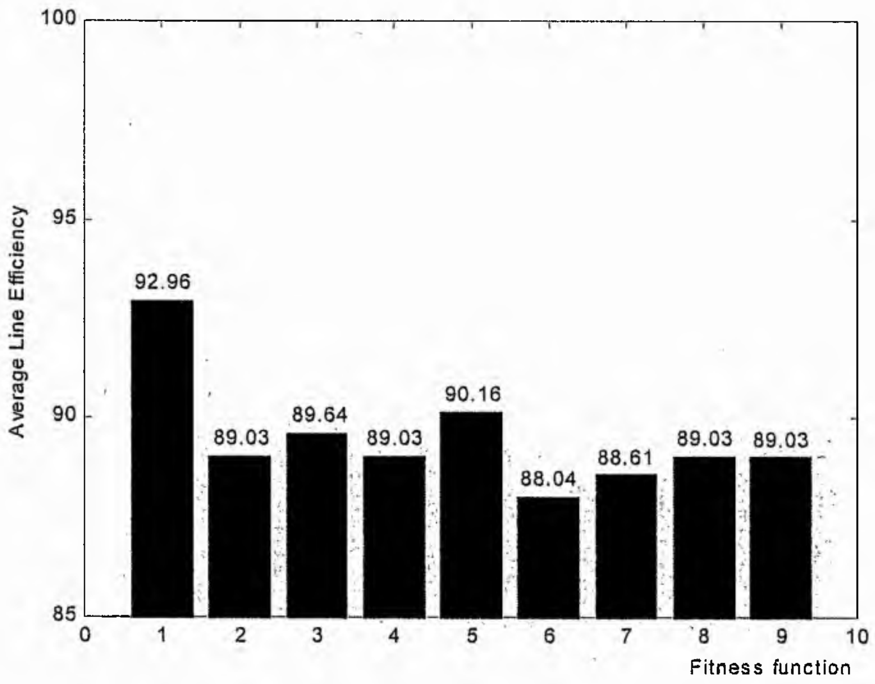


Figure 5.9. Average line efficiency variation

Figure 5.10 summaries the average balance efficiency obtained by the fitness models. The highest average was obtained by fitness function FF_7, (95.77%) which is 0.16% above the second best obtained by the fitness function FF_8, developed on a pure balance efficiency definition. Five fitness function models showed a high average, over 95.5%, at the expense of average line efficiency. The front-loading fitness function displayed a low average of 92.80% and it is 4.08% above the lowest. FF_3 proposed by Falkenauer and Delchambre (1992) obtained the lowest of 88.72%.

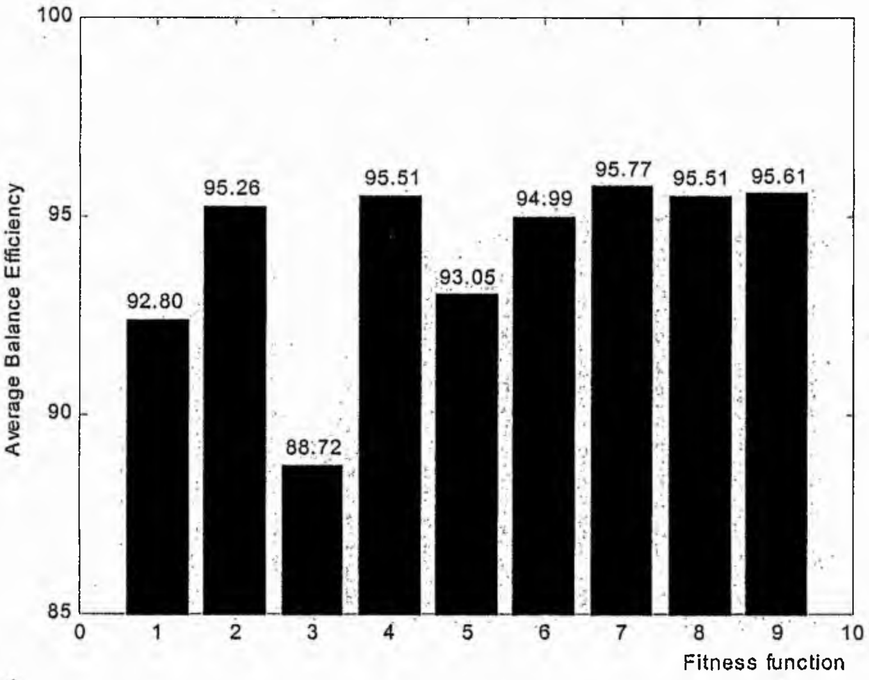


Figure 5.10. Average balance efficiency variation

Table 5.1 shows the number of workstations in the final solution obtained by nine fitness models. All the six test problems have the same theoretical minimum number of workstations (m^*) of 25. In cases where the number of workstations is greater than the m^*, the difference is specified following the '+' sign.

The fitness function FF_1 showed the least number of workstations above m^*. In cases, MU_{94}^{176} and AR_{111}^{6269} it obtained the theoretical minimum number of workstations and, in cases LU_{89}^{20}, BA_{148}^{170} and SH_{297}^{2787} it was one workstation above the

m^*. The total number of workstations above the m^* obtained by all the other fitness functions was twice or more that obtained by FF_1. These findings show the new model's power and applicability for solving simple assembly line balancing problems.

Fitness function	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
FF_1	27^{+2}	26^{+1}	25	25	26^{+1}	26^{+1}
FF_2	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}
FF_3	29^{+4}	26^{+1}	26^{+1}	28^{+3}	27^{+2}	27^{+2}
FF_4	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}
FF_5	29^{+4}	27^{+2}	26^{+1}	26^{+1}	27^{+2}	27^{+2}
FF_6	30^{+5}	27^{+2}	27^{+2}	28^{+3}	27^{+2}	27^{+2}
FF_7	30^{+5}	27^{+2}	26^{+1}	28^{+3}	27^{+2}	27^{+2}
FF_8	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}
FF_9	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

Table 5.1. Number of workstations in the final solution

Maximum fitness of a population is a good measure of assessing the performances and convergence of the genetic algorithm. In the proposed model, forward and backward loading results in increasing and decreasing fitness respectively, and decreasing the number of workstations shows a significant increase in fitness. Figures 5.11 to 5.16 show the performance of the front-loading fitness function on individual test cases. The change in the number of workstations is also shown below each fitness curve to illustrate its effect on maximum fitness. For all cases, FF_1 showed a stepwise increase in maximum fitness. Small increases were due to a station filling up and large step increases appeared when the number of stations in the solution decreased.

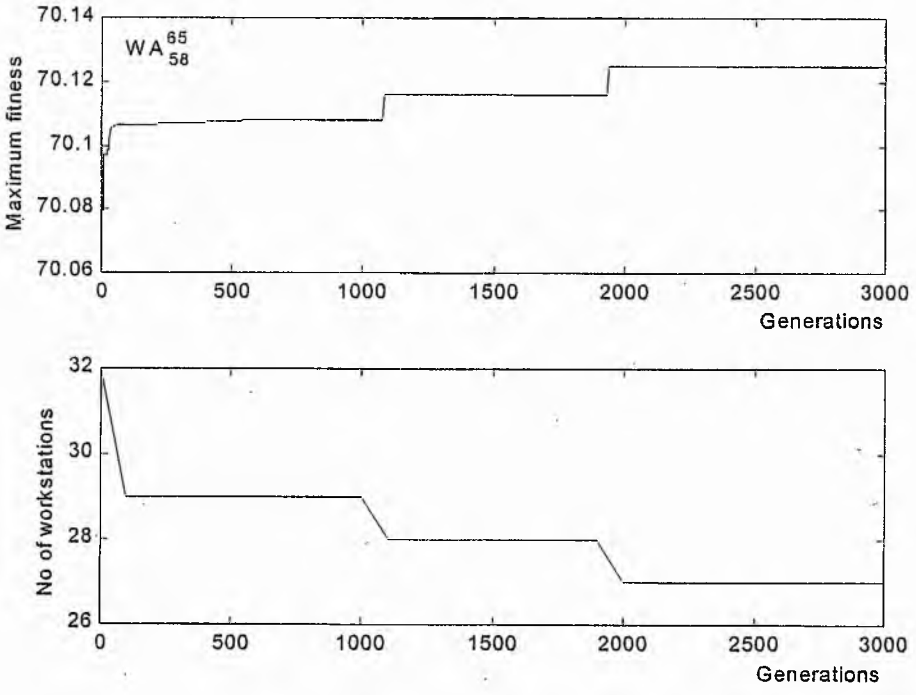


Figure 5.11. Maximum fitness and the number of workstations against generations for the WA_{58}^{65} problem

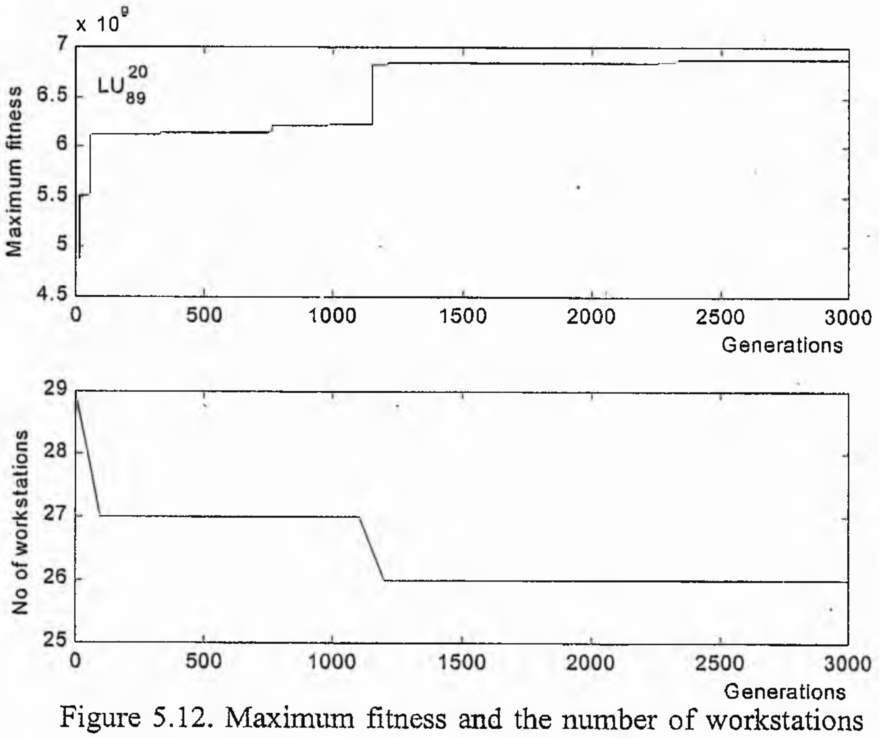


Figure 5.12. Maximum fitness and the number of workstations against generations for the LU_{89}^{20} problem

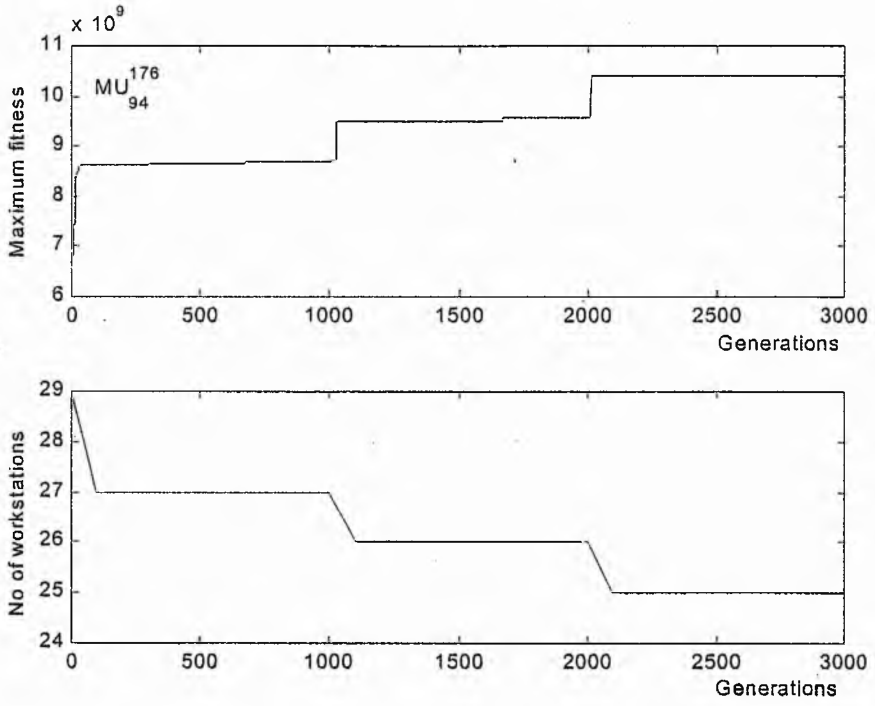


Figure 5.13. Maximum fitness and the number of workstations against generations for the MU_{94}^{176} problem

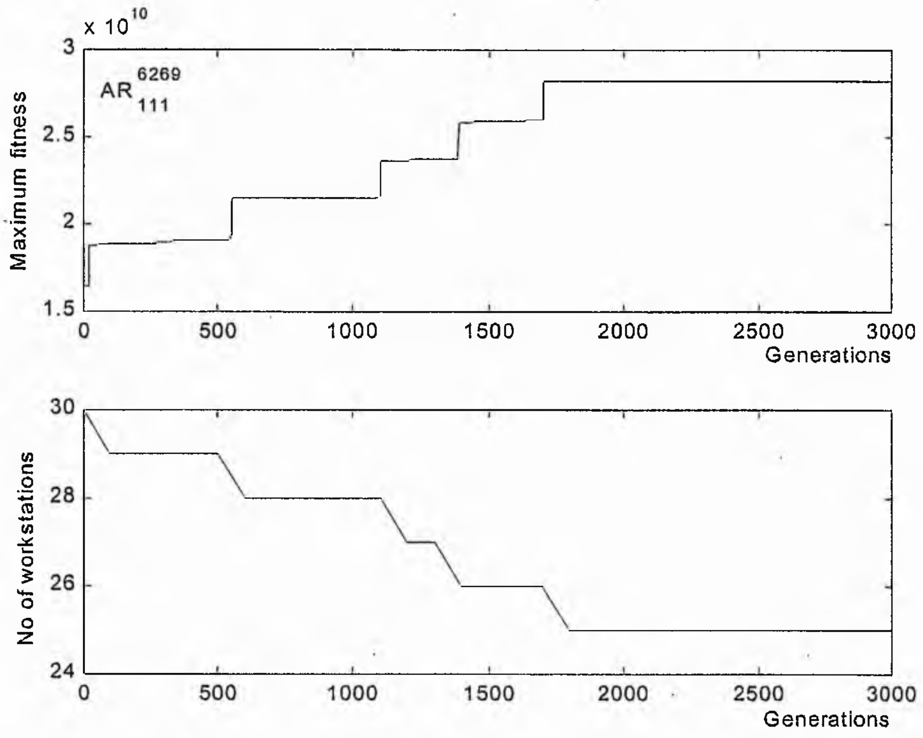


Figure 5.14. Maximum fitness and number of workstations against generations for the AR_{111}^{6269} problem

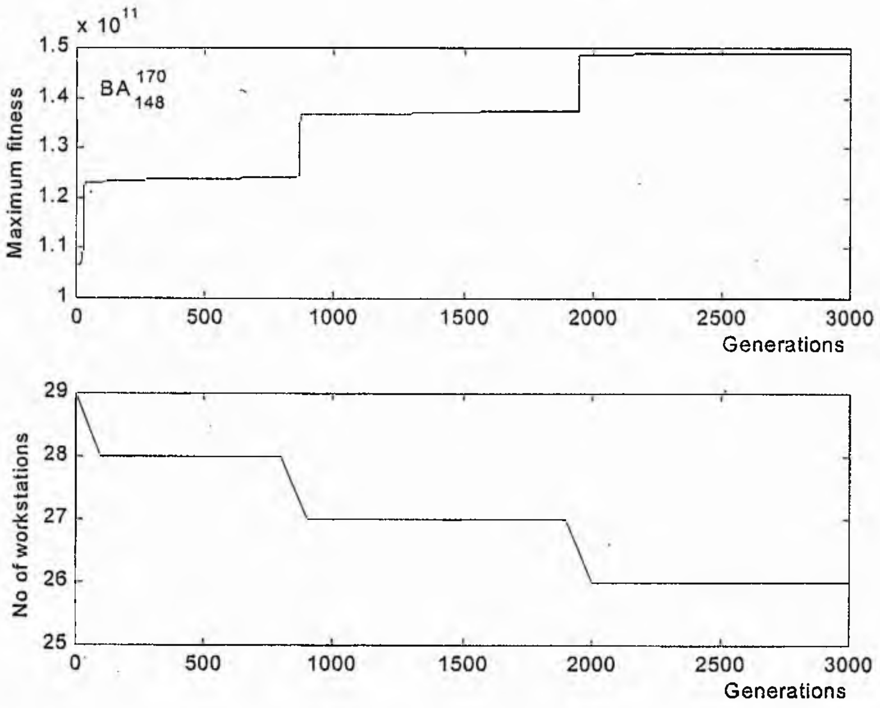


Figure 5.15. Maximum fitness and the number of workstations against generations for the BA_{148}^{170} problem

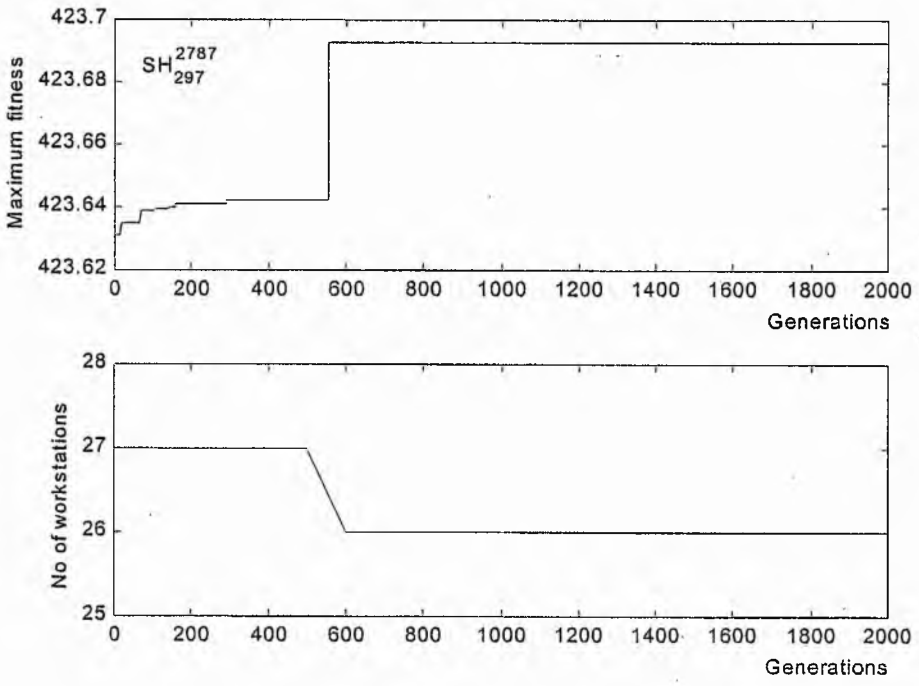


Figure 5.16. Maximum fitness and the number of workstations against generations for the SH_{297}^{2787} problem

After about 2,500 generations there was hardly any change in maximum fitness, indicating the further shifting of elements to earlier workstations was restricted. This suggests that, 2500 generations is the best termination value for the selected test cases irrespective of the number of elements in the problem.

5.2 THE INFLUENCE OF FITNESS FUNCTION PARAMETERS

5.2.1 FRONT LOADING CONSTANT

The value of the front-loading constant (*R*) determines the propensity of elements to be shifted to front stations. It was shown that this value must be greater than unity, in order to satisfy the front-loading concept (equation 3.9). Theoretically, increasing *R* can force the algorithm for heavy front loading, but its upper bound must be less than a certain value for good performance. This upper bound is normally dependent on the approximate number of workstations in the final solution. Table 5.2 shows the number of workstations obtained in the final solution for different front-loading constants. Figure 5.17 displays the average number of workstations above the optimum solution.

Front loading constant	Test problem					
(*R*)	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
1.5	26^{+1}	26^{+1}	25	25	26^{+1}	26^{+1}
2.0	27^{+2}	27^{+2}	25	26^{+1}	26^{+1}	27^{+2}
10.0	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	28^{+3}
20.0	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	28^{+3}
40.0	29^{+4}	28^{+3}	28^{+3}	27^{+2}	27^{+2}	28^{+3}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

Table 5.2. Number of workstations in the final solution

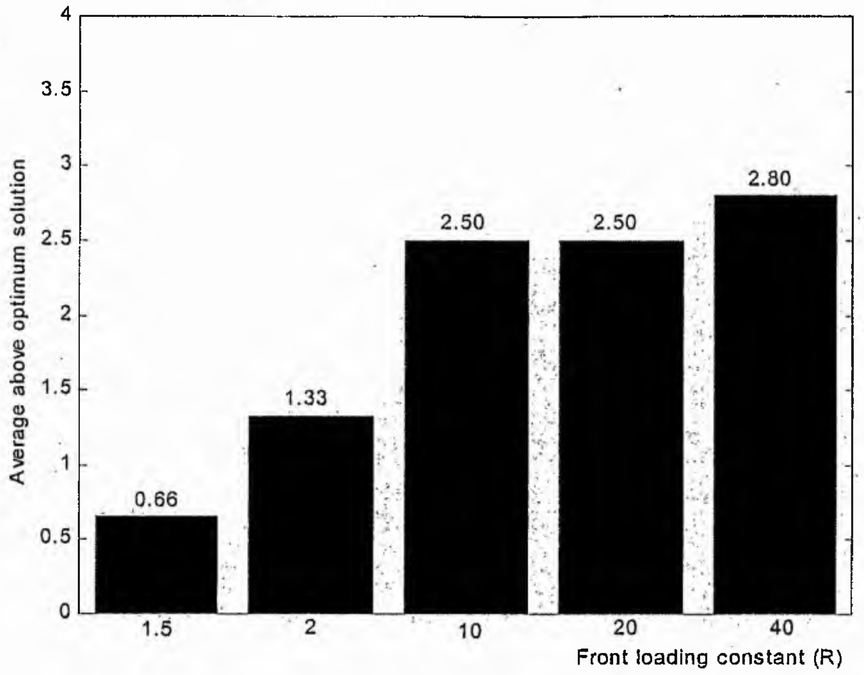


Figure 5.17. Average number of workstations above the optimum

When R = 1.5, the fitness model indicated the lowest average above the optimum of 0.66 and, in problems MU_{94}^{176} and AR_{111}^{6269}, it yielded the optimum solutions. The other four problems reported one stations above the optimum. As the front-loading constant increases, the quality of the solutions became poorer and, after R >10 the number of workstations in the final solution were about 3-4 workstations above the optimum.

The above results can be explained by considering the fitness change per unit element plot shown in figures 5.18(a)-(d) for different R-values. All possibilities of transferring elements between two stations are considered for the problem BA_{148}^{170}, and the fitness change (ΔZ) was plotted against relevant workstations. It can be clearly seen that, for higher values of R, the change of fitness, even after the first few workstations, was constant (figure 5.18(d)), resulting in a very low force on the algorithm for forward loading, but within the first few stations, it showed a significant change. A small R eliminates this problem (figure 5.18(a)) and, therefore is recommended for test problems with a large number of workstations in the final solution. However, high front-loading constant values also show excellent performance for test cases with small number of workstations. These results are consistent with the values suggested in section 3.2.2.4.

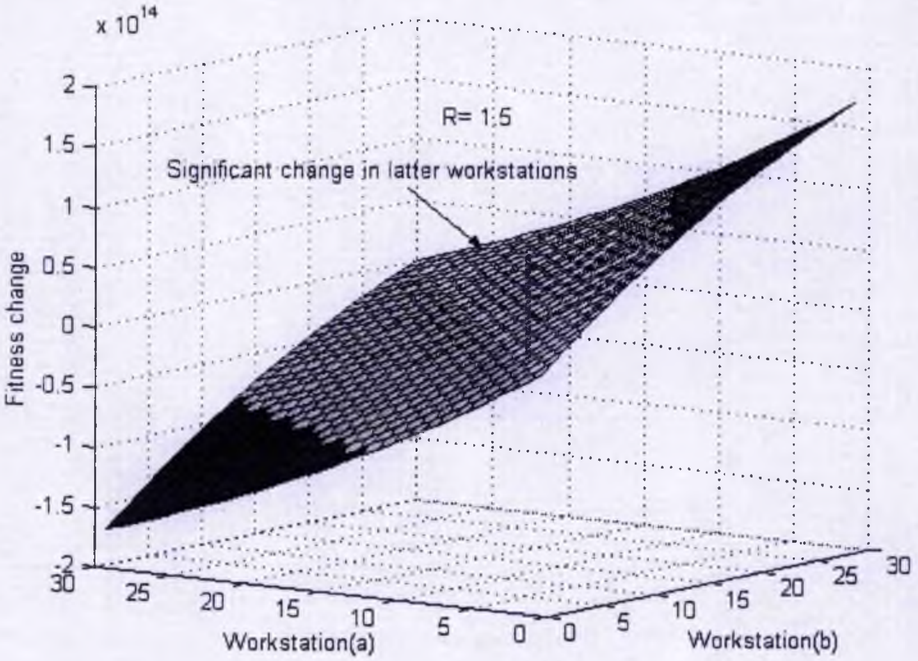


Figure 5.18(a). Fitness change plot for $R = 1.5$

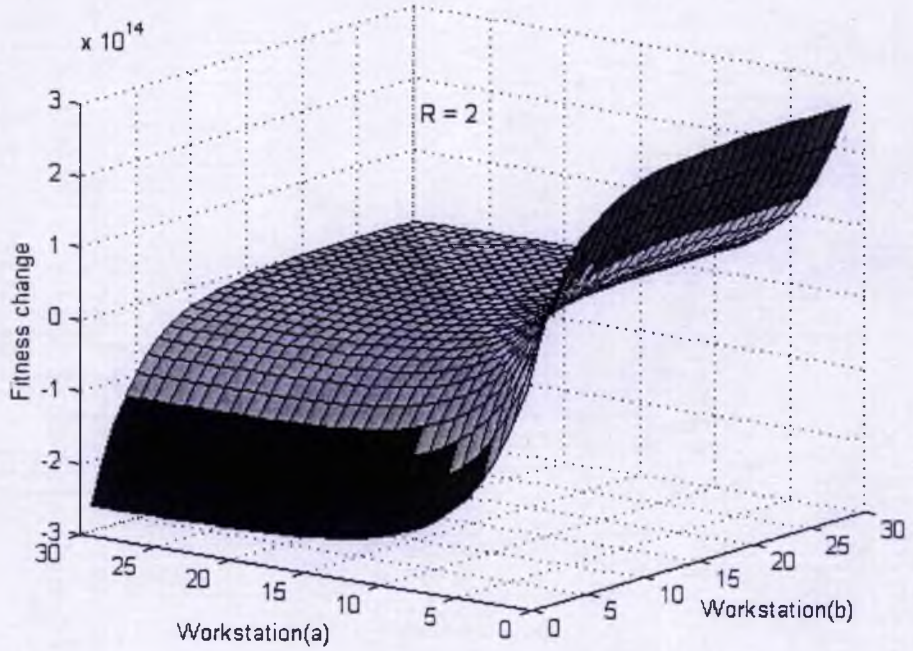


Figure 5.18(b). Fitness change plot for $R = 2.0$

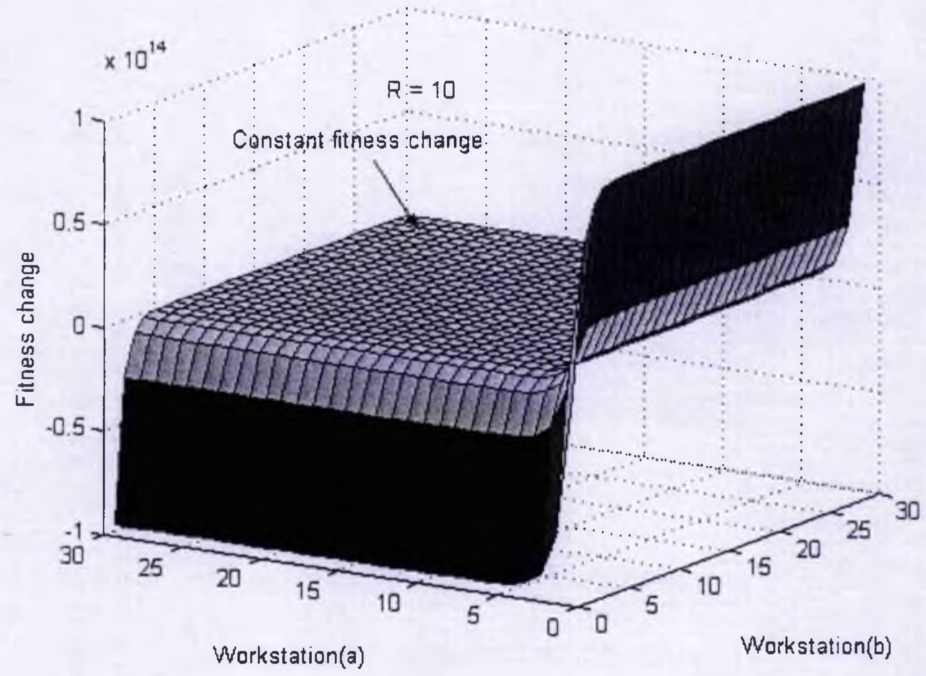


Figure 5.18(c). Fitness change plot for $R = 10.0$

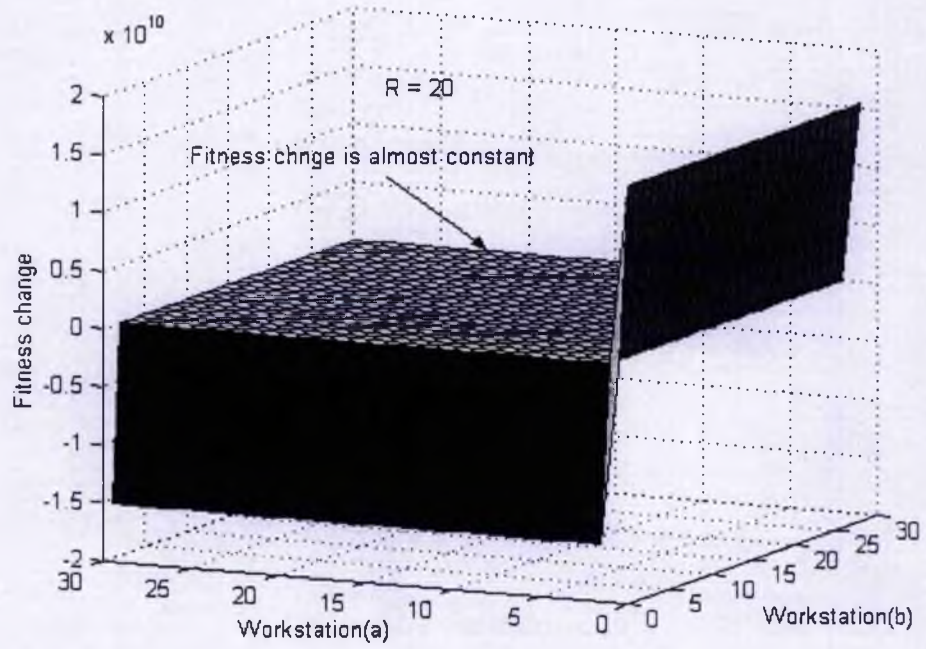


Figure 5.18(d). Fitness change plot for $R = 20.0$

5.2.2 THE INFLUENCE OF NUMBER OF GENERATIONS PERMITTED PER WORKSTATION

As described in the previous chapter, the number of generations permitted per workstation (*G*) is the second most significant parameter, which influences the performance. Table 5.3 shows the number of workstations in the final solution after running the algorithm for five different termination levels. These results are summarized and presented in the figure 5.19.

Figure 5.19 shows that for the number of generations per workstation on and above 120, the algorithm showed the lowest average above the optimum of 0.66. Both test problems MU_{94}^{176} and AR_{111}^{6269} obtained optimal solutions after 120 generations per workstation and all the other four problems reported one workstation above the optimum indicating a sufficient number of iterations is required for reaching better solutions. The highest average above the optimum was obtained for 40 generations per workstation and this is mainly due to an insufficient number of generations for the algorithm to do enough forward loading.

Number of generations per station	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
40	28^{+3}	27^{+2}	26^{+1}	27^{+2}	26^{+1}	27^{+2}
80	27^{+2}	27^{+2}	25	27^{+2}	26^{+1}	27^{+2}
120	26^{+1}	26^{+1}	25	25	26^{+1}	26^{+1}
160	26^{+1}	26^{+1}	25	25	26^{+1}	26^{+1}
200	26^{+1}	26^{+1}	25	25	26^{+1}	26^{+1}

Number of workstations above the optimum is specified following the '+' sign.

Table 5.3. Number of workstations in the final solution

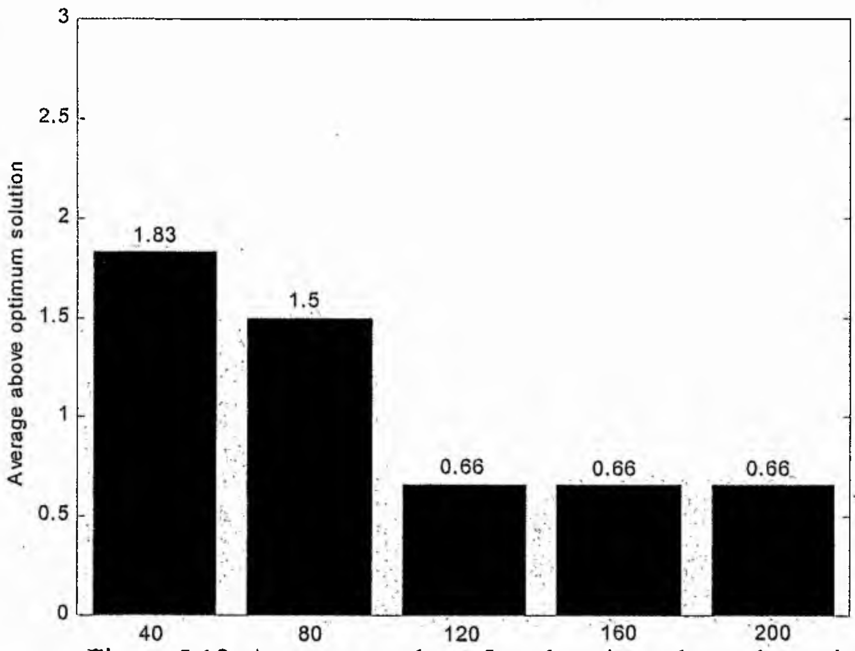


Figure 5.19. Average number of workstations above the optimum

Table 5.4 shows the CPU time taken for different levels of *G* (40, 80, 120, 160 and 200). Running the algorithm for 1000, 2000, 3000, 4000 and 5000 generations (considering 25 workstations in the final solution) obtained the above *G* values. Figure 5.20 displays the average CPU time for different termination levels.

Total number of generations (*G*)	Test problem					
	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
1000(40)	19.72	30.0	32.0	35.6	49.5	122.3
2000(80)	70.1	54.0	113.2	71.3	96.5	248.9
3000(120)	62.2	87.5	183.8	107.9	135.6	260.7
4000(160)	82.7	113.2	236.8	138.57	187.1	478.3
5000(200)	94.4	146.1	290.5	173.3	231.3	584.3

Table 5.4. CPU time in seconds for different termination levels

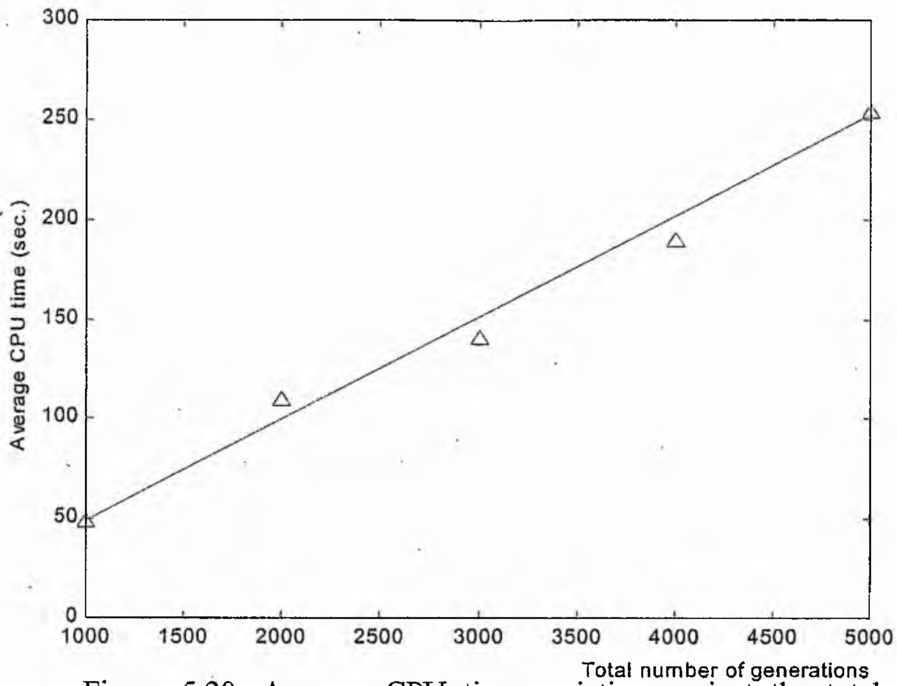


Figure 5.20. Average CPU time variation against the total number of generations

From figure 5.20 it can be seen that the termination level was correlated with average CPU time showing a linear relationship. This implies that the model could be run for a long time if needed, without the CPU time growing exponentially. Based on the above findings, it can be concluded that, the overall best number of generations for the studied test problems (where m*< 25) is 3000 generations and running the algorithm beyond would hardly improve the quality of the solution.

Additionally, the capability of the new fitness model in addressing the primary objective of the assembly line balancing problems is confirmed by solving the well-known four test cases (KW_{45}^{69}, KW_{45}^{92}, KW_{45}^{138}, and KW_{45}^{184}) from the 45-element Kilbridge and Wester problem. Eight selected fitness functions and the proposed fitness function (FF_1) were applied to solve the above four problems, and after 3000 generations the number of workstations in the final solution were recorded and they are shown in table 5.5.

C	m^*	Fitness function								
		FF_1	FF_2	FF_3	FF_4	FF_5	FF_6	FF_7	FF_8	FF_9
69	8	8	9	9	9	9	9	9	9	9
92	6	6	7	7	7	7	7	7	7	7
138	4	4	5	5	5	5	5	5	5	5
184	3	3	4	4	4	4	4	4	4	4

C is the cycle time and m^* is the optimum number of workstations

Theoretical minimum number of workstations

Table 5.5. Number of workstations after 3000 generations

Table 5.5 identifies the power of the proposed fitness function FF_1 by solving all four problems optimally, whereas the other eight fitness models solved the problems with an extra cost of a workstation. Figures 5.21-24 represent the station time distributions at generations 1, 1000, 2000 and 3000 for the four problems with the fitness function FF_1.

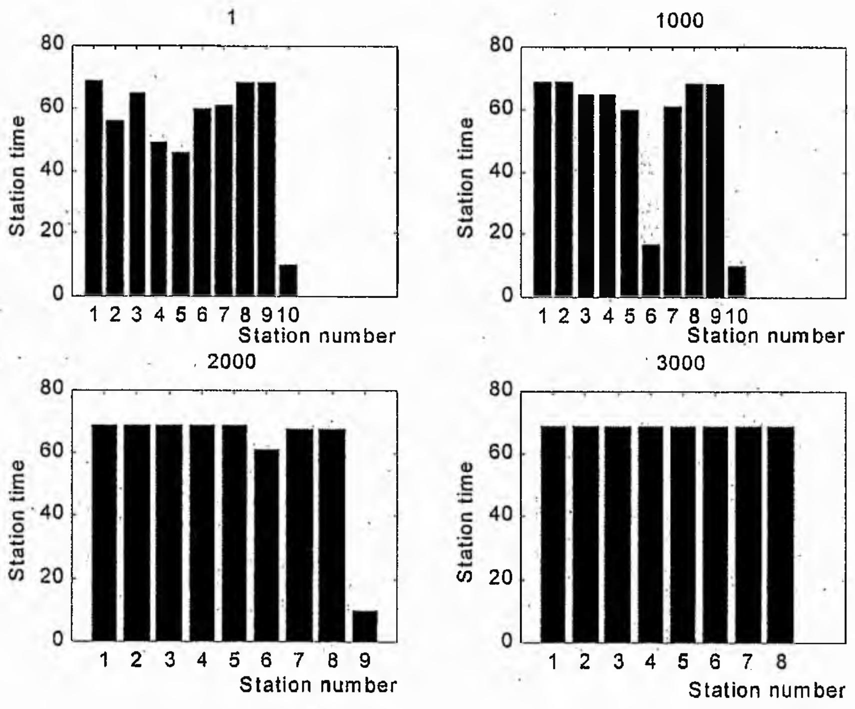


Figure 5.21. Station time distribution after 1,1000,2000, and 3000 generations for the problem KW_{45}^{69}

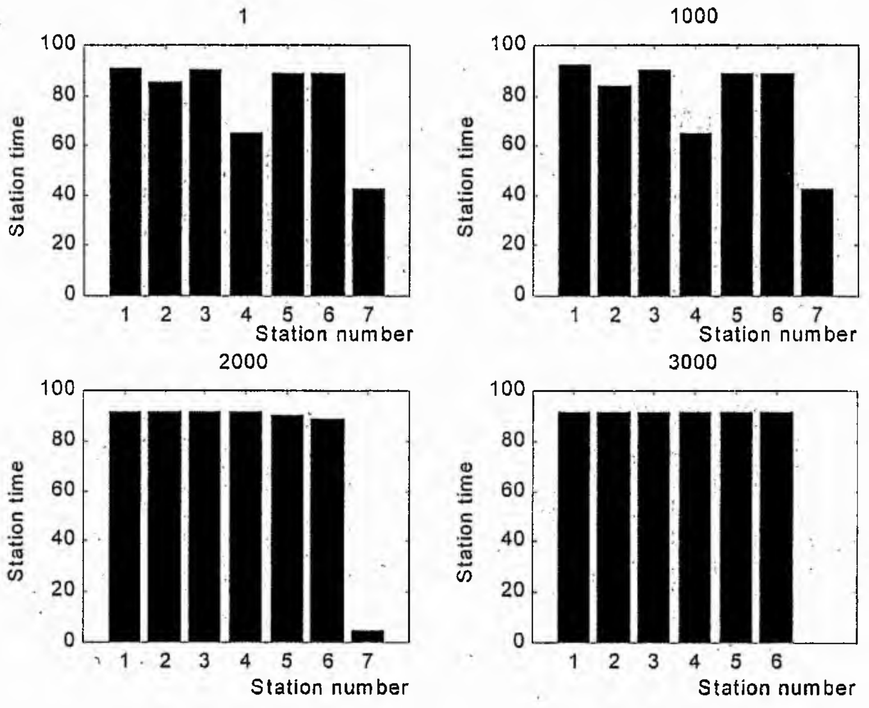


Figure 5.22. Station time distribution after 1,1000,2000, and 3000 generations for the problem KW_{45}^{92}

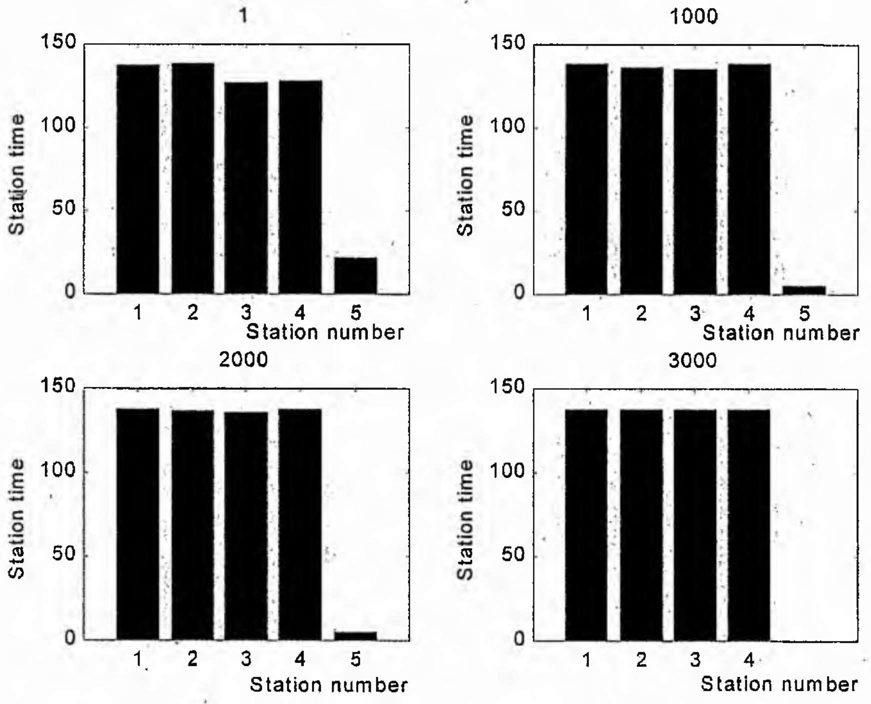


Figure 5.23. Station time distribution after 1,1000,2000, and 3000 generations for the problem KW_{45}^{138}

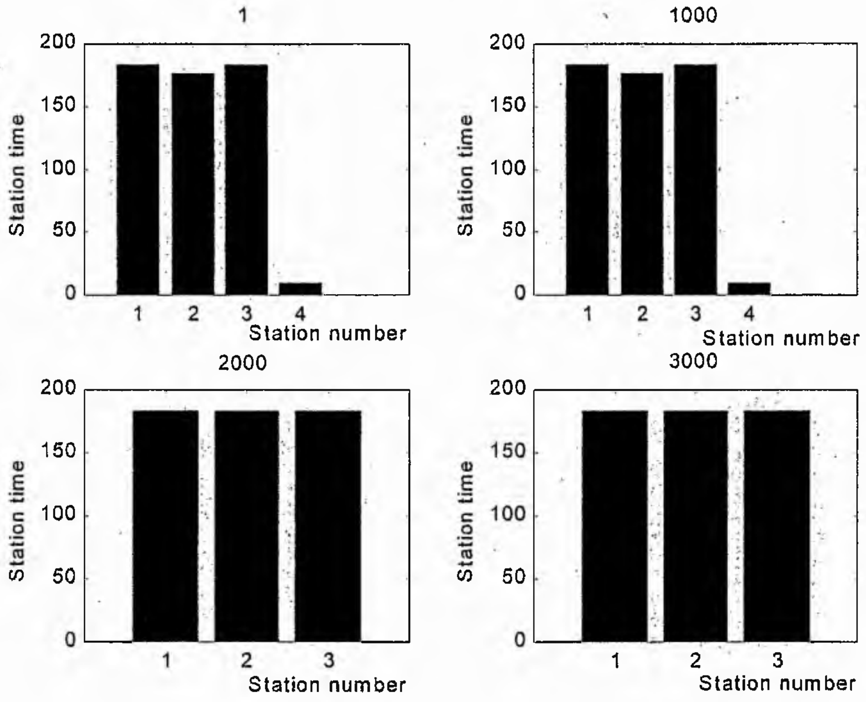


Figure 5.24. Station time distribution after 1,1000,2000, and 3000 generations for the problem KW_{45}^{184}

Figures 5.21-23 can be used to illustrate the forward loading mechanism of the proposed fitness function. Consider the problem KW_{45}^{69}, which is the hardest problem among those selected. As can be seen from figure 5.21, after one generation the first workstation was packed to its capacity (cycle time = 69), but after one thousand generations, workstations 1 and 2 were packed, and after 2000 generations the first five workstations were packed to the cycle time. Between 2000 and 3000 generations the number of workstations was reduced from nine to eight and all the workstations were packed to capacity yielding the optimal solution.

The same progressive filling mechanism can be seen in all the other three problems and they achieved optimal solutions between 2000-3000 generations. However, in KW_{45}^{184}, optimum solution was reached between 1000-2000 generations.

5.2.3 THE EFFECT OF PROBLEM COMPLEXITY ON FITNESS MODEL

Talbot et al (1986) claimed that the performance of heuristic line balancing methods were significantly effected by the cycle time. To find out the influence of problem complexity on the performances of the new model 24 cycle times consisting of both hard and easy problems were solved. Table 5.6 summarizes the number of workstations yielded after 3000 generations, where m^* denotes the theoretical minimum number of workstations for the particular set of test cases.

The proposed fitness model could not solve any of the hard problems optimally, however as problems become easy, in other words, when the cycle time is larger, more optimal solutions were achieved. For, example, five out of six cases of the group where $m^* = 5$, reached the optimal solutions and only one optimal solution was recorded in the second and third groups.

Figure 5.25 shows the average number of workstations above the optimum in each group ($m^* = 5$, 10, 15 and 20). The lowest of 0.16 were obtained by problems in group I ($m^* = 5$) where all the problems are considered as easy problems and their average cycle times are far away from the maximum task times.

m^*	Test problem					
	WA_{58}	LU_{89}	MU_{94}	AR_{111}	BA_{148}	SH_{297}
5	6^{+2}	5	5	5	5	5
10	11^{+1}	10	11^{+1}	11^{+1}	11^{+1}	11^{+1}
15	16^{+1}	15	16^{+1}	17^{+2}	16^{+1}	16^{+1}
20	22^{+2}	21^{+1}	22^{+2}	22^{+2}	21^{+1}	21^{+1}

Number of workstations above the optimum is specified following the '+' sign.

▨ Hard problems ░ Theoretical minimum number of workstations

Table 5.6. Number of workstations in the final solution

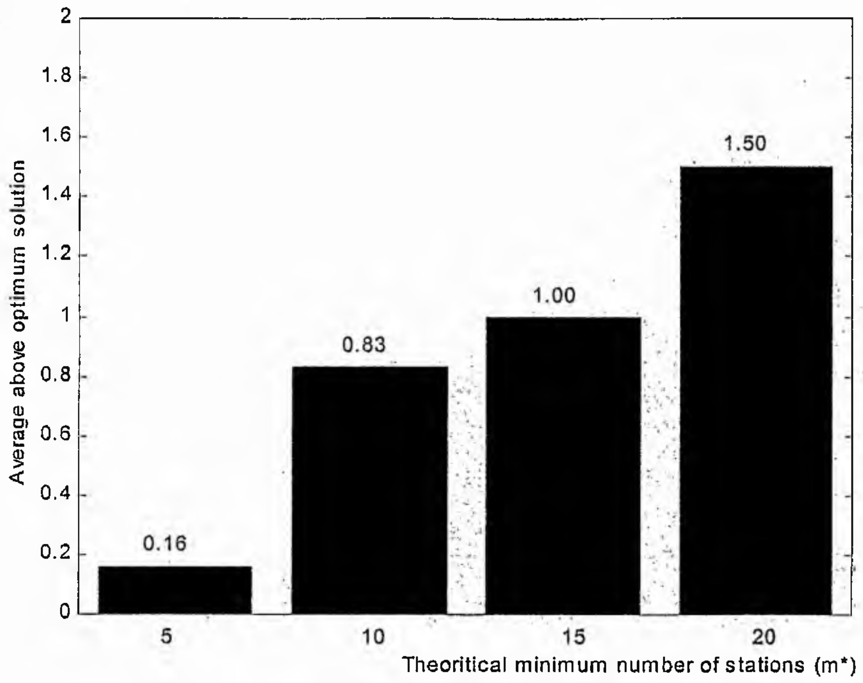


Figure 5.25. Average number of workstations above optimum

From the above findings it can be concluded that Talbot et al's claim is also valid for the proposed model and, generally, cycle times closer to the maximum task times are hard problems.

5.3 THE INFLUENCE OF POPULATION

5.3.1 POPULATION SIZE

The population size of the genetic algorithm must be selected to increase its efficiency and arrive at good solutions within a reasonable time. Table 5.7 shows the number of workstations obtained in the final solution for different sizes of populations and figure 5.26 displays the average number of workstations above the optimum. The populations of size 40 obtained the lowest average increase above the optimum. Populations of size 40 and 60 obtained optimal solutions in the problems MU_{94}^{176} and AR_{111}^{6269}, but the average increase above the optimum in populations of 60 was 0.17 above the population of 40. Larger populations (80, 100) showed considerable deviation from the best and this is mainly due to scattering the genetic search all over the solution domain. However, small populations will restrict the search around small neighborhoods resulting in poor quality solutions.

Population size	Test problem					
	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
30	26^{+1}	27^{+2}	25	26^{+1}	28^{+3}	27^{+2}
40	28^{+3}	26^{+1}	25	25	27^{+2}	26^{+1}
60	28^{+3}	26^{+1}	25	25	28^{+3}	26^{+1}
80	29^{+4}	26^{+1}	26^{+1}	26^{+1}	28^{+3}	27^{+2}
100	29^{+4}	26^{+1}	26^{+1}	27^{+2}	28^{+3}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.

$^{}_{}$ Theoretical minimum number of workstations

Table 5.7. Number of workstations the final solution for different population sizes

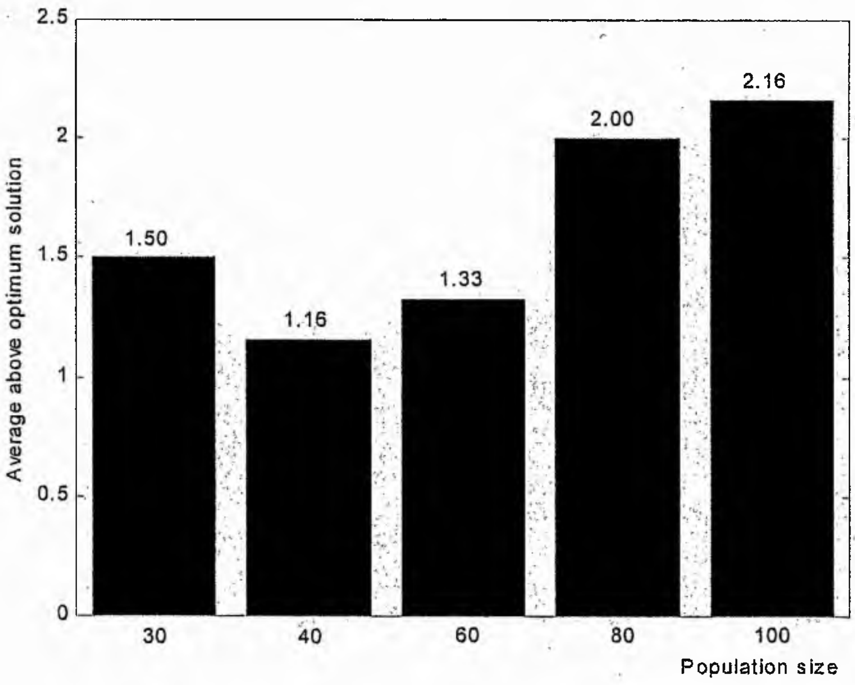


Figure 5.26. Average number of workstations above optimum for different population sizes

Figure 5.27 shows the average CPU time consumption for different population sizes. The CPU time grows rapidly as the size of population increases. Clearly, on the above findings, it can be concluded that the best population size for the proposed genetic algorithm model is 40 for the problem studied irrespective of the size of the problem.

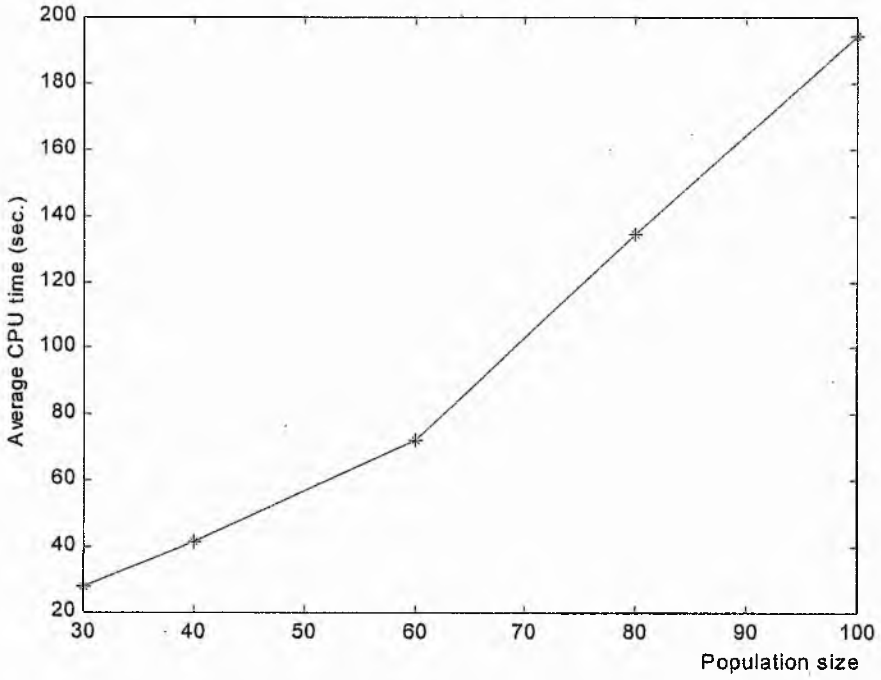


Figure 5.27. Average CPU time consumption for different population sizes

5.3.2 THE INITIAL POPULATION

Generally, a well-adapted (feasible) initial population guarantees faster convergence and reduces the computational time. Table 5.8 shows the number of workstations in the initial and final solutions: the initial solutions obtained by the heuristic techniques are shown within square brackets. The number of workstations above the optimum in the final solution is specified by the number following the '+' sign.

Out of the four heuristic techniques, the initial solutions generated by the Hoffman matrix procedure (Hoffmann, 1963) were closer to the optimum solutions and solutions generated by Rank Positional Weight (Helgeson and Birnie, 1961) showed the highest increase above the optimum. The bin packing population consisted of random permutations of numbers between 1 to *n*, where *n* is the number of elements in the problem and none of these chromosomes represented a feasible solution.

Solution technique	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
COMSOAL	[29] 27^{+2}	[27] 26^{+1}	[28] 26^{+1}	[28] 26^{+1}	[28] 27^{+2}	[27] 27^{+2}
Hoffman	[29] 27^{+2}	[26] 26^{+1}	[26] 26^{+1}	[27] 27^{+2}	[26] 26^{+1}	[26] 26^{+1}
Random	[32] 27^{+2}	[28] 26^{+1}	[28] 25	[30] 25	[29] 26^{+1}	[27] 26^{+1}
RPW	[31] 28^{+3}	[29] 27^{+2}	[30] 27^{+2}	[30] 27^{+2}	[29] 27^{+2}	[27] 27^{+2}
Bin Packing	No feasible solutions					

▨ Theoretical minimum number of workstations

Table 5.8. Number of workstations in the initial and final solutions

Figure 5.28 displays the average number of workstations above the optimum for each initial population. The lowest average was obtained by the population consisting of solutions generated by the random feasible task assignment technique (section 3.3), and the highest average was recorded with RPW technique (Rank Positional Weight). Moreover, in problems MU_{94}^{176} and AR_{111}^{6269}, random initial solutions finally converged into optimum solutions. Populations with the Hoffman solutions gave the next best, but none of the problems yielded optimum solutions. Even after 3000 generations, bin-packing solutions could not generate a single feasible solution.

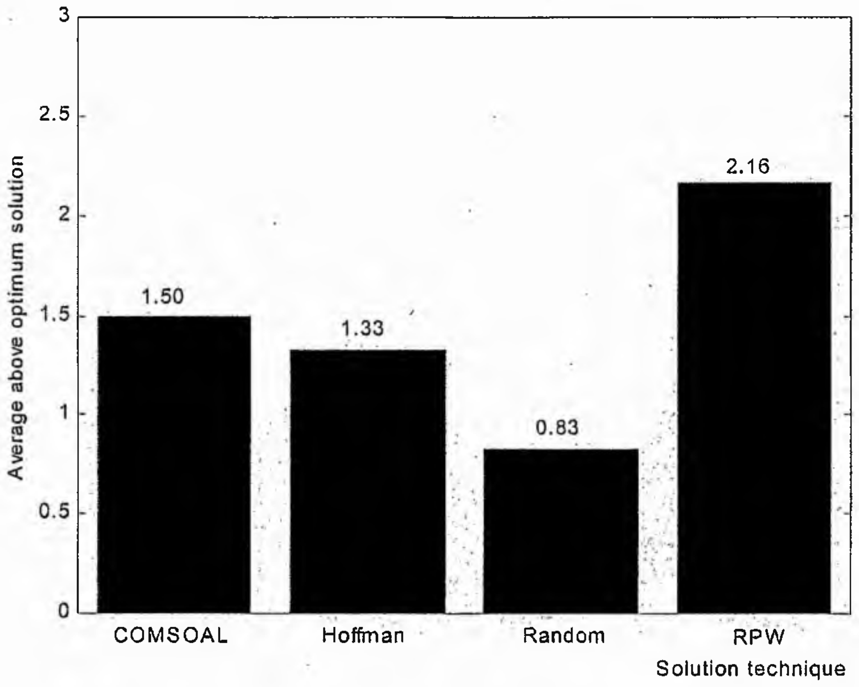


Figure 5.28. Average number of workstation above optimum for different initial populations

It can be seen from the findings that initial populations consisting of random feasible solutions is ideal for the new fitness model. Theses solutions are normally far from the optimal and give enough room for forward loading rather than closely packed solutions like the Hoffmann solutions.

As pointed out earlier, the best initial population is the one with solutions created by the random task assignment technique. The number of such solutions in the p opulation i ncreases t he c onvergence r ate and t he q uality o f t he final s olution. Table 5.9 shows the number of workstations in the final solution for initial populations consisting of a number of feasible solutions per initial population.

Populations with 20 random solutions achieved optimum solutions in problems MU_{94}^{176} and AR_{111}^{6269} and one with 10 solutions reached the optimum solution in the problem MU_{94}^{176}. Figure 5.29 shows the average number of workstations above the optimum for different level; of feasible solutions in the initial population. Once more, a population of 20 solutions showed the least average of 0.83 above the optimum and it was 0.50 below the second best, which consisting10 solutions.

Random solutions per population	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
1	28^{+3}	26^{+1}	26^{+1}	27^{+2}	28^{+3}	27^{+2}
5	28^{+3}	26^{+1}	26^{+1}	27^{+2}	28^{+3}	27^{+2}
10	28^{+3}	26^{+1}	25	26^{+1}	27^{+2}	26^{+1}
20	26^{+1}	26^{+1}	25	25	27^{+2}	26^{+1}
40	27^{+2}	26^{+1}	26^{+1}	26^{+1}	27^{+2}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.

Theoretical minimum number of workstations

Table 5.9. Number of workstations in the final solution

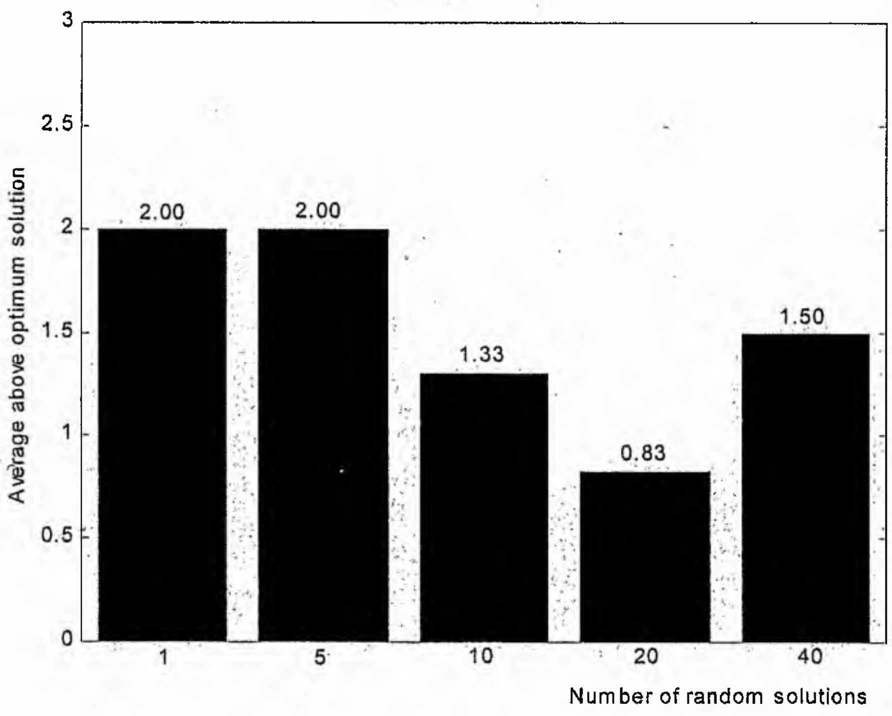


Figure 5.29. Average number of workstations above the optimum for different feasible solution levels

A small number of feasible solutions in the population generally slow down the convergence and restrict the searching process around a few neighborhoods. On the other hand, an initial population with a large number of solutions (well-seeded initial population) accomplishes a faster convergence by searching more solutions per population in different neighborhoods. Therefore, It can be concluded that, populations with number of random solutions is better for the genetic algorithm model. The number of solutions for better overall performance depends on the test problem and for the problems studied, populations with a number of random solutions equal to half of the total population provide the best overall performance.

5.4 COMPARISON OF SELECTION TECHNIQUES

The first genetic operation in the genetic algorithm is selection and a good selection scheme avoids both high selective pressure and premature convergence. Table 5.10 shows the number of workstation obtained in the final solution by six selection methods, and the average number of workstations above the optimum solutions is presented in figure 5.30.

Table 5.10. Number of workstations in the final solution for different selection schemes.

Selection Technique	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
Random	28^{+3}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}
Rank-Based (Modified)	26^{+1}	26^{+1}	25	25	26^{+1}	27^{+2}
Roulette Wheel	27^{+2}	26^{+1}	26^{+1}	26^{+1}	27^{+2}	28^{+3}
Good + Bad	27^{+2}	26^{+1}	25	27^{+2}	28^{+3}	29^{+4}
Tournament	28^{+3}	26^{+1}	26^{+1}	27^{+2}	27^{+2}	27^{+2}
Rank-Based	27^{+2}	26^{+1}	26^{+1}	26^{+1}	27^{+2}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

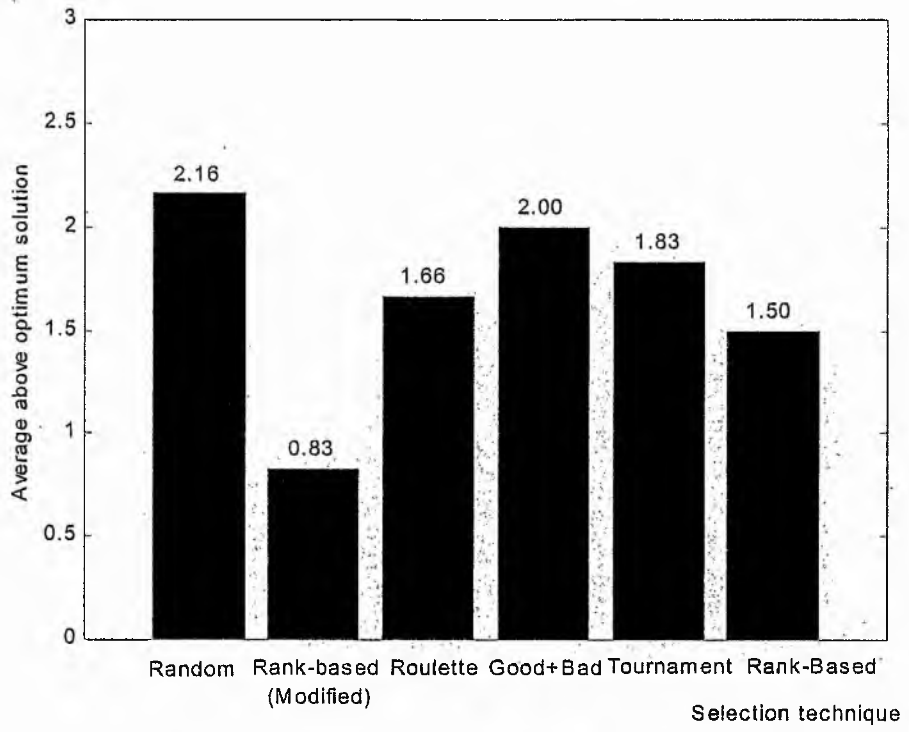


Figure 5.30. Average numbers of workstations above the optimum

The modified rank-based selection scheme obtained optimum solutions in problems MU_{94}^{176} and AR_{111}^{6269}, showing the overall best of 0.83 average increases above the optimum. The second best was achieved by the classic rank-based technique and, it was a 0.67 worse than modified selection scheme.

The classic rank-based selection method showed the best performance among the published selection techniques, and agrees with Whitley's (1989) findings (the rank-based selection technique achieves faster convergence and produces better solutions than other published methods). The roulette wheel selection, the most used technique in genetic algorithms (Appendix B) reported a 0.16 average above the classic rank based and a 0.83 average increase above the modified selection technique respectively. The random selection technique showed the worst overall performance (2.16 above the average), an increase of 1.33 above the best average.

5.4.1 THE EFFECT OF SELECTION POOL SIZE

In the modified rank-based selection scheme, the chromosomes are selected from the selection pool for mating. The performances obtained by different sizes of pool are presented in table 5.11 and the average number of workstations above the optimum is shown in figure 5.31.

Table 5.11. Number of workstation in the final solution

Selection pool size	Test problem					
	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
10	29^{+4}	27^{+2}	27^{+2}	27^{+2}	28^{+3}	27^{+2}
20	29^{+4}	26^{+1}	26^{+1}	27^{+2}	27^{+2}	27^{+2}
30	27^{+2}	26^{+1}	25	25	26^{+1}	26^{+1}
40	28^{+3}	27^{+2}	26^{+1}	28^{+3}	26^{+1}	26^{+1}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

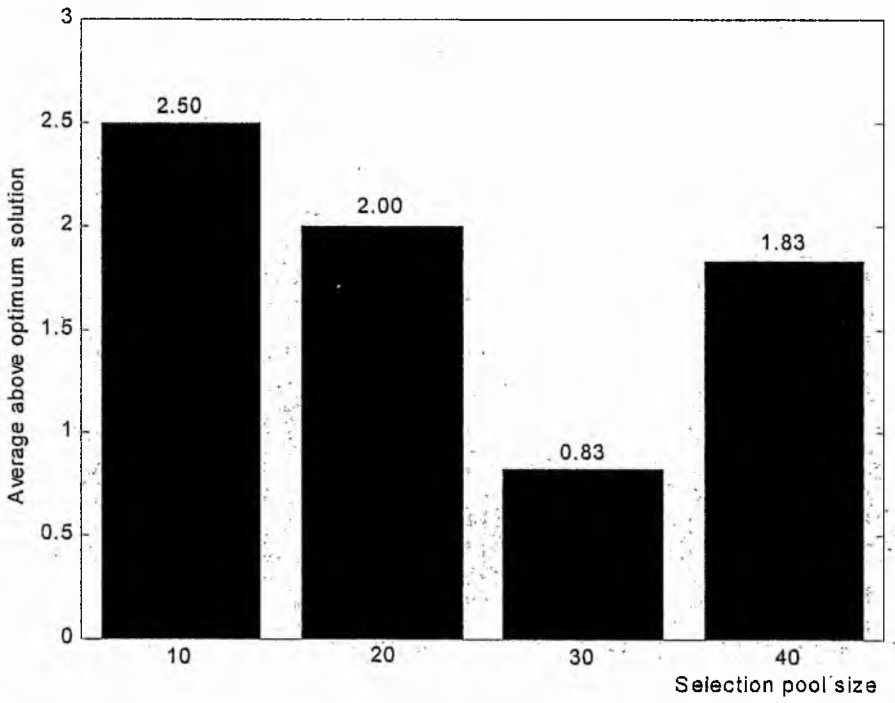


Figure 5.31. Average number of workstations above the optimum

A selection pool of size 30 achieved the best overall performances, an average of 0.83 above the optimum and reaching optimality in problems MU_{94}^{176} and AR_{111}^{6269}. The second best was achieved by size of 40 and small pool sizes showed relatively poor quality solutions displaying higher averages above the optimum. Generally, a small size of pool consists of only a few best solutions and allows limited mating. However, a large selection pool contains both good and bad solutions and normally generates low quality solutions after mating.

5.4.2 SELECTIVE PRESSURE

As Whitely (1989) pointed out, high selective pressure in the rank-based selection technique increased the number of identical solutions and decreases the population diversity. The modified rank-based selection scheme addressed this issue very effectively and figures 5.32 to 5.37 display the selective pressure plot as the number of generations increase for both rank based and modified rank based techniques for the six selected problems.

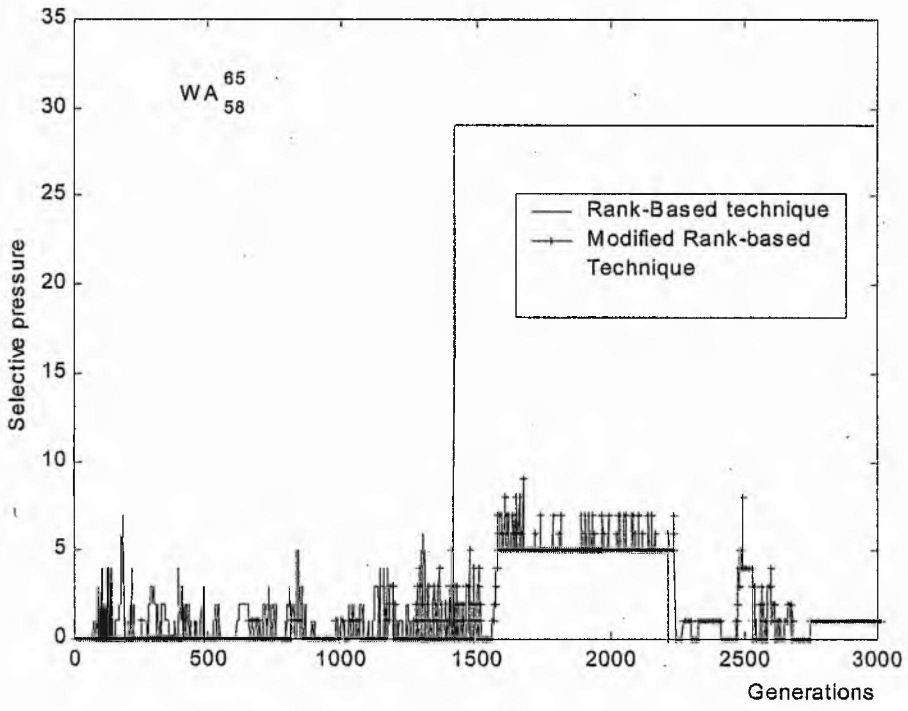


Figure 5.32. Selective pressure variation for test problem WA_{58}^{65}

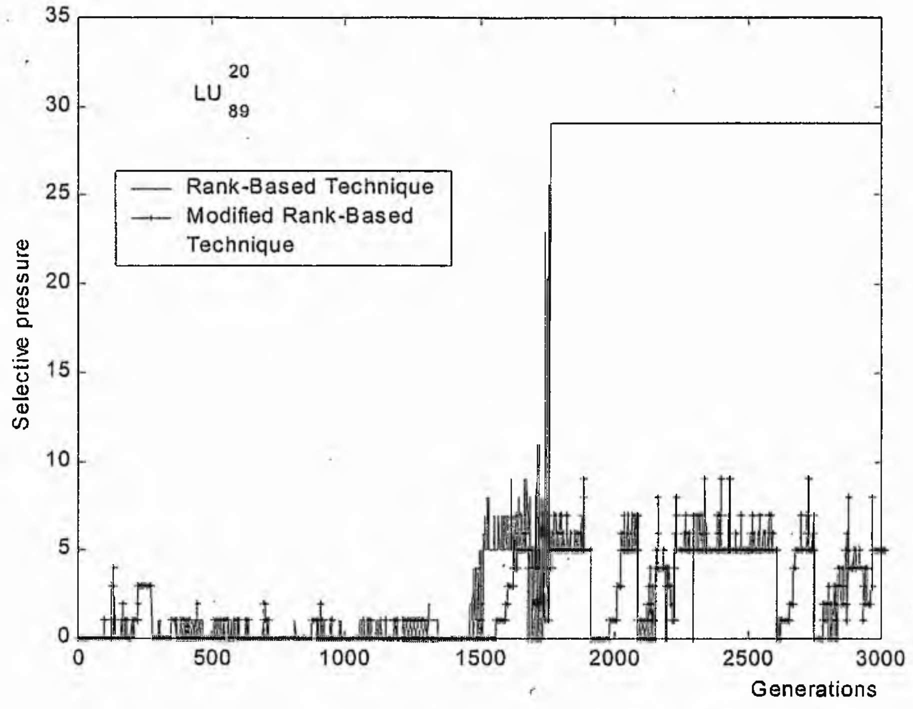


Figure 5.33. Selective pressure plot for test problem LU_{89}^{20}

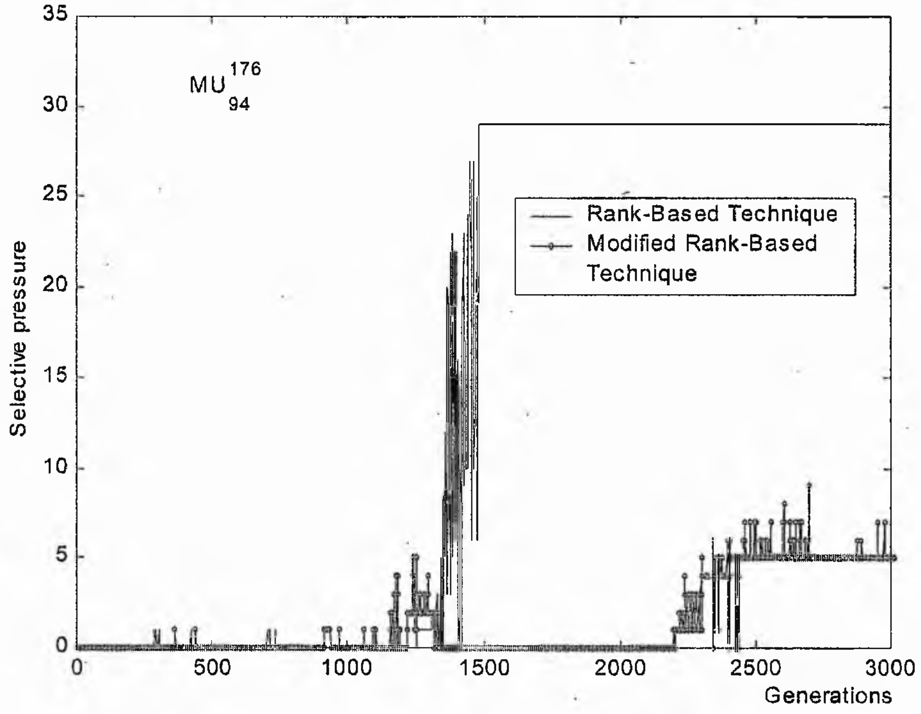


Figure 5.34. Selective pressure plot for test problem MU_{94}^{176}

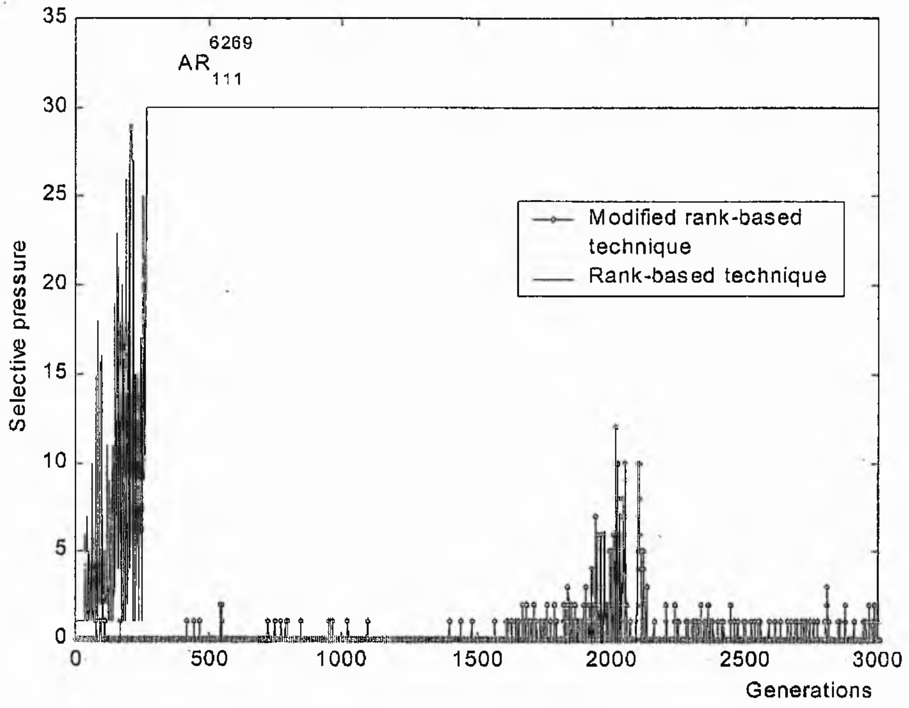


Figure 5.35. Selective pressure plot for test problem AR_{111}^{6269}

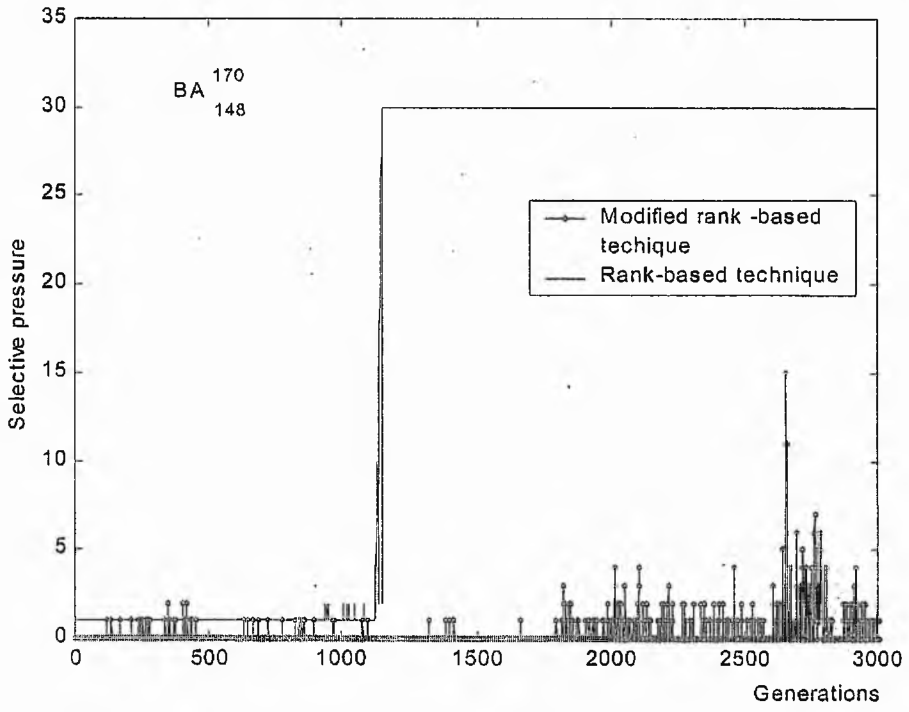


Figure 5.36. Selective pressure plot for test problem BA_{148}^{170}

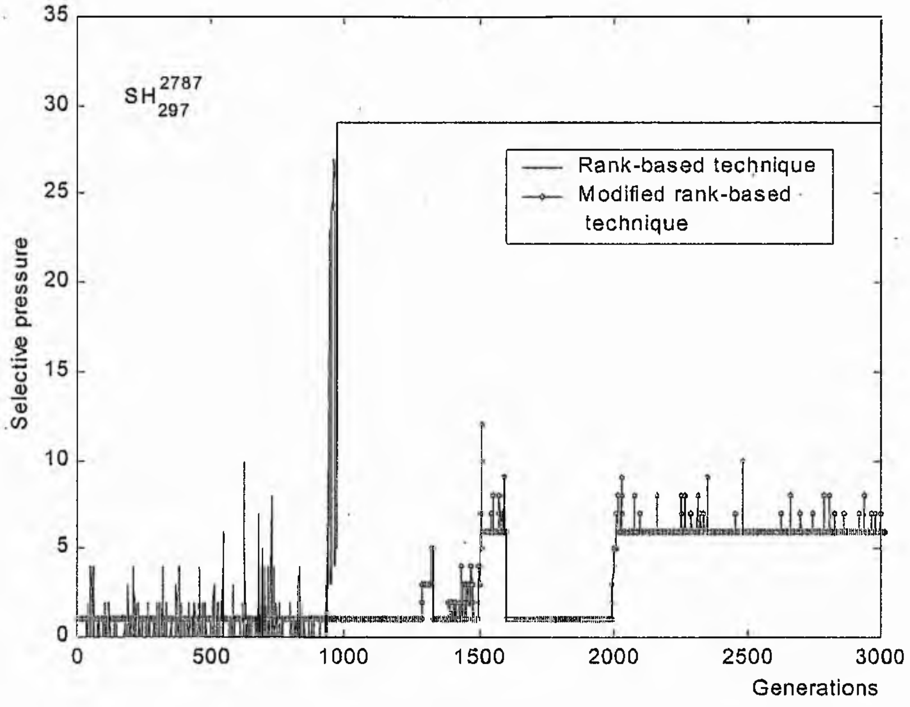


Figure 5.37. Selective pressure plot for test problem SH_{297}^{2787}

As can be seen from the figures, except AR_{111}^{6269}, in all the other test problems, the classic rank-based selection technique showed high selective pressure between 1000-2000 generations (more than 95% of the population containing the same chromosome). Problem AR_{111}^{6269} showed an early rise in selective pressure and thereafter it was maintained throughout the genetic process.

However, the modified rank-based selection scheme showed low selective pressure (less than or equal to 15 identical solutions) throughout the genetic process. In all test problems, within the first 1000 generations, it was maintained at a low value (<10). After 2000 generations, it displayed slightly higher selective pressure, but fluctuating peaks in the plot indicated that after each high selective pressure, there was a significant drop in selective pressure. Therefore, It can be concluded that, the new selection technique has eliminated the high selective pressure issue in the ordinary rank-based technique, and could be applied to all test problems irrespective of their size and complexity.

5.5 COMPARISON OF CROSSOVER TECHNIQUES AND ITS CONTROL PARAMETERS

5.5.1 CROSSOVER TECHNIQUE COMPARISON

In the proposed genetic algorithm model, about 80% ($p_c = 0.8$) of the offspring in the new population are created by a crossover operation. Table 5.12 shows the final number of workstations obtained by eight crossover techniques including two proposed techniques and six published methods. The average increase of workstations above the optimum is illustrated in figure 5.38.

Table 5.12. Number of workstations in the final solution with different crossover techniques

Crossover Technique	Test problem					
	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
VBMCP	27^{+2}	26^{+1}	25	25	27^{+2}	26^{+1}
FBMCP	29^{+4}	26^{+1}	26^{+1}	26^{+1}	27^{+2}	27^{+2}
SPC	28^{+3}	26^{+1}	25	26^{+1}	28^{+3}	27^{+2}
TPC	28^{+3}	26^{+1}	25	26^{+1}	28^{+3}	27^{+2}
ORD	28^{+3}	29^{+4}	28^{+3}	27^{+2}	28^{+3}	28^{+3}
POS	27^{+2}	26^{+1}	25	26^{1}	27^{+2}	27^{+2}
FRG	29^{+4}	27^{+2}	27^{+2}	27^{+2}	27^{+2}	27^{+2}
UNI	27^{+2}	26^{+1}	25	26^{+1}	27^{+2}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

VBMCP = Variable Boundary Moving Crossover Point
FBMCP = Fixed Boundary Moving Crossover Point
SPC = Single Point Crossover
TPC = Two Point Crossover
ORD = Order Crossover
POS = Position Based Crossover
FRG = FRaGment Reordering Crossover
UNI = Uniform Crossover

The proposed VBMCP method showed the lowest average increase above optimum (1.00) and both POS and UNI techniques displayed the second best, an average 0.33 above the VBMCP. Problems MU_{94}^{176} and AR_{111}^{6269} reached the optimum solutions with the new crossover technique and all the other problems performed with the least number of workstations above their optimum solutions.

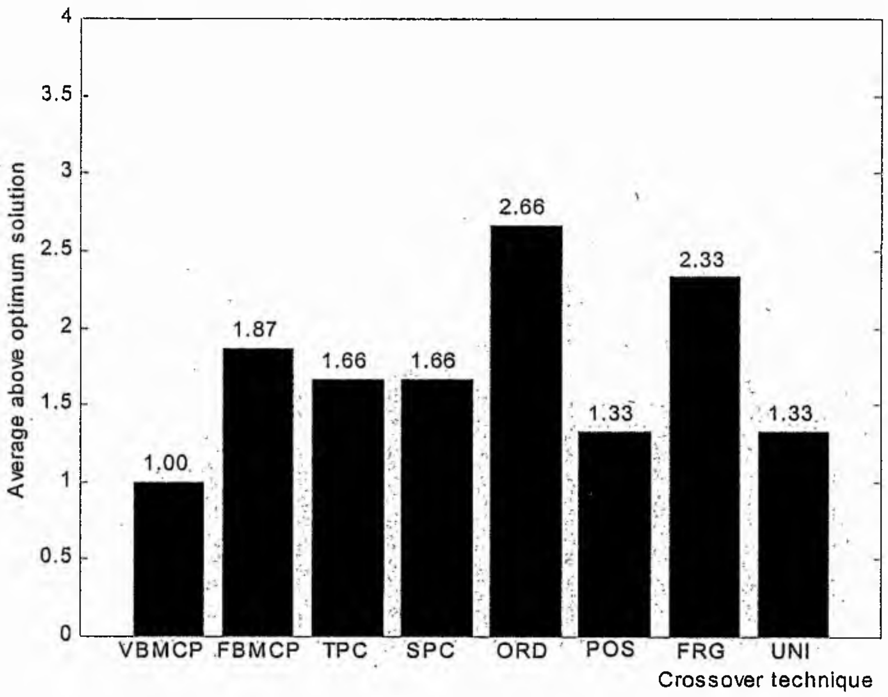


Figure 5.38. Average number of workstations above the optimum

The classic two point crossover (TPC) and single point crossover (SPC) also achieved optimum solutions in problem MU_{94}^{176}, and showed the same average above the optimum (1.66). Rubinovitz (1995) developed the Fragment Reordering Crossover (FRG) method especially for the assembly line balancing problem and claimed that it was the best among the published techniques. However, It seems that this technique does not perform well with the new front-loading fitness function.

5.5.2 THE EFFECTIVE CROSSOVER SPAN RATIO

In the variable boundary moving crossover point technique, the crossover span where the crossover point is selected is the key parameter and controls the overall performances. The size of the crossover span is determined by the Crossover Span Ratio (CSR) and is the maximum number of elements (c_s) that could pack in to a bin of capacity equals to the cycle time and is generally a constant for a particular cycle time. Table 5.13 shows the number of workstations obtained in the final solution for five crossover spans and figure 5.39 displays the average increase in number of workstations above the optimum.

Crossover spans with size equal to half of c_s (CSR=0.5) showed the best performance obtaining the least average above the optimum (1.16). The second best was achieved by a span with CSR = 1.0 and both of these reached the optimum solutions in problem MU_{94}^{176} and

$$\text{Crossover Span Ratio (CSR)} = \frac{\text{Crossover span size}}{c_s}$$

Table 5.13. Number of workstations in the final solution

	Test problem					
Crossover span ratio	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
	7	14	17	41	23	45
0.5	27^{+2}	26^{+1}	25	26^{+1}	26^{+1}	27^{+2}
1.0	27^{+2}	26^{+1}	25	27^{+2}	27^{+2}	27^{+2}
1.5	27^{+2}	26^{+1}	26^{+1}	28^{+3}	26^{+1}	28^{+3}
2.0	28^{+3}	26^{+1}	26^{+1}	28^{+3}	26^{+1}	28^{+3}
2.5	28^{+3}	26^{+1}	26^{+1}	28^{+3}	27^{+2}	28^{+3}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

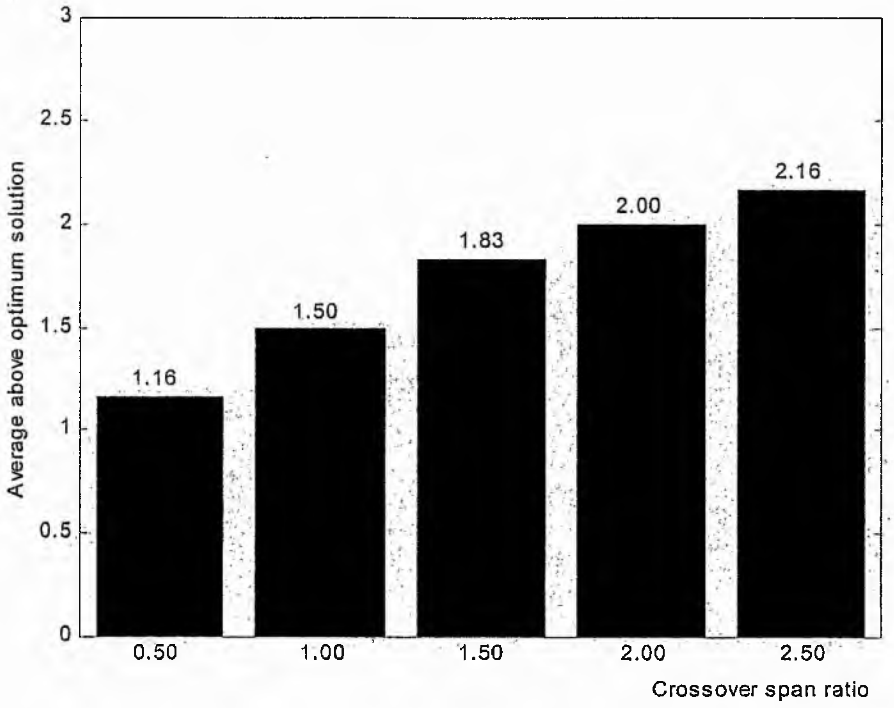


Figure 5.39. Average number of workstations above optimum

Increasing crossover span size resulted in decreasing performance. Large spans allow the recombination process to propagate both good and bad attributes (feasible and infeasible links) of parents to offspring resulting in inferior quality solutions. Small crossover spans allow largely good parts of the solutions to transfer in to offspring increasing its solution quality. However, very small spans would not transfer enough characteristics to make good solutions. Therefore, it can be concluded that for the problems studied the best crossover span ratio for overall best performance is spans with CSR = 0.5.

5.6 THE EFFECT OF REPAIR TECHNIQUE ON FEASIBLE SOLUTIONS.

Distortion of feasibility in chromosomes after a crossover operation is inevitable. Generally, a repair technique must be used to mend these chromosomes in order to restore its feasibility. Two new repair techniques were developed in this research and, the number of workstations obtained in the final solution by each repair technique is shown in table 5.14. The average number of workstations above the optimum is illustrated in figure 5.40.

Table 5.14. Number of workstations in the final solution

Repair technique	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
REP1	27^{+2}	26^{+1}	25^{0}	25^{0}	26^{+1}	27^{+2}
REP2	27^{+2}	26^{+1}	25^{0}	27^{+2}	27^{+2}	27^{+2}
REP3	26^{+1}	26^{+1}	25^{0}	25^{0}	26^{+1}	26^{+1}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

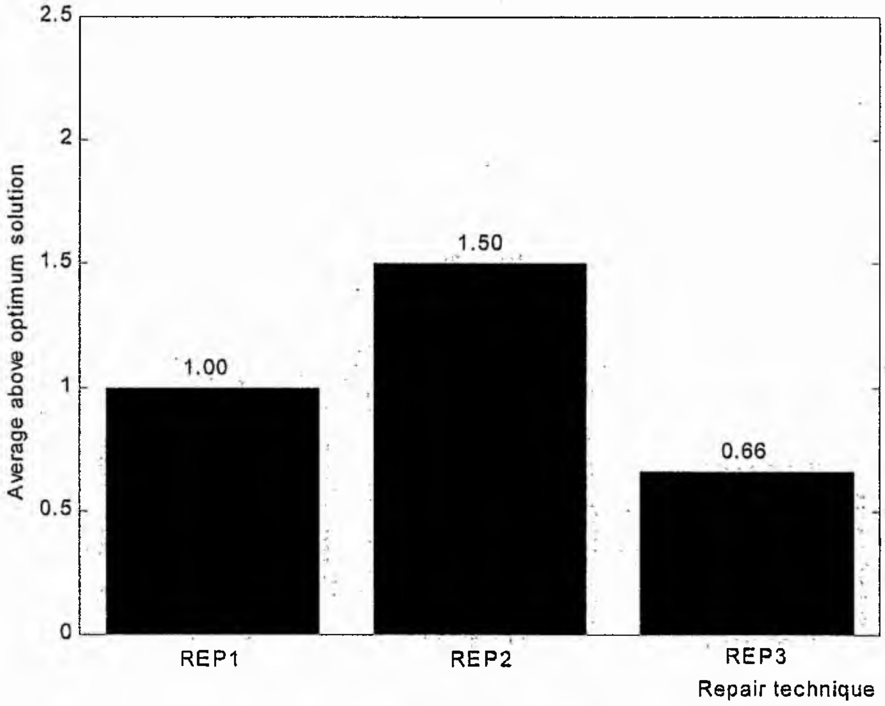


Figure 5.40. Average number of workstations above optimum

The repair technique that re-assigning duplicating elements taking into account its rank positional weight (REP3) (the highest rank available element first, section 3.3.4.3) showed the least average above optimum and it solved both MU_{94}^{176} and AR_{111}^{6269} problems optimally. The second best was achieved by the random based repair technique (REP1, section 3.3.4.1) and the worst performances were displayed by the order based repair technique (REP3, section 3.3.4.2). Table 5.15 illustrates the average number of feasible solutions created per generation by the three repair techniques.

Table 5.15. Average number of feasible solutions per generation

Repair technique	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
REP1	16.00	13.46	26.61	25.20	29.65	38.02
REP2	21.50	15.76	30.80	36.10	38.62	38.23
REP3	20.38	16.64	28.93	27.00	28.90	38.20

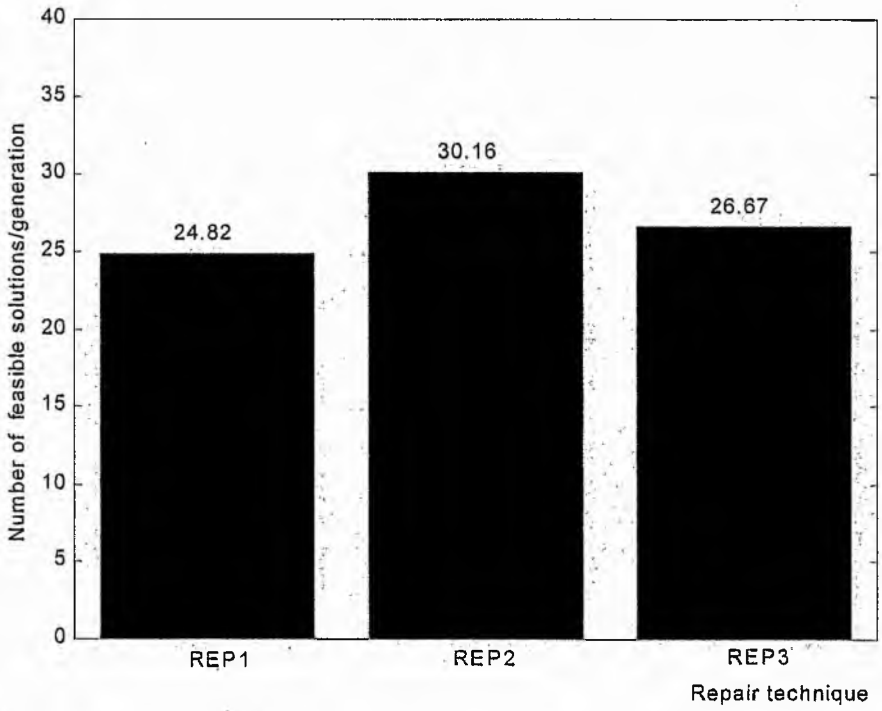


Figure 5.41. Number of feasible solutions per generation for three repair techniques

Figure 5.41 displays the overall average number of feasible solutions created per generation by each repair technique. The order based repair technique showed the highest average and second best was generated by the rank positional based repair technique. The worst performances were displayed by the random based repair technique and it was 5.34 solutions per generation less than the best.

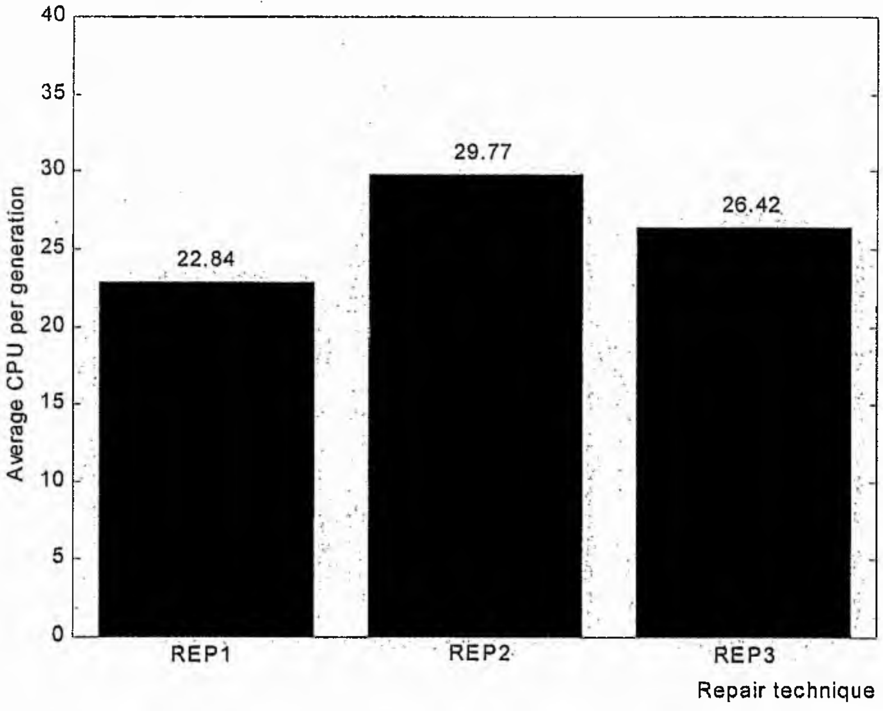


Figure 5.42. Average CPU time per generation for the repair techniques

The average CPU consumption per generation by the three repair techniques is shown in figure 5.42. It seems that, although the order based repair technique (REP2) generate more feasible solutions, it consumes more computational time. The REP3 (rank-based) consumes less average CPU time than REP2 and showed the lowest average above the optimum.

5.7 COMPARISON OF MUTATION TECHNIQUES AND ITS CONTROL PARAMETERS

5.7.1 COMPARISON OF MUTATION TECHNIQUES

Mutation is the other genetic operation that generates new chromosomes away from the current neighborhood allowing the algorithm to explore a wider region. Five new mutation techniques were considered in this research and they are compared with the classic mutation. Table 5.16 shows the number of workstations obtained after each mutation technique and figure 5.43 indicates the average number of workstations above the optimum.

Table 5.16. Number of workstation in the final solution

Mutation Technique	Test problem					
	WA^{65}_{58}	LU^{20}_{89}	MU^{176}_{94}	AR^{6269}_{111}	BA^{170}_{148}	SH^{2787}_{297}
FBRM	27^{+2}	26^{+1}	25	26^{+1}	27^{+2}	28^{+3}
VBAM	27^{+2}	26^{+1}	25	25	26^{+1}	27^{+2}
VBRM	28^{+3}	28^{+3}	26^{+1}	28^{+3}	27^{+2}	28^{+3}
FBAM	28^{+3}	26^{+1}	26^{+1}	27^{+2}	26^{+1}	28^{+3}
RAM	28^{+3}	28^{+3}	28^{+3}	28^{+3}	27^{+2}	28^{+3}
ADM	27^{+2}	27^{+2}	27^{+2}	28^{+3}	27^{+2}	27^{+2}

Number of workstations above the optimum is specified following the '+' sign.
Theoretical minimum number of workstations

FBRM = Fixed Boundary Random Mutation
FBAM = Fixed Boundary Adjacent Mutation
VBRM =Variable Boundary Random Mutation
VBAM = Variable Boundary Adjacent Mutation
RAM = Random Mutation
ADM = Adjacent Mutation

The variable boundary adjacent mutation (VBAM) showed the lowest average above the optimum and reached optimum solution in problems MU_{94}^{176} and AR_{111}^{6269}. The second best performance was obtained by the fixed boundary random mutation technique (FBRM). The classic random mutation displayed the highest average number of workstations above the optimum. Therefore, it can be concluded that the VBAM technique seems to be the best for the problems studied.

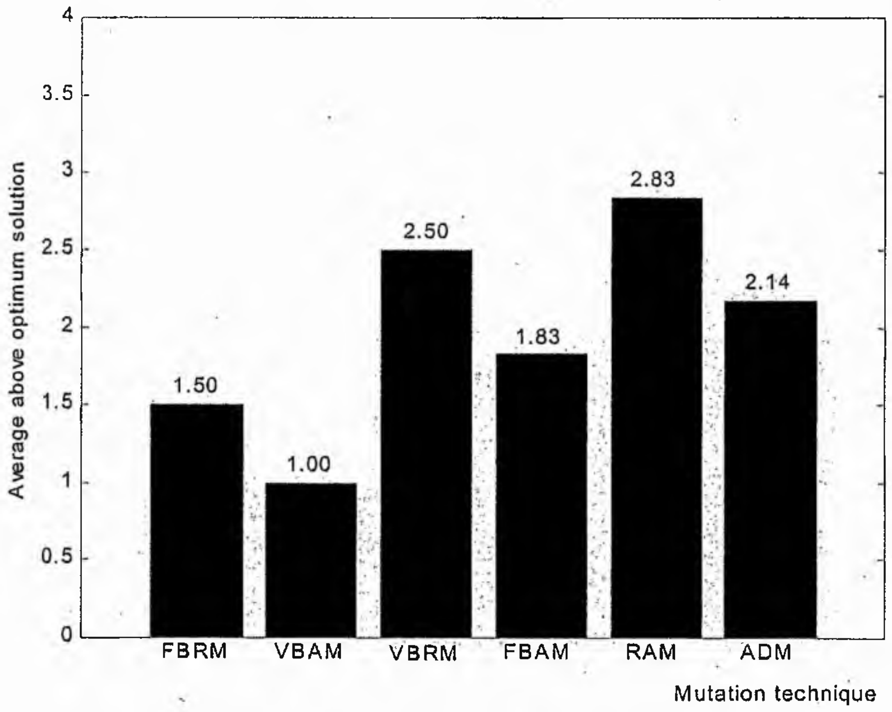


Figure 5.43. Average number of workstations above optimum

5.8 THE INFLUENCE OF ELITISM ON THE PERFORMANCE

Propagation of best chromosomes from current population to the next population is crucial for convergence. The number of elite chromosomes copying to the next generation is a vital factor for faster convergence but, copying more chromosomes may decrease population diversity. Table 5.17 shows the number of workstations in the final solution for four elitism levels and figure 5.44 illustrates the average increase of workstations above the optimum.

Propagation of 6 and 10 chromosomes showed the best performances and, the first level of elitism (6 chromosomes) yielded optimum solutions in three problems (LU_{89}^{20}, MU_{94}^{176} and AR_{111}^{6269}). Increasing the number of elite chromosomes decreases the overall performance and, 6-10 elite chromosomes that is 3-5% of the selection pool, seems to be a good choice for the test cases studied.

Table 5.17. Number of workstations in the final solution

Number of elite strings	Test problem					
	WA_{58}^{65}	LU_{89}^{20}	MU_{94}^{176}	AR_{111}^{6269}	BA_{148}^{170}	SH_{297}^{2787}
6	27^{+2}	25	25	25	27^{+2}	27^{+2}
10	27^{+2}	26^{+1}	25	25	26^{+1}	27^{+2}
20	27^{+2}	26^{+1}	25	26^{+1}	27^{+2}	27^{+2}
30	28^{+3}	27^{+2}	26^{+1}	26^{+1}	27^{+2}	28^{+3}

Number of workstations above the optimum is specified following the '+' sign.
▨ Theoretical minimum number of workstations

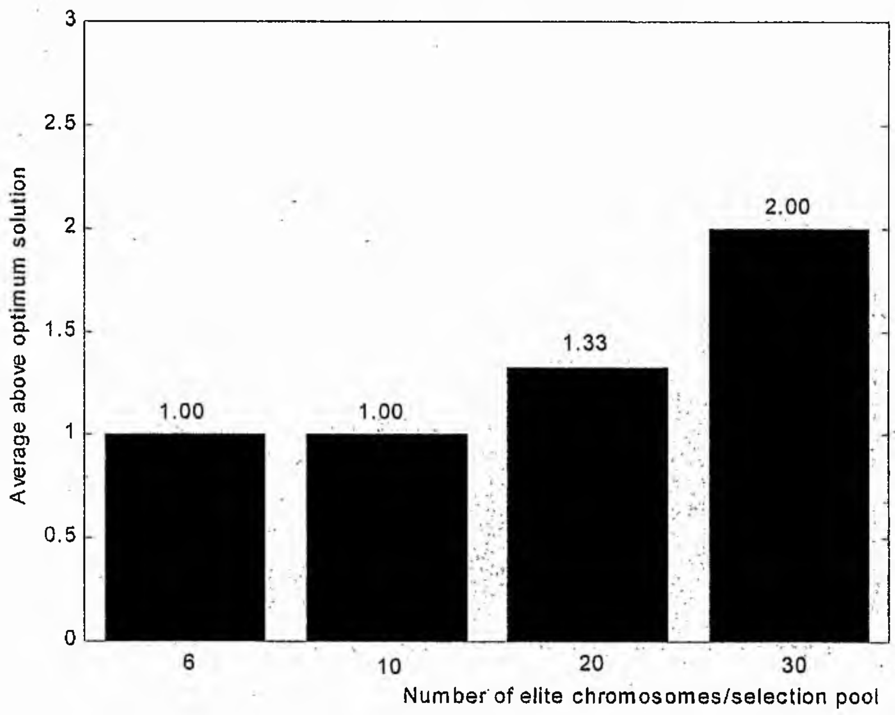


Figure 5.44. Average number of workstations above optimum

5.9 COMPARISON WITH OTHER HEURISTIC TECHNIQUES

In order to draw general conclusions about the new genetic algorithm model, it was compared with four heuristic line balancing procedures and the number of workstations obtained in the final solution for fifty-four test problems are shown in table 5.18. The first column of the table represents the test problem reference and eighth and ninth columns display the theoretical number of workstations and the best performance that have been achieved by any technique. These results are further categorized into four groups depending on the theoretical minimum number of workstations and they are shown in table 5.19. The results are summarized and presented in figures 5.45 and 5.46.

Table 5.18. Final solutions obtained by heuristic techniques

Reference	Problem ID	Hoffman	COMSOAL	Modified Hoffman	Genetic Algorithm	RPW	m^*	m^b
Scholl (1979)	SH_{297}^{2488}	29	30	29	30	30	28	28
	SH_{297}^{2680}	27	29	27	27	29	26	26
	SH_{297}^{2787}	26	27	26	26	27	25	25
	SH_{297}^{7000}	10	11	10	10	11	10	10
Bartholdi (1993)	BA_{148}^{152}	29	33	29	30	33	28	28
	BA_{148}^{152}	29	31	27	29	31	27	27
	BA_{148}^{163}	28	30	27	27	30	26	26
	BA_{148}^{170}	26	29	26	27	29	25	25
Arcus (1963b)	AR_{111}^{6269}	27	29	26	25	30	24	25
Mukherjee and Basu (1964)	MU_{94}^{176}	26	31	25	25	30	24	25
	MU_{94}^{222}	21	23	20	20	22	19	20
	MU_{94}^{281}	16	17	16	16	18	15	16
	MU_{94}^{301}	15	16	15	15	17	14	15
	MU_{94}^{351}	13	13	13	13	13	12	13

RPW –Rank Positional Weight Technique, m^* is the theoretical minimum number of workstations and m^b is the best performance has been achieved by any technique.

▨ Theoretical minimum number of workstations

Table 5.18. Final solutions obtained by heuristic techniques (cont..)

Reference	Problem ID	Hoffman	COMSOAL	Modified Hoffman	Genetic Algorithm	RPW	m^*	m^b
Lutz (1974)	LU_{89}^{18}	30	31	30	29	32	27	27
	LU_{89}^{19}	28	30	27	27	30	26	27
	LU_{89}^{20}	27	28	26	25	29	25	25
	LU_{89}^{21}	25	26	25	24	29	24	24
Arcus (1963b)	AR_{83}^{3985}	21	23	20	20	23	20	20
	AR_{83}^{5408}	15	16	15	14	16	14	14
	AR_{83}^{7571}	11	11	11	10	12	10	10
Tonge (1961)	TO_{70}^{251}	16	17	15	15	18	14	15
	TO_{70}^{320}	12	13	12	12	13	11	12
Warnecke(1971)	WA_{58}^{111}	14	16	14	14	16	14	14
	WA_{58}^{104}	16	17	16	15	19	15	15
	WA_{58}^{97}	17	20	17	16	21	16	16
	WA_{58}^{92}	19	20	18	17	22	17	17
	WA_{58}^{86}	21	23	20	19	24	18	19
Kilbridge and Wester (1962)	KW_{45}^{69}	9	10	8	8	9	8	8
	KW_{45}^{92}	7	7	6	6	7	6	6
	KW_{45}^{138}	4	5	4	4	5	4	4
	KW_{45}^{184}	3	4	3	3	4	3	3
Gunther et al (1983)	GU_{35}^{41}	16	17	15	14	17	12	14
	GU_{35}^{49}	11	12	11	11	12	10	11
	GU_{35}^{54}	10	11	9	9	11	9	9
	GU_{35}^{81}	7	7	7	7	7	6	7
Sawyer (1970)	SA_{30}^{27}	14	17	14	14	19	14	14
	SA_{30}^{30}	12	14	12	12	14	11	12
	SA_{30}^{41}	9	9	8	8	10	8	8
Buxey(1974)	BU_{29}^{27}	13	16	13	13	14	12	13
	BU_{29}^{33}	11	12	11	11	13	10	11
	BU_{29}^{36}	10	11	10	10	12	9	10
	BU_{29}^{41}	9	9	8	8	10	8	8
	BU_{29}^{47}	8	8	8	8	9	7	7

RPW –Rank Positional Weight Technique, m^* is the theoretical minimum number of workstations and m^b is the best performance has been achieved by any technique. ▨ Theoretical minimum number of workstations

Table 5.18. Final solutions obtained by heuristic techniques (cont.)

Reference	Problem ID	Hoffman	COMSOAL	Modified Hoffman	Genetic Algorithm	RPW	m^*	m^b
Rosenberg and Ziegler (1992)	RO^{14}_{25}	11	12	10	10	12	9	10
	RO^{16}_{25}	9	9	9	8	9	8	8
	RO^{21}_{25}	6	7	6	6	7	6	6
	RO^{32}_{25}	4	4	4	4	4	4	4
Mitchell(1957)	MI^{15}_{21}	9	9	9	8	8	7	8
	MI^{26}_{21}	5	5	5	5	5	5	5
Mansoor (1964)	MO^{48}_{11}	4	4	4	4	5	4	4
	MO^{62}_{11}	4	4	4	4	4	3	4
	MO^{94}_{11}	2	2	2	2	2	2	2
Bowman (1960)	BO^{20}_{8}	5	6	6	5	6	4	5

RPW –Rank Positional Weight Technique, m^* is the theoretical minimum number of workstations and m^b is the best performance has been achieved by any technique.

▨ Theoretical minimum number of workstations

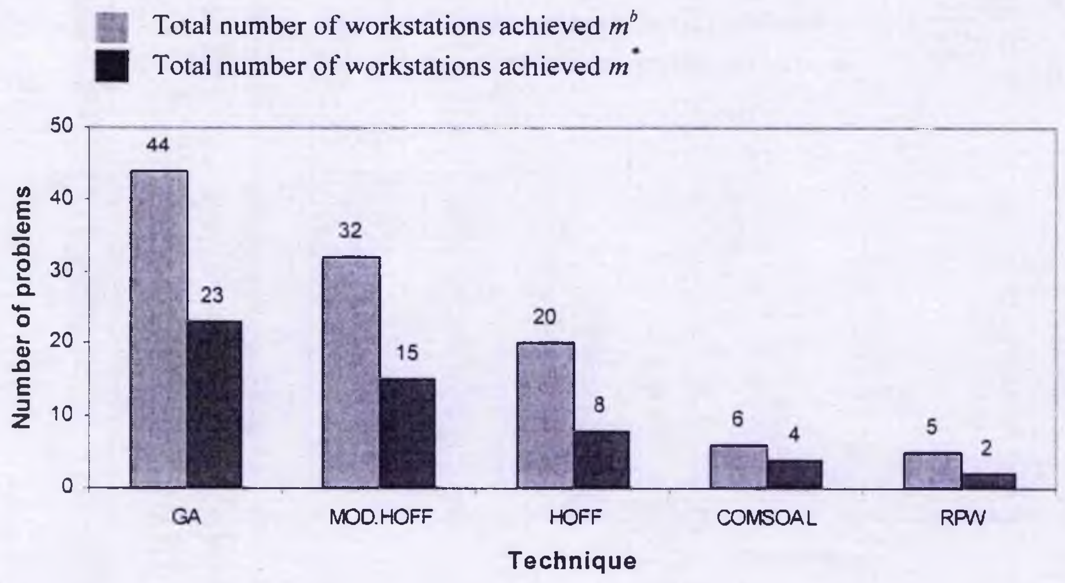


Figure 5.45. Number of problems achieved m^b and m^* by heuristic techniques

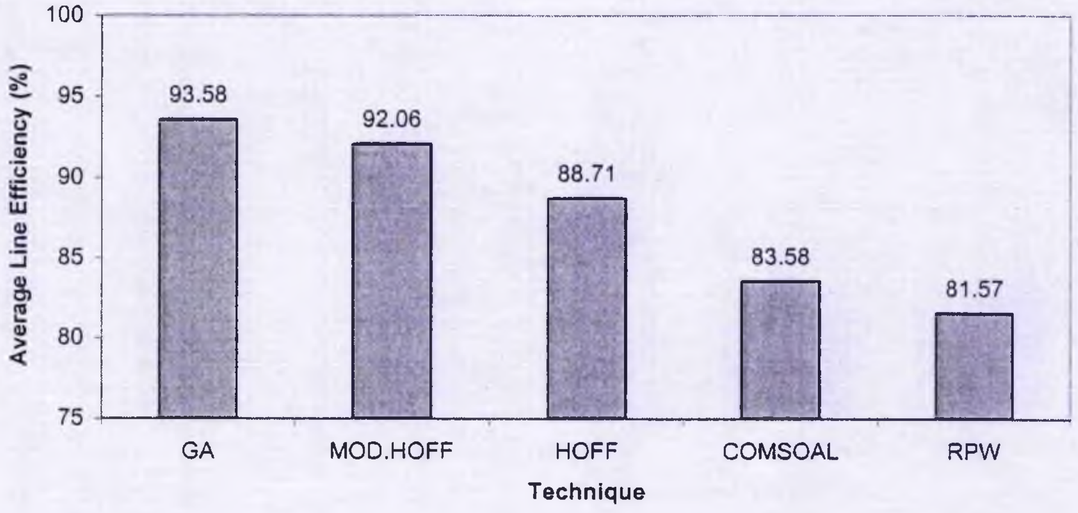


Figure 5.46. Average line efficiency obtained by line balancing techniques

Table 5.19. Number of problems achieved m^b by the heuristic techniques

Theoretically minimum number of workstations	Hoffman	COMSOAL	Modified Hoffman	Genetic Algorithm	Rank positional weight
$m^* \leq 10$	10	5	16	21	4
$10 < m^* \leq 20$	10	1	12	17	1
$20 < m^* \leq 25$	0	0	2	5	0
$m^* > 25$	0	0	2	1	0

For forty-four test problems (81%), the new algorithm achieved the best performance that has been achieved by any heuristic technique and it is approximately twice that achieved by the original Hoffman precedence technique. In none of the instances did the Hoffman technique outperform the genetic model! In twenty-three instances (42%), the proposed technique achieved the theoretical minimum number of workstations.

The genetic algorithm achieved the highest average line efficiency (93.58%) and is 4.87% above the Hoffman technique and it balanced the four test cases (69, 92, 138 and 184 cycle times) of the Kibridge and Wester problem producing zero idle time optimal solutions, confirming its ability to achieve excellent solutions. The second best is obtained by the modified Hoffman heuristic and is 1.52% less than the genetic algorithm.

The model indicated excellent performance in test cases where $m^* < 10$ and for larger m^* it showed good performances by obtaining near optimum solutions (table 5.19) either better or the same as that obtained by the Hoffmann p rocedure. Therefore, It can be finally concluded that the new genetic model seems better than the Hoffman precedence procedure irrespective of problem size and its complexity, accepting the main hypothesis of this research at a very high confidence level.

5.10 FACTORIAL DESIGN EXPERIMENT

As previously discussed, to enable experimentation on the main interaction between variables, a half fraction factorial experimental analysis is to be completed, comparing six significant parameters optimising the performance of the GA line balancing model. The widely available MINITAB© software package is employed to support analysis of results.

Run	Line Efficiency	Run	Line Efficiency	Run	Line Efficiency	Run	Line Efficiency
1	79.00	9	74.70	17	74.70	25	79.00
2	86.40	10	86.40	18	86.40	26	86.40
3	74.70	11	76.80	19	76.80	27	74.70
4	86.40	12	86.40	20	86.40	28	86.40
5	81.00	13	86.40	21	83.80	29	83.80
6	86.40	14	89.20	22	89.20	30	86.40
7	89.2	15	83.80	23	86.40	31	86.40
8	89.2	16	89.20	24	89.20	32	92.10

Table 5.20 Line efficiencies for the experiment

The purpose of using MINITAB is to analyse the planned thirty two parameter settings (section 4.5.1) and resultant line efficiencies as given in table 5.20. The effect estimates and the sum of squares for the thirty two treatment combinations are shown in table 5.21. Figure 5.47 presents the normal probability plot of the estimated effects.

Variable	Estimated Effect	Sum of Squares	Variable	Estimated Effect	Sum of Squares
A	6.931	384.31	*CE*	0.181	0.26
B	0.931	6.93	*CF*	0.656	3.44
C	5.631	253.66	*DE*	0.106	0.08
D	0.181	0.26	*DF*	0.206	32.19
E	0.181	0.26	*EF*	0.856	5.86
F	-0.944	7.12	*ABC*	-0.419	1.40
AB	0.131	0.137	*ABD*	0.856	5.86
AC	-3.169	80.34	*ABE*	0.206	0.33
AD	0.181	0.26	*ABF*	-0.444	1.57
AE	0.181	0.26	*ACD*	0.181	0.26
AF	2.006	32.19	*ACE*	0.181	0.26
BC	1.481	17.52	*ACF*	0.406	1.31
BD	-0.494	1.95	*ADE*	-0.444	1.57
BE	0.156	0.19	*ADF*	0.156	0.19
BF	0.106	0.08	*AEF*	-0.494	1.95
CD	0.181	0.26			

Table 5.21 estimated effects, sum of squares for the experiment

A – Fitness function	B- Crossover technique	C- Selection method
D – Number of Generations	E – Mutation technique	F – Population size

For the majority of effects, the result clusters close the center (zero) line, confirming a low level of interaction. The single factor *A*, fitness function has the largest estimated effect of 6.931(furthest from the center line), identifying this parameter as the most significant parameter of the model. The next highest of 5.631 is shown by single factor *C* (selection method). The two single parameters have been shown to create a significance level substantially higher than any other single or combination set. With interest continuing to examine the interaction results, the interactions of *AC*, *AF*, *CF*, and *EF* are also reasonably substantial and require interpretation.

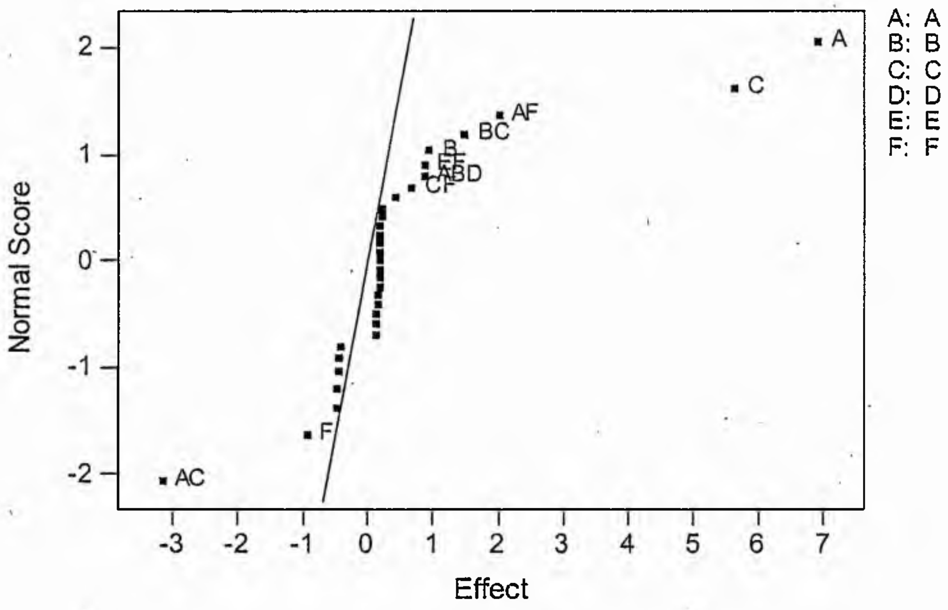


Figure 5.47 Normal probability plot of effects for experiment

Table 5.22 examines the analysis of variance obtained from experimentation. The total sum of squares (SS_T) and the model sum of squares (SS_M) for all parameters is 842.27 and 797.23 respectively. Examining the lower significance levels and selecting a threshold of 2.00 for sum of squares, twenty two of the thirty-two effects and interactions can be excluded, leaving ten single or multiple variable combinations that cover 94% of the total variability as shown. The main effects *A* and *C* are confirmed as dominant, accounting for over 75% of the total variability. Interactions *AC*, *AF* and *BC* provide 15% of the total variability. Main effects *B* ,*F* and the other three interactions (*CF*, *EF* and *ABC*) accounts for less than 3.5% in total.

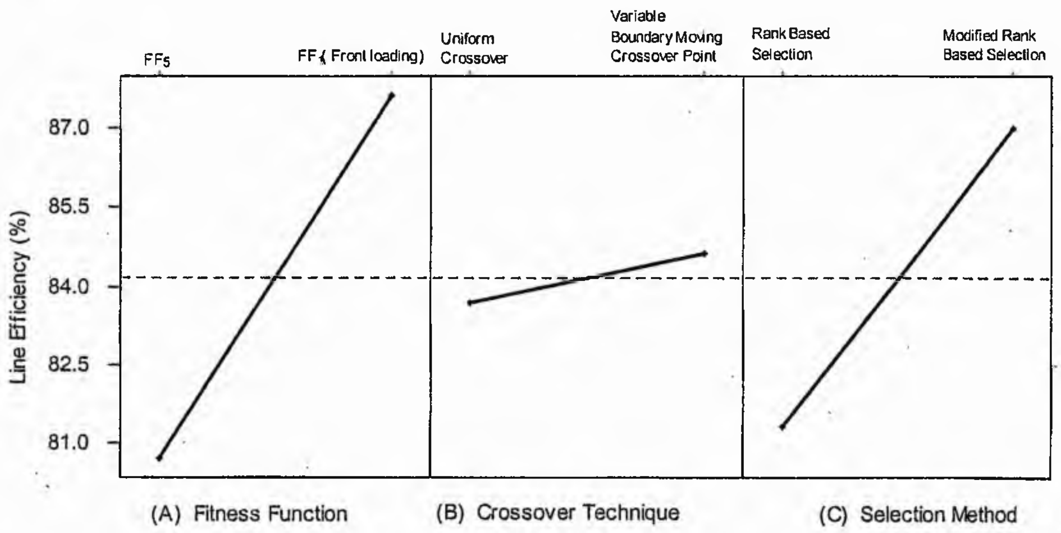
For a given confidence level α, all factors or interactions with value of $P \leq \alpha$ are statistically significant, whilst other factors may be disregarded. The two main factors, the fitness function and the selection technique, have values of $P \leq 0.05$ (at 95% confidence level) and therefore statistically significant. The factor *A*, the fitness function has a large effect on the performance and it has significant interactions with two other factors, selection technique and number of generations.

Source of Variable	Sum of Squares	Degrees of Freedom	Mean Square	F_0	P-Value
A	384.31	1	382.31	187.83	0.0001
B	6.93	1	6.93	3.38	0.0756
C	253.66	1	253.66	123.97	0.0001
F	7.12	1	7.12	3.49	0.0712
AC	80.34	1	80.34	39.38	0.0001
AF	32.19	1	32.19	15.77	0.0004
BC	17.52	1	17.52	8.58	0.0063
CF	3.44	1	3.44	1.68	0.2045
EF	5.86	1	5.86	2.86	0.1008
ABD	5.86	1	5.86	2.86	0.1003
Error	45.02	22	2.04		
Total	842.25	32			

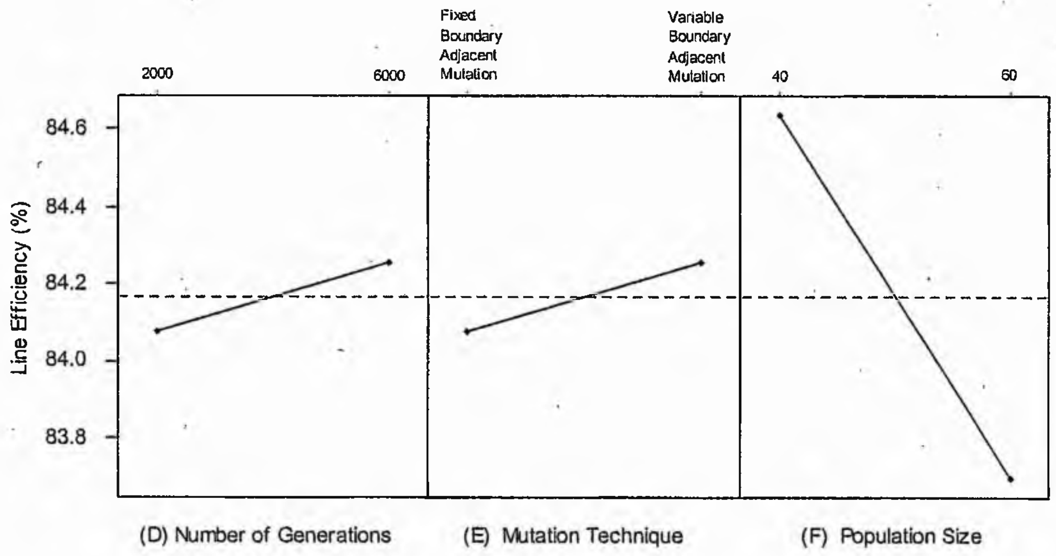
Table 5.22 Analysis of Variance for the Line Efficiency data for WE_{58} problem

Figures 5.48(a) and 5.48(b) display the main effects of the six factors. Fitness function and selection method show significant positive effect, indicating both the front loading fitness function and the modified rank based selection technique are better than either the alternative fitness function FF_5 or the original rank based selection technique. The variable boundary moving crossover technique shows a marginal improvement over the uniform crossover technique for the problem.

Increasing population size from forty to sixty shows a negative effect. Knowing the process of assigning elements to stations by the GA, the negative effect can be explained by noting the need to consider changes in population size at the same time as examining changes in the number of initial solutions, again proposed in future work.



Figures 5.48(a) Main effect plots of factors *A, B* and *C*



Figures 5.48(b) Main effect plots of factors *D, E* and *F*

In addition to crossover technique, number of generation and mutation technique show negligible effects on overall performance for the test set. The results show that for the chosen test set, increasing the number of generations above 3000 iterations did not have any effect on performance (table 4.7 and table 5.20). Future work can examine the possible existence of an iteration limit for a wider set of test cases.

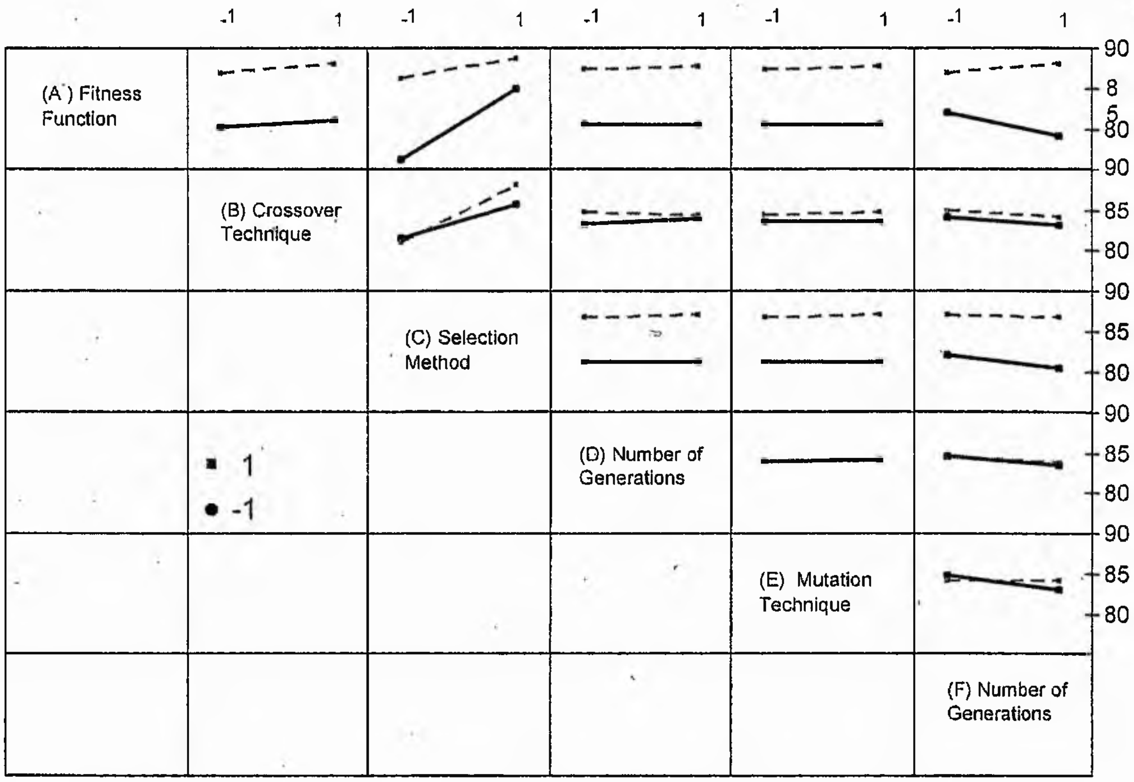


Figure 5.49(a) all possible interactions of the six factors.

Figure 5.49 (a) shows all the possible interactions of the selected six factors. Twelve interaction plots out of fifteen show almost parallel lines indicating no interactions even if they go up or down. The interactions *AC*, *BC* and *AF* show non parallel lines and they need interpretations. Figures 5.49 (b), 5.49(c) and 5.49(d) display the significant interactions of *AC*, *BC* and *AF* respectively.

Figure 5.49 (b), drawn from figure 5.49(a), exhibits a strong interaction between fitness function and selection method. Both fitness functions FF_1 and FF_5 display better performances with the modified rank based selection technique. Notably, the front loading fitness function (FF_1) and modified selection method combination, shows better line efficiency of over 88% , a superior result to the FF_5 and modified selection method combination.

(

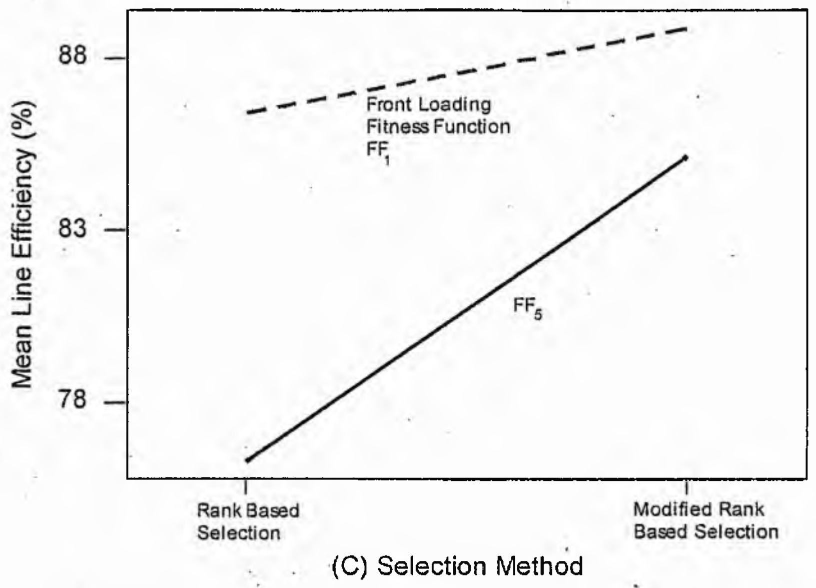


Figure 5.49(b) interaction diagram for selection and fitness function

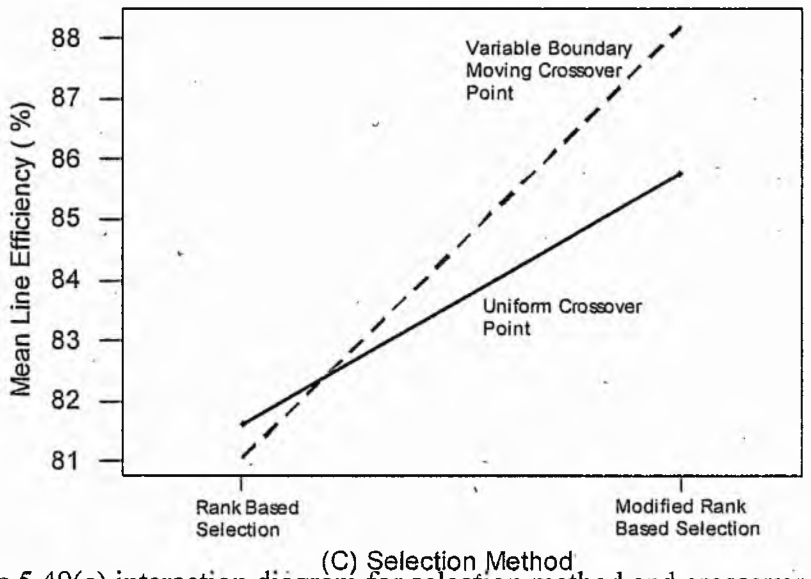


Figure 5.49(c) interaction diagram for selection method and crossover technique

A significant interaction between selection method and crossover technique can be seen in figure 5.49(c). Both crossover techniques display positive interaction when combined with the selection methods. Rank based selection technique combined with the uniform crossover method produced better results than the Variable Boundary Moving Crossover Point (VBMCP) technique plus rank based selection combination. Modified rank based selection technique combined with VBMCP shows a significant improvement over the alternative modified rank based technique plus uniform crossover.

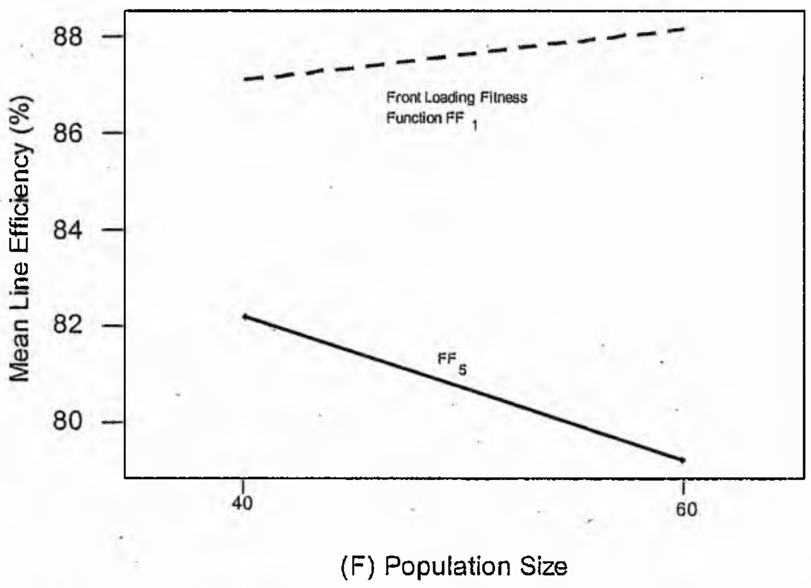


Figure 5.49(d) interaction diagram for population size and fitness function

The interaction between population size and fitness function is complex (figure 5.49(d)). The front loading fitness function shows a slight increase of performance with a larger population of 60. Whereas, the fitness function (FF_5) shows substantial negative effect with a larger population. The conclusion is to revisit this parameter combination with a wider range.

Summarising the results of the half fractional factorial analysis, the conclusions possible are:

A. Main effects: The front loading fitness function and the modified rank based selection technique have significant positive effect on the model performances.
B. The remaining four single variables have no significant main effect.
C. Cross variables: The interaction between the front-loading fitness (FF_1) function and modified rank selection method is important. However, the interaction between the front loading fitness function and the variable

boundary crossover point method is marginal. This was significant in the single factor experiment.

D. The remaining combination effects were of minor significance for the test case programme.

CHAPTER 6

CONCLUSIONS

The main objective of this research was to develop a Genetic Algorithm line-balancing model for the single model assembly line balancing problem capable of outperforming existing GA models and the Hoffmann precedence matrix technique. This was accomplished by defining a new fitness function (based on the front-loading theory), crossover and mutation techniques, and modifying an existing selection technique. The main objectives of the research have been achieved and the performance of the model was interesting. The following conclusions can be made based upon the results of the study.

The findings in Chapter 5 appear to indicate that the Genetic Algorithm model achieved the overall best performance irrespective of problem size and complexity. Forty-four test problems out of fifty-four (81%) achieved the best solutions obtained by any technique so far which is twenty-four problems (44%) more than that obtained by the Hoffmann procedure. In twenty-three instances, the Genetic Algorithm model achieved the optimum solutions, which are seventeen instances more than that achieved by the Hoffmann precedence procedure.

The modified Hoffman heuristic technique developed based on the front-loading theory achieved the second overall best, which is nine cases less than that achieved by the Genetic Algorithm model. Only in problems, SH_{297}^{2488}, BA_{148}^{170}, BA_{148}^{152} and BU_{29}^{27} were the solutions obtained by the model one workstation more than that generated by the modified Hoffmann technique. For test problems with a large number of elements (*n* >100) with task time distribution skewed towards the minimum element time (SH_{297} and BA_{148}) and cycle time just above the maximum task time, the modified Hoffman technique outperformed the genetic model. However, for hard problems, in which task time distribution skewed towards the maximum element time and cycle time just above the maximum element time, (WA_{58} and LU_{89}) the solutions obtained by the model is superior to both the Hoffmann and Modified Hoffmann methods.

Therefore, it can be concluded that the new Genetic Algorithm model is better than the Hoffmann precedence matrix procedure and it is more suitable for complex assembly line balancing problems for which other heuristic methods generate solutions far away from optimality.

The key feature of the genetic model is the front-loading fitness function. This fitness function preformed well in obtaining optimum solutions for the assembly line balancing problem and outperformed all the existing GA fitness models. The simulation results, particularly those obtained from test cases KW_{45}^{69}, KW_{45}^{92}, KW_{45}^{138} and KW_{45}^{184}, verified the front-loading theorem, and the propensity to achieve one hundred percent balanced solutions.

The proposed fitness function that drives the algorithm towards forward loading demonstrated excellent performance, predominantly in test cases where $m^* <$ 20. However, in cases where the theoretical minimum number of workstations is greater than 25, the function is not powerful enough for forward loading especially in latter workstations (>20). This is mainly due to the characteristics of the negatively exponential (R^{-x}) fitness function, which shows almost constant values where x > 20, resulting in a marginal fitness change for elements transferring among latter workstations

This limitation was eliminated to a certain extend by controlling the front-loading constant (*R*) in the fitness function and following values are recommended for better performance.

Theoretical number of workstations (m^*)	R
$m^* \leq 10$	1.5
$10 < m^* \leq 20$	1.3
$m^* > 20$	1.1

Table 6.1. Recommended R-values

The second control parameter is the number of generations permitted for a workstation (G). It seemed that most of the selected problems achieved very good solutions running the algorithm for 3000 generations (i.e., 120 generations per workstation) and is recommended irrespective of the complexity of the problem for test cases where $m^* < 25$.

The CPU time variation against G showed a linear relationship and therefore the model does not become especially time consuming when increasing the number of generations per workstation. This property is very useful for solving very hard problems and the model can be left to run for long enough to find better solutions.

The most economical population size for the model was 40 for the problems studied and increasing population size decreases the diversity and consumes extra CPU time. On the other hand, too low populations showed premature convergence.

Different initial populations were examined and a well-seeded population is favoured for the genetic model. It was confirmed that a population containing all bin-packing solutions would hardly generate a feasible solution even after 10,000 generations. Anderson and Ferries (1990) claimed stating with a solution generated by the COMSOAL algorithm increases the convergence, but this model achieved excellent results with solutions generated by the random task assignment technique. One possible explanation is that solutions obtained by this technique are not closely packed unlike the Hoffmann and COMSOAL procedures, and therefore give enough room for the forward loading process.

Whitley (1989) claimed the rank-based selection technique is superior to all the other selection techniques and the findings reported in this thesis are consistent with his claim. The main drawbacks of this technique are high selective pressure and low p opulation d iversity. T he m odified r ank-based s election t echnique u sed i n t his research overcame this problem effectively and it performed well on all the selected problems irrespective of the problem size and complexity. The user control the maximum number of identical chromosomes in the selection pool and it is seems that one third of the selection pool size is a typical value for the test problems studied.

A new crossover technique called Variable Boundary Moving Crossover Point (VBMCP) was developed for the model to propagate the bulk of the good attributes of the parents to their offspring. The crossover probability was set to 0.80 based on the findings of De Jong (1975). The technique was compared with six existing techniques developed for the assembly line balancing problem and the performance was encouraging. It outperformed reported techniques and also enhanced the progressive filling of workstations in the front-loading process.

The main control parameter of the above technique, other than the crossover probability, is the crossover span size. The results showed that the overall performance of the model significantly depend on the crossover span size, which is a test problem constant. In this research, the crossover span size was fixed to 0.5 c_s (where c_s is the maximum number of elements that can be packed in to a bin size of the cycle time), and increasing it resulted in propagation of both good and bad attributes of parents to offspring. It is apparent that as workstations are filled, the number of elements available for filling will decrease and therefore, c_s should vary accordingly. But t his a spect w as n ot considered i n t his r esearch a nd l eft for future research work.

Generally, duplicating elements in chromosomes after crossover is inevitable and these chromosomes must be repaired. Two new repair techniques were experimented with in this research and the one using rank positional weights of elements in the reassigning of elements gave better results. It generated more feasible solutions than the other techniques without consuming extra CPU time.

The model consists of the Variable Boundary Adjacent Mutation technique for the mutation operation. This technique was compared with five other mutation techniques and it showed good overall performance. The main idea behind the technique is to progressively move the mutation zone with the crossover span and select adjacent elements for mutation. This process strengthens the progressive filling of workstations as the Variable Boundary Moving Crossover Point technique (VBMCP). The key controlling parameter is the mutation span size and experimental results showed that a mutation span of $2c_s$ gives better performance for the test cases considered. A part from that, the mutation probability p_m was set to 0.01 based upon previous findings of other researchers.

As Rudolph (1994) pointed out, copying the best chromosomes to the next generation is vital for rapid convergence and achieving near optimum solutions. However, the number of elite chromosomes must be determined carefully, copying more elite chromosomes leads to low diversity and premature convergence. Based on the test results 20-25% of the best chromosomes are recommended.

This study has taken a step in the direction of developing a Genetic Algorithm model for the assembly line balancing problem. The model was developed by integrating a number of Genetic Algorithm components including fitness function, genetic operators etc., and these components have their own control parameters which give the best performances. Therefore, as a whole, there are a large number of parameters in the model that have to be evaluated for optimum performance. These parameters are closely linked with the complexity of the line-balancing problem and they vary significantly from problem to problem. For example, two cycle times of the same precedence network could have a marked difference in complexity; therefore, it is worth stating that these significant parameters should be properly tuned before actually being used to solve a real assembly line balancing problem by conducting a number of pilot runs.

In general, the Genetic Algorithm model consumes more CPU time than heuristic procedures although it mainly depends on the number of generations. However, it can be minimized by distributing computations to a number of processors (parallel processing), which will be discussed in the next Chapter.

The single factor design experiments, varying one experimental condition at a time repeating under the same conditions has been applied in this research to identify and examine the leading parameter combinations. The follow-on half fractional factorial design, included to establish the cross variable effects has been added. Further conclusions in relation to both individual parameters and combined parameter interaction is summarised as follows.

The half fractional factorial design experiment confirmed the two most significant individual parameters as front loading fitness function and the modified rank based selection technique. For combinations of parameters, new conclusions confirm, the main feature of the GA model, the front loading fitness function has a high level of interaction with the modified rank based selection method and a low level of interaction with the variable boundary moving crossover technique. This crossover technique was originally designed to support the forward loading process; however the results show that the support is marginal for the selected test case. Overall results from the half fractional factorial design support the conclusion that the new features of the GA line balancing model collectively show better performance than the previously published line balancing GA models.

CHAPTER 7

SUGGESTIONS FOR FURTHER RESEARCH

It was stated in the previous chapter that the new fitness function is not powerful enough in the forward loading of elements, especially for problems with workstations of more than twenty. This is mainly due to the levelling off of the front-loading weights in latter workstations (because of the negative exponential characteristics of the loading curve (R^{-x})). In order to maintain the same strength of loading among latter workstations, the characteristics of the fitness curve must be modified. Changing the loading weight curve every 20 workstations could be one possible modification and the modified curve can be mathematically expressed as follows.

$$R_j = \begin{cases} \frac{a}{2} + R^{m-j} & K < 20 \\ \frac{a}{2} R^{m-j+20} & 20 \le K < 40 \end{cases}$$

At the beginning of the evaluation, K =1, and if $\left(\frac{S_j}{C}\right) = 1$ or the generation number equals to $G*K$, the value of K is incremented by one unit. Where G is the number of generations per workstation, a is a loading function constant depends upon the cycle time (C), the number of elements (n) and the theoretical minimum number of workstations ($m*$) , R_j is the loading weight of the jth workstation and the other parameters are as defined before.

The modified loading weight curve for a =1000 and R = 2.0 is displayed in figure 7.1, which shows a marked difference of loading weights among later workstations. Further research must be carried out with the modified function to verify its power of forward loading for test problems of workstations more than 20.

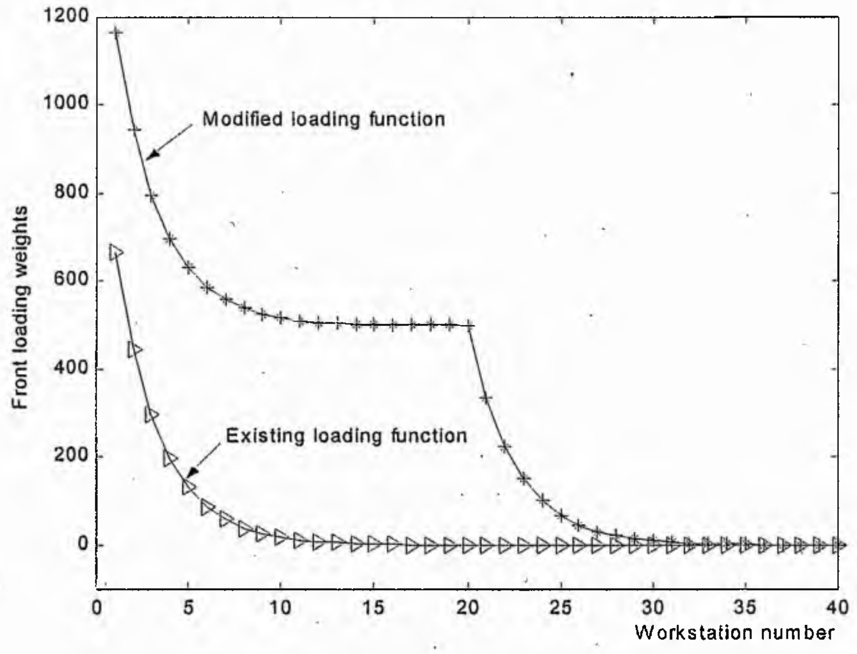


Figure 7.1 Existing and modified front-loading functions

The complexity of the test problem has a significant impact on the performance and the need of more time to explore larger domains has been indicated. If the model was left to run for long enough, the solutions would have been much more closer to the optimum solution. The proposed model is not intelligent enough to make a decision on the number of generations per workstation required according to the complexity. Therefore, an index must be defined and integrated into the model to guide the number of generations by analysing the current sector of the precedence network where front-loading takes place.

Since the genetic algorithm is considering a large number of solutions in the search domain, the computational time of the genetic model is relatively high compared to that of heuristic techniques. However, as genetic algorithms are good candidates for effective parallelization, implementing parallel genetic algorithms can significantly increase computational power and thus reduce CPU time consumption. Additionally, evaluating a number of subpopulations at the same time and swapping best chromosomes among the populations allows faster convergence. Therefore, implementing parallel genetic algorithms for the genetic model would certainly improve its computational power and extend its applicability for real world problems.

The surveys of Chase (1974) and Milas (1990) showed only a few companies utilizing the published techniques to balance their lines. The reason for this were reported to be the practitioner's unfamiliarity with the published techniques, the complexity of algorithms and the inflexibility to model the actual conditions of assembly lines. Additionally, the need of complete, user friendly and ready to use software for practitioners was highlighted in each survey. The model in this thesis utilises MATLAB software, with problem specifications and control parameters entered using a command window. Since MATLAB offers Graphical User Interface (GUI) tools, this feature can be used to develop a powerful, user-friendly genetic algorithm model for line balancing.

With regard to test case coverage, by starting with a small, totally enumerable problem (30-element, Sawyer 1970), the effect on processing time, solution generation and solution quality of problem size can be examined up to the largest known published problem (297-element, Scholl 1999).

Extending the review of parameters is also an area of future interest. Six parameters were identified as problem dependent (front-loading constant, crossover and mutation span size, repair technique, the number of elite chromosomes and feasible solutions in the initial population). An added test case programme would further enhance the analysis of effect on the six remaining parameters.

Research in this dissertation has shown the frontloading fitness function and modified selection technique have a definable contributing relationship. Confirming this further with a set of test cases at the extremes of line balancing conditions will lead to interesting further publications. The extreme conditions would include Order Strength (from 20.00 to 75.00, noting the current test case was 59.1), task time variation (distributions with positively and negatively skewed towards the cycle time) and finally problem size variation (from 30 to 295 elements).

The use of MINITAB proved a powerful support to the detailed half fractional analysis. The opportunity to further use the software to add a full fractional analysis should not be missed.

REFERENCES

Ackley, D, 1987. *A connection machine for genetic hill climbing.* Kluwer Academic Publisher, Boston, USA.

Agrawal, P K, 1985. The related activity concept in assembly line balancing. *International Journal of Production Research,* 23(2), 403 - 421.

Akagi, F, Osaki, H and Kikuchi, S, 1983. A method for assembly line balancing with more than one worker in each station. *International Journal of Production Research,* 21(3),755-770.

Alippi, C, and Treleaven, P, 1991. GAME: A Genetic Algorithm Manipulation Environment in *Internal Report Department of Computer Science,* UCL.

Anderson, E, and Ferris, M C, 1990. A genetic algorithm for the assembly line balancing problem. *Computer Science Technical Report #926,* Cambridge, England.

Anderson, E, and Ferris, M C, 1994. Genetic algorithms for combinatorial optimisation: the assembly line balancing problem. *ORSA Journal on Computing,* 6, 2, 161-173.

Arcus, A L, 1963. *An analysis of a computer method of sequencing assembly line operations.* Ph.D. dissertation, University of California, Berkeley, CA, USA.

Arcus, A L, 1966. COMSOAL: A computer method of sequencing operations for assembly lines. *International Journal of production research,* 4(4), 259-277.

Baker, L E, 1985. Adaptive selection methods for genetic algorithms. *Proceedings of the First International Conference on Genetic Algorithms and their Applications.* Lawrence Erlbaum Associates, Hillsdale, NJ, USA.421-439.

Balas, E, 1965. An additive algorithm for solving linear programs with zero-one variables. *Operation Research,* 113, 517-546

Barnes, R M, 1980. *Motion and time study: Design and measurements of work.* 7th ed., Wiley, New York. USAA.

Bartholdi, J J, 1993. Balancing two sided assembly lines: a case study. *International Journal of Production Research,* 31(2), 2447-2461.

Baybars, I, 1986a. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science,* 32(8), 909-932.

Baybars, I, 1986b. An efficient heuristic method for the simple assembly line balancing problem. *International Journal of Production research,* 24(1), 149-166.

Bennett, G B, and Byrd, J, 1976. A trainable heuristic procedure for the assembly line balancing problem. *AHE Transaction.*8, 195-201.

Bhattacharjee, T K, and Sahu, S, 1990. Complexity of single model assembly line balancing problems. *Engineering Costs and Production Economics,* 18, 203 -214.

Bowman, E H, 1960. Assembly Line Balancing Liner Programming, *Operation Research,* 8, 385-389.

Bruns, R, 1993. Direct chromosome representation and advanced genetic operation scheduling. *Proceedings of the 5th International Conference on Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, USA, 352-359.

Bryton, B, 1954. *Balancing of a continuous production line.* M.Sc. Thesis, North-western University, Evanston, ILL, USA.

Buffa, E S, 1983. *Modern production operations management.* Wiley, New York.

Buxey, G M, 1974. Assembly line balancing with multiple stations. *Management Science*, 29(6), 1010-1021.

Buxey, G M, and Sadjadi, D, 1976. Simulation studies of conveyor-paced assembly lines with buffer capacity. *International Journal of Production Research*, 14(3), 607-624.

Buxey, G M, 1979.The nature of manual, moving belt flow lines with overlapping stations. *International Journal of Production Research*, 17, 143-154.

Buzacott, L A, and Shanthikumar, L G, 1993. *Stochastic models of manufacturing systems.* Prentice Hall, Englewood Cliffs, NJ, USA.

Cavicchio, D J, 1970. *Adaptive search using simulated evolution.* Doctoral dissertation, University of Michigan, Ann Arbor, MI, USA.

Chambers, L D, 1999. *Practical handbook of genetic algorithms,* applications Volume I, CRC Press.

Chang, Y L, and Sullivan, R S, 1991. *QS (Quant Systems),* Version2, Englewood Cliffs, NJ, USA.

Chase, R B, 1975. Survey of paced assembly lines. *The Journal of Industrial Engineering.* 6, February.

Chow, W M, 1990. *Assembly line design: methodology and applications.* Dekker, New York, USA.

Dar-El, E M, 1973. MALB - A heuristic techniques for balancing large single-model assembly line.' *AIIE Transaction.5,* 343-356.

Dar-El, E M and Rubinovitch, Y, 1979. MUST- A multiple solutions technique for balancing single model assembly lines. *Management Science.* 25(11), 1105-1114.

Davis, L, 1985a. Applying adaptive algorithms to epistemic domains. *Proceedings of the 9th International Joint Conference on Artificial Intelligence,* 162-164.

Davis, L,1985b. Job shop scheduling with genetic algorithms. *Proceedings of the First International Conference on Genetic Algorithms and their Applications.* Lawrence Erlbaum Associates, Hillsdale, NJ, USA,136-140.

Davis, L, 1991. *Handbook of genetic algorithms,* Van Nostrand Reinhold, New York, USA.

De John, K A, 1975. *An analysis of the behaviour of class of genetic adaptive systems.*(.Doctoral dissertation, University of Michigan).Dissertation abstract international 36(10), 5140B (University Microfilms no. 76-93).

Domschke, W, and Drexl, A, 1998. *Einfuhrung in Operations Research.* 4th ed., Springer, Berlin, Germany.

Downey, B S, and Leonard, M S, 1992. Assembly line with flexible work-force. *International Journal of Production Research,* 30(3), 469-483.

Driscoll, J, and Thilakawardana, D, 2000a. Definition and evaluation of assembly line solution. 10th *International conference on Flexible Automation and Intelligent Manufacturing,* Maryland, USA, 1157-1166.

Driscoll, J, and Thilakawardana, D, 2000b. Using quantitative parameters to guide assembly line balancing. *International Conference on Production Research,* AIT, Bangkok.

Driscoll, J, and Thilakawardana, D, 2000c. The definition of assembly line balancing difficulty and evaluation of balance solution quality. *International Journal of Robotics and Computer Integrated Manufacturing.* 17,81-86.

Driscoll, J, Thilakawardana, D, and Deacon G, 2001, Cost function experimentation in genetic algorithm line balancing. *Proceedings of the 16th International Conference on Production Research*, Prague, Czech Republic.

Easton, F F, 1990. A dynamic program with fathoming and dynamic upper bounds for the assembly line balancing problem. *Computers and Operations Research,* 17,163-175.

Elmaghraby, S E, and Herroelen, W S, 1980. On the measurement of complexity in active networks. *European Journal of Operation Research*, 5, 223-234.

Erel, E, Sarin, S C, 1998. A survey of the assembly line balancing procedures, *Production Planning and Control,* 9, 414-434.

Faland, B H., Klastorin, T D, Schmitt, T G, and Shtub, A, 1992. Assembly line balancing with resource dependent task times. *Decision Sciences,* 23, 343-364.

Falkenauer, E, 1992. *Generic algorithms and grouping problems.*. John Wiley & Sons, Chichester, England.

Falkenauer, E, and Delchambre, A, 1992. A generic algorithm for bin packing and line balancing. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation,* Nice, France-May 1992.

Filho, J L R, Alippi, C, and Treleaven, P, 1993. *Genetic algorithm programming environment in* Parallel Genetic Algorithm: theory and applications, IOS Press.

Fisher, R A, 1958. *The genetic theory of natural selection.* New York: Dover.

Forsyth, R S, 1989. *Machine learning principles and techniques.* Chapman & Hall.

Fox, M S, and McMahon, M B, 1991. Genetic operators for sequencing problems. *First workshop on the Foundation of Genetic Algorithms and Classifier Systems,* Morgan Kaufmann Publishers, San Mateo, CA, USA.

Frantz, D R, 1972. *Non-linearities in genetic adaptive search.* Doctoral dissertation, University of Michigan, MI, USA.

Gagnon, R J, and Ghosh, S, 1991. Historical roots: research life cycle and future directions. *OMEG.* 19.

Garey, M R, and Johnson, D S, 1979. *Computers and intractability - A guide to the theory of NP-completeness.* W.H.Freeman, San Fransisco, USA.

Geoffrion, A M, 1967. Integer Programming by Implicit Enumeration and Balancing Method. *SIAM Rev.*, 9,178-190.

Gehrlein, W V, and Patterson, H, 1978. Balancing single-model assembly lines: comments on a paper by E M Dar-El. *ALLE Transactions, 10,* 109-112.

Ghosh, S and Gagnon R L, 1989. A comprehensive literature review and analysis of the design, balancing, and scheduling of assembly systems. *International Journal of Production Research,* 27, 637-670.

Goldberg, D E, 1989a. *Genetic algorithms in search, optimisation, and machine learning,* Addison-Wesley, Reading, MA.

Goldberg, D E , 1989b. Sizing populations for serial and parallel genetic algorithms, *Proceedings of the third International Conference on Genetic Algorithms,* George Manson University, Morgan Kaufmann, San Mateo, CA, USA.

Goldberg, D E and Lingle, R, 1985. Alleles, loci, and travelling salesman problem, *Proceedings of an International conference on Genetic Algorithm and their Applications,* Carnegie-Mellon University, Pittsburgh, PA, Lawrence Erlbaum Associates, Hillsdale, NJ, USA.154-159.

Graves, S C and Lamar, B W, 1983. An integer programming procedure for assembly system design problems. *Operations Research.* 31, 522-545.

Grefenstette, J J, 1981. GENESIS: A system for using genetic search procedures. *Proceedings of the Conference on Intelligent Systems and Machines,* 161-165.

Grefenstette, J J, (Editer), 1986. Optimisation of control parameters for genetic algorithms. *IEEE Transactions on Systems,* Man and Cybernetics, vol. 16,1,122-128.

Grefenstette, J J, 1987. *Proceedings of the Second International Conference on Genetic Algorithms,* Lawrence Erlbaum Associates, Hillsdale, NJ.323-342.

Gutjahr, A L and Nemhauser, G L, 1964. An algorithm for the line-balancing problem. *Management Science.* 11(2), 308-315.

Gunther, R E, Johnson, G D and Peterson, R S, 1983. Currently practised formulations for the assembly line balance problem, *Journal of Operations Management*, 3, 209-221.

Hackman, S T, Magazine, M L and Wee, T S, 1989. Fast, Effective algorithms for simple assembly line balancing problems. *Operations Research.* 37, 916-924.

Held, M , Karp, R M., and Shareshian, R , 1963. Assembly line balancing dynamic programming with precedence constraints. *Operations research.* 11, 442-459.

Helgeson, W B and Birnie, D P, 1961. Assembly line balancing using the Ranked Positional Weight technique. *The Journal of Industrial Engineering.* 12, 394-398.

Hoffmann, T R, 1963. Assembly line balancing with a precedence matrix. *Management Science.* 9(4), 551-562.

Hoffmann, T R., 1990. Assembly line balancing: a set of challenging problems. *International Journal of Production Research,* 28, 1807-1815.

Hoffmann, T R, 1992. EUREKA: A hybrid system for assembly line balancing. *Management Science,* 38(8), 39-47.

Holland, J H, 1975. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, USA.

Ignall, E J, 1965. A review of assembly line balancing. *Journal of industrial Engineering,* 16, 244-254.

Jackson, J R, 1956. A computing procedure for line balancing problem. *Management Science.* 2, 261-271.

Johnson, R V, 1988. Optimally balancing large assembly lines with "FABLE" on Hoffmann's data sets. *Management Science*, 39, 1190-1193.

Karp, R M, 1972. Reducibility among combinatorial problems. *In complexity* of Computer Applications, R E Miller and J W Thatcher (eds.), Plenum, New York, 85-104.

Kilbridge, M D and Wester, L, 1961a. A heuristic method of assembly line balancing. *The Journal of Industrial Engineering,* 12, 292-298.

Kilbridge, M and Wester, L, 1961b. The Balancing Delay. *Management Science.* 8(1), 69-84.

Kilbridge, M, and Wester, L, 1962. A review of analytical systems of line balancing. *Operations research.* 10(5),626-638.

Kim, Y K, Kim, Y J, and Kim, Y, 1996. Generic algorithms for assembly line balancing with various objectives, *Computer and Industrial Engineering,* 30(3), 397-409.

Kim,Y J, Kim, Y K, and Cho, Y ,1998. A heuristic based genetic algorithm for workload smoothing in assembly lines. *Computers Operations Research.* 25, 99-111.

Klein, M , 1963. On assembly line balancing. *Operations research.* 11, 274-281.

Laszewski, V G, 1991. Intelligent structural operators for k-way group partitioning problem. *Proceedings of the 4th International Conference on Genetic Algorithms,* R Belew and L Booker, Morgan Kaufmann, 492-497.

Lau, H S and Shtub, A, 1987. An exploratory study on stopping a paced line when incompletion occur. *IEEE Transactions,* 19, 463-467.

Leu, Y Y, Matheson, L A, and Rees, L P, 1994. Assembly line balancing using genetic algorithms, with heuristic generated initial populations and multiple evaluation criteria. *Decision Sciences,* 25, 581-606.

Liepins, G E, and Hilliard, M R, 1989. Genetic algorithms: foundations and applications. *Annals of Operations Research,* 21,31-58.

Liepins, G E, and Vose, M D, 1990. Representational issues in genetic optimisation. *Journal of Experimental and Theoretical Artificial Intelligence,* 2, 101-115.

Lutz, L, 1974. *Abtakten von montagelinien.Krausskopf,* Mainz.

Mastor, A A, 1970. An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science,* 16(11),728-746.

MATLAB, 1998. *Version 5 User's Guide,* Prentice Hall.

Mansoor, E M, 1964. Assembly line balancing –an improvement on the Ranked Positional Weight technique. Journal of Industrial Engineering, 15, 73-77 and 322-323.

Mercer, R E, 1977. *Adaptive search using a reproductive meta-plan.* Master's thesis, University of Alberta, Edmonton, Canada.

Mertens, P, 1967. Flibbandabstimmung mit dem Verfahren der begrenzten Enumeration nach Muller-Merbach. Ablauf-und Planungsforschung, 8,429-433.

Milas, G H, 1990. Assembly line balancing, let's remove the mystery. *Industrial Engineering,* (May), 31-36.

Mitchell, J, 1957. A computational procedure for balancing zoned assembly lines. Research report 6-94801-1R3,Westinghouse Research Laboratories, Pittsburgh.

Miltenburg, L and Wijngard, L, 1994. The U-line balancing problem. *Management Science,* 40, 1378-1388.

Michalewicz, Z and Xiao, J, 1995. Evaluation of paths in evolutionary planner/navigator. *Proceedings of the International Workshop on Biologically inspired Evolutionary Systems.* Tokyo. Japan.45-52.

Minagawa, M and Kakazu, Y, 1992. A genetic approach to line balancing. *Human Aspects in Computer Integrated Manufacturing,* 73 7-743.

Moodie, C L, and Young, H H, 1965. A heuristic method of assembly line balancing for assumptions of constant or variable work element times, *Journal of Industrial Engineering,* 16.

Mukherjee, S K and Basu, S K, 1964. An application of heuristic method of assembly line balancing in an Indian industry. *Proceedings of Institution of Mechanical Engineers,* 178/1/11,277-292.

Murata, T, Ishibuchi, H, and Tanaka, H, 1996. Multi objective genetic algorithm and its application to shop flow scheduling. *Computers and Industrial Engineering,* 30, 4,957-968.

Nakasu, M M, and Leung, KH, 1995. A stochastic approach to assembly line balancing. *International Journal of Production Research,* 33(5), 975-991.

Nevins, A J, 1972. Assembly line balancing using best bud search. *Management Science,* 18(9), 529-539.

Nevins, J L and Whitney, D E, 1980. Assembly research. *Automatica,* 16,595-613.

Oliver, I M, Smith, D J, and Holland, J R C, 1978. A study of permutation crossover operators on the travelling salesman problem. *Proceedings of the second International Conference on Genetic Algorithms,* Lawrence Erlbaum Associates, Hillsdale, NJ. 224-230.

Patterson, J H and Albracht, J, 1975. Assembly line balancing :zero-one programming with Fibonacci search. *Operations Research,* 23, 166-172.

Pinto, P A, Dannenbring, D G and Khumawala, B M, 1978. A heuristic network procedure for the assembly line balancing problem. *Naval Research Logistics Quarterly,* 25, 299-307.

Pinto, P A, Dannenbring, D G and Khumawala, B M, 1981. Branch and bound and heuristic procedures for assembly line balancing with paralleling of stations. *International Journal of Production Research,* 19(2), 565-576.

Ponnambalam, S G , Aravindan, P, and Naidu, G M, 2000. A multi-objective genetic algorithm for solving assembly line balancing problem. *International Journal of Manufacturing Technology,* 16(5), 341-3 52.

Powell, D, and Skolnick, M W, 1993.Using genetic algorithm in engineering design optimisation with non-linear constraints. Proceedings of the fifth International Conference on Genetic Algorithm, Morgan Kaufmann. San Mateo, CA.424-430.

Prenting, T O and Battaglin, M , 1964. The precedence diagram: a tool for analysis in assembly line balancing. *Journal of Industrial Engineering,* 15, 208-213.

Prenting, T O and Thomopoulos, N, 1974. *Humanism and technology in assembly line* systems. (Hayden Book Company).

Rachamadugu, R M V, and Talbot, B, 1991. Improving the equality of workload assignments in assembly lines. *International Journal of Production Research,* 29(6),755-768.

Reeve, N R, 1971. *Balancing continuous assembly lines.* PhD dissertation, State University of New York at Buffalo, NY, USA.

Robbins, G, 1992. EnGENEer-The Evolution of solutions in *Proceedings of the 5th Annual Seminar on Neural Networks and Genetic Algorithms.*

Rosenberg, O and Ziegler, H, 1992. A comparison of Heuristic algorithms for cost-oriented assembly line balancing. *Zeitshrift fur Operations·Research*, 36, 477-495.

Rubinovitz, J and Levitin, G, 1995. Genetic algorithm for assembly line balancing. *International Journal of Production Economics,* 419(5),343-354.

Rudolph, G, 1994. Convergence analysis of canonical genetic algorithms, *IEEE Transactions on Neural Networks, special issue on evolutionary computation,* volume 1.

Sabuncuoglu, I, Erlel, E and Tanyer, M, 2000. Assembly line balancing using genetic algorithms. *Journal of Intelligent Manufacturing,* 11(3), 295-310.

Salveson, M E, 1955. The assembly line balancing problem. *The Journal of Industrial Engineering,* 6(3), 18-25.

Sarker, B R and Shanthikumar, J G, 1983. A generalized approach for serial or parallel line balancing. *International Journal of Production Research,* 21,109-133.

Sawyer, J H F, 1970. Line balancing. The Machinery Publishing, Brighton.

Schofield, N A, 1979. Assembly line balancing and the application of Computer techniques. *Computer and Industrial Engineering.* 3(1), 53-69.

Scholl, A, 1993. *Data of assembly line balancing problems.* Schriften zur Quantitativen Betriebswirtschaftslehre, 16/93,TH Dannstadt.

Scholl, A and Vob, S, 1996. Simple assembly line balancing-heuristic approaches. *Journal of Heuristics,* 2, 217-244.

Sholl, A and Klein, R, 1999. Balancing assembly lines effectively-a computational comparison. *European Journal of Operational Research,* 114(2), 50-58.

Scholl, A , 1999.Balancing and sequencing of assembly lines.2nd ed., Physica-Verlag.

Scharge, L and Baker, K R, 1978. Dynamic programming solution of sequencing problems with precedence constraints, *Operations Research* 26, 444-459.

Shtub, A. and Dar-El, 1990. An assembly chart oriented assembly line balancing approach. *International Journal of Production Research,* 28(8), 1137-1151.

Smith, A. 1982. Wealth of Nations, *Liberty Fund Inc.*

Starkweather, T ,Mcdaniel, S, Mathias, K, Whitly, D and Whitly, C, 1991. A comparison of genetic sequencing operators. *Proceedings of the 4th International Conference of Genetic Algorithms and their Applications.* 69-76.

Suresh, G, Vinod, V V, and Sahu, S, 1996. A genetic algorithm for assembly line balancing, *Production planning and Control,* 7, 38-46.

Syswerda, G, 1989. Uniform crossover in genetic algorithm, *Proceeding of the third International Conference on genetic Algorithms,* George Mason University, 4-7 June, CA, Morgan Kaufmann, San Mateo.

Syswerda, G, 1990. Schedule optimisation using genetic algorithms. Davis (Ed), *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York, NY, USA.

Talbi, E G and Bessiere, P, 1991. A parallel genetic algorithm for the graph-partitioning problem. *Proceedings of ACM-ICS91 (International Conference on Super-computing),* Koln, Germany.

Talbot, F B and Patterson, J H, 1984. An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science,* 30, 85-99.

Talbot, F B, Patterson J H and Gehrlein, W V, 1986. A comparative evaluation of heuristic line balancing techniques. *Management Science,* 32, 430-454.

Thangavelu, S R and Shetty C M, 1971. Assembly line balancing by zero-one integer programming. *AIIE Trans,* 3, 61-68.

Thilakawardana, D, Driscoll, J, and Deacon, G, 2002. A new assembly line balancing technique- A modified version of Hoffmann's procedure, Belfast, Ireland.557-566.

Tonge, F M, 1960. Summary of heuristic line balancing procedure. *Management Science,* 7, 2139.

Tonge, F M, 1961. *A heuristic program for assembly line balancing.* Prentice Hall Englewood Cliffs, NJ, USA.

Tonge, F M, 1965.Assembly line balancing using probabilistic combinations of heuristics. *Management Science,*11,727-735.

Tsujimuya, Y, Gen, M and Kubota, E, 1995. Solving fuzzy assembly line balancing problem with genetic algorithms. *Computers and Industrial Engineering,*1(4), 543-547.

Van Assche, F and Herroelen, W S , 1979. An optimal procedure for the single model deterministic assembly line balancing problem. *European Journal of Operational Research,* 3(4), 142-149.

Warnecke, H J, 1971. Anwendung mathematischer methoden bei der leistungsabstimmung von montagelinien. *XPR-Annals* 20, 99-100, and Fertigung 2/5, 173-177.

Wee, T S and Magazine, M J , 1981a. *An efficient branch and bound algorithm for an assembly line balancing problem - part I Minimize the number of workstations.* Working paper No. 150, University of Waterloo, Waterloo, Ontario, Canada.

Wee, T S and Magazine, M J , 1981b. *An efficient branch and bound algorithm for an assembly line balancing problem - part II. Maximise the production rate.* Working paper No. 15 1, University of Waterloo, Waterloo. Ontario, Canada.

Wee, T S and Magazine, M J, 1982. Assembly line balancing as generalized bin packing. *Operation Research letters,* 1/2,56-58.

White, W W, 1961. Comments on a paper by Bowman. *Operation Research,* 9, 274-276.

Whitley, D, 1989. The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. *Proceedings of the third International Conference on Genetic Algorithms.* Palo Alto, CA. Morgan Kaufmann Publishers, 116-122.

Yano, C A and Bolat, A , 1989. Survey, development and application of algorithms for sequencing paced assembly lines. *Journal of Manufacturing and Operations Management,* 2, 172-198.

APPENDIX A

GLOSSARY

Genetic algorithms are biologically inspired computational models and the terminology has been borrowed from different disciplines including biology, mathematics, and computer science. Some of the terminology used in this report is described below:

Allele: The value of a gene.

Building Block: A group of genes that give a chromosome a high fitness.

Chromosome: An array of parameters or genes that is passed to the fitness function.

Converge: To arrive at the solution. A gene is said to have converged when 95% of the chromosome contain the same allele for that gene. Genetic algorithms are considered converged when they stop finding better solutions for a specified period of operation.

Cost: Output of the cost function

Cost function: Function to be optimised.

Crossover: An operator that forms a new chromosome from two parent chromosomes by combining part of the information from each.

Crossover rate: A number between zero and one that indicates how frequently crossover is applied to a given population.

Darwinism: Theory founded by Charles Darwin that evolution occurs through random variation of heritable characteristics, coupled with natural selection (survival of the fittest).

Elitism: The chromosome with the best cost is kept from generation to generation.

Evolution: A series of genetic changes in which living organisms acquire the characteristics that distinguish it from the other organisms.

Fitness: Opposite of cost. A value associated with a chromosome that assigns a relative merit to that chromosome.

Fitness function: Has the negative output of the cost function. A mathematical subroutine that assigns a value or fitness to a set of parameters.

Gene: A unit of heredity that is transmitted in a chromosome and controls the development of a trait.

Genetic Algorithm: A type of evolutionary computation devised by John Holland. It models the biological genetic process by including crossover and mutation operators.

Genotype: the genetic composition of an organism. The information contained in the genome.

Global Minimum: True minimum of the entire search space.

Hill climbing: Investigates adjacent points in the search space, and moves in the direction giving the greatest increase in fitness. Exploration technique that is good at finding local extrema.

Inversion: A reordering operator that works by selecting two cut points in a chromosome, and reversing the order of all the genes between those two points.

Local minimum: A minimum in a subspace of the search space.

Mating pool: A set of the population selected for potential parents.

Mutation: a reproduction operator that randomly alters the values of genes in a parent chromosome.

Mutation rate: percentage of bits in a population mutated in each iteration of the Genetic Algorithm.

Natural selection: Fit individuals' reproduce, passing their genetic information on to their offspring.

Offspring: an individual generated by any process of reproduction.

Optimisation: the process of iteratively improving the solution to a problem with respect to a specified objective function.

Parallel Genetic algorithm: A genetic algorithm written to run on a parallel-processing computer.

Parent: An individual that produces to generate one or more other individuals, known as offspring, or children.

Phenotype: the environmentally and genetically determined traits of an organism. That trait actually observed.

Population: A group of individuals that interact (breed) together.

Permutation problem: A problem that involves reordering a list

Recombination: Combining the information from two parent chromosomes via crossover.

Reproduction: The creation of offspring from two parents (sexual reproduction) or from a single parent (asexual reproduction)

Search space: All possible values of all parameters under consideration.

Selection: The process of choosing parents for reproduction (usually based on fitness)

Simulation: The act of modelling a process.

Species: A group of organisms that interbreed and are reproductively isolated from all other groups. A subset of the population.

Survival of the fittest: Only the individuals with the highest fitness value survive.

APPENDIX B

GA APPLICATIONS IN ASSEMBLY LINE BALANCING

Table 1: Recent genetic algorithm applications in the assembly line balancing problem

Reference	Initial Population	Selection	Crossover	Mutation	Termination
Ponnambalam et. al (2000)	Randomly generated population	Roulette Wheel	Two point crossover	Classic Mutation	Number of generations
Sabuncuoglu et al (2000)	Feasible random solutions	Roulette Wheel	Dynamic partition crossover technique	Scramble Mutation	Number of generations
Kim et al (1998)	Heuristic population	Tournament	Heuristic Structural Crossover	Heuristic structural mutation	Number of generations
Kim et al (1996)	Feasible solutions + bin packing solutions	Tournament	Order crossover Two point crossover	Classic mutation	Number of generations
Suresh et al (1996)	(Two populations) Feasible solutions with bin packing solutions + infeasible population.	Roulette Wheel	Single point crossover	Classic mutation	Number of generations
Rubinovitz and Levitin (1995)	Singe pass heuristic solutions + bin packing solutions	Rank based selection	Fragment reordering crossover	Classic mutation	Number of generations

Cont…

Reference	Initial Population	Selection	Crossover	Mutation	Termination
Tsujimura et al (1995)	Randomly generated feasible solutions with repair technique	Elitist selection	Partially mapped crossover	Classic mutation	Number of generations
Leu et al (1994)	Feasible solutions + bin packing solutions	Roulette Wheel	Two point crossover	Scramble Mutation	Number of generations
Anderson and Ferris (1994)	Arcus procedure + bin packing solutions	Roulette Wheel	Single point crossover	Classic mutation	Number of generations
Minagawa and Kakazu (1992)	Feasible solutions + bin packing solutions	Roulette Wheel	Single and two point crossover applied alternatively	Point mutation	Number of generations
Falkenauer and Delchambre (1992)	Feasible solutions + bin packing solutions	Roulette Wheel	Two point crossover	Classic mutation	Number of generations
Anderson and Ferris (1990)	Arcus procedure + bin packing solutions	Roulette Wheel	Single point crossover	Classic mutation (Very low mutation rate)	Number of generations

APPENDIX C

PROOF

The novel front-loading fitness function (equation 3. 15) consists of the following properties:

1. Decreasing number of workstations (m) increases the overall fitness.
2. Moving an element from a latter workstation to an earlier workstation increases the overall fitness.

The second property has already been discussed in Chapter3 and the proof of the first property is given here.

Consider the overall fitness function given in equation 3.15,

$$FF = \begin{cases} n^3 l\beta & l < P \\ X + Y + Z & l = P \end{cases} \tag{3.15}$$

Where $X = n^3 l\beta$ $\quad Y = n^2\beta \dfrac{\sum_{j=1}^{K}\left(\dfrac{S_j}{C}\right)}{m - m^* + 1}$ $\quad Z = \alpha \dfrac{\sum_{j=1}^{m}\left(\dfrac{S_j}{C}\right)^k R^{m-j}}{mR^{m+1}}$

$$\max(X) > \max(Y) > \max(Z) \qquad \forall X, Y, Z$$

It can be seen that, component X is independent of the number of workstations (m) and therefore, decreasing or increasing the number of workstations does not effect the fitness through X. However both Y and Z terms consist of variable m and they are sensitive to any change in number of workstations.

Consider component Y in equation 3.14,

$$Y = n^2\beta \frac{\sum_{j=1}^{K}\left(\dfrac{S_j}{C}\right)}{m - m^* + 1} \tag{c.1}$$

Expanding the right had side of equation c.1,

$$Y = \frac{n^2\beta}{m - m^* + 1}\left\{\frac{S_1}{C} + \frac{S_2}{C} + \frac{S_3}{C} + \frac{S_4}{C} + \cdots\cdots + \frac{S_K}{C}\right\}$$

$$Y \le \frac{n^2}{m - m^* + 1}K \qquad \text{since } 1 \ge \left(\frac{S_x}{C}\right) \quad \forall\ S_x \qquad \text{(c.2)}$$

Where K is the workstation index, which varies according to the following relationship, g is the number of generations and G is the number of generations per workstation (test problem constant).

If $g = 1; K = 1$
Else if $g = G*K$; then $K := K+1$

Without loss of generality, in order to illustrate the properties of this relationship, setting $n = 58$, $m = 25$ generates a family of curves (depending on K), and figure (a) shows the fitness variation of Y while the number of workstations decrease from 30 to 25 in the solution.

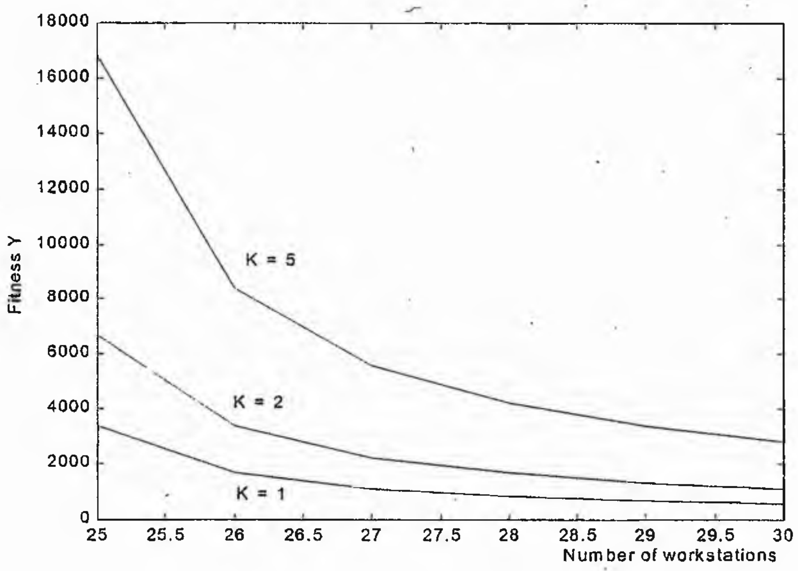


Figure a. Fitness variation against number of workstations for K=1,2 and 5.

From figure (a) it can be seen that, decreasing the number of workstations in the solution increases the fitness *Y* irrespective of the value of *K*.

Next, consider the *Z* component of the overall fitness function that is responsible for forward loading.

$$Z = \alpha \frac{\sum_{j=1}^{m} \left(\frac{S_j}{C} \right)^k R^{m-j}}{mR^{m+1}} \qquad \text{(c.3)}$$

Expanding the right hand side of the above equation c.3, and substituting *k=1*, gives

$$Z = \frac{\alpha}{m} \left\{ \left(\frac{S_1}{C} \right) R^{-2} + \left(\frac{S_2}{C} \right) R^{-3} + \left(\frac{S_3}{C} \right) R^{-4} + \left(\frac{S_4}{C} \right) R^{-5} + \cdots \left(\frac{S_m}{C} \right) R^{-(m+1)} \right\}$$

Since $\left(\frac{S_x}{C} \right) R^{-(x+1)} \le R^{-(x+1)}$, the above expansion can be simplified in to the following expression.

$$Z \le \frac{\alpha}{m} \left\{ \left(\frac{1}{R} \right)^2 + \left(\frac{1}{R} \right)^3 + \left(\frac{1}{R} \right)^4 + \cdots + \left(\frac{1}{R} \right)^{m+1} \right\} \qquad \text{(c.4)}$$

Rearranging the expression c.4,

$$Z \le \frac{\alpha}{m} \left(\frac{1}{R} \right)^2 \left\{ 1 + \left(\frac{1}{R} \right) + \left(\frac{1}{R} \right)^2 + \left(\frac{1}{R} \right)^3 + \left(\frac{1}{R} \right)^4 + \cdots + \left(\frac{1}{R} \right)^{m-1} \right\}$$

The first *m* terms in the above expression represents a geometric series and the summation of the progression is given by (S_{sum})

$$Z \le S_{sum} = \frac{\alpha}{m}\left(\frac{1}{R^2}\right)\left(\frac{1-\left(\frac{1}{R}\right)^{m-1}}{1-\left(\frac{1}{R}\right)}\right)$$

Substituting $\alpha = nCR^{m^*+1}$ and, further simplifying gives,

$$Z \le \left(\frac{nCR^{m^*}}{R-1}\right)\frac{1}{m}\left\{1-\frac{1}{R^{m-1}}\right\} \qquad \text{(c.5)}$$

Without loss of generality, in order to illustrate the properties of this relationship, setting n = 58, m = 25, and C = 65, generates a family of curves (depending on R), and figure (b) displays the fitness variation of Z when the number of workstations decrease from 30 to 25 in the solution for R = 1.2.

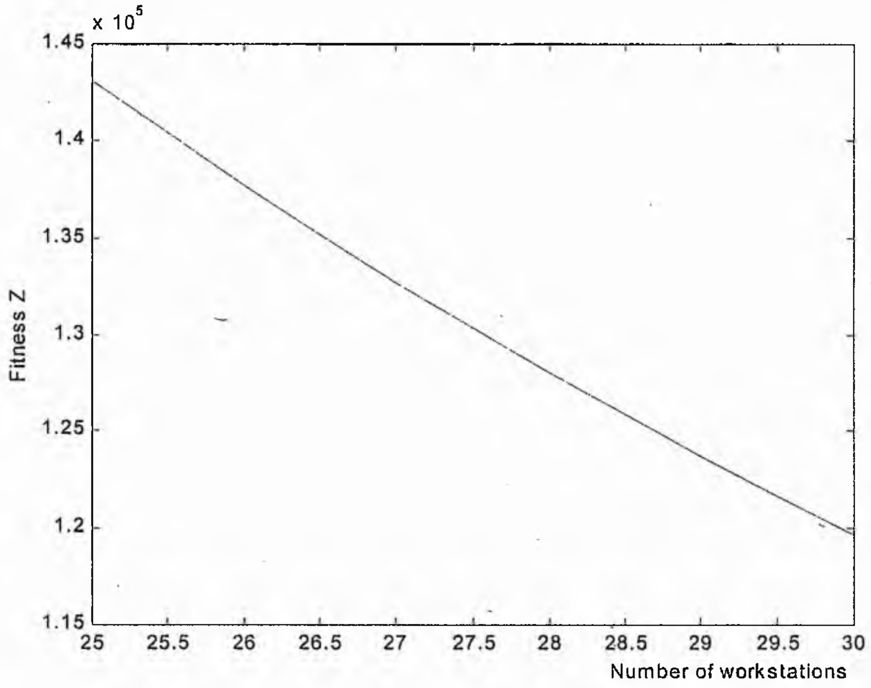
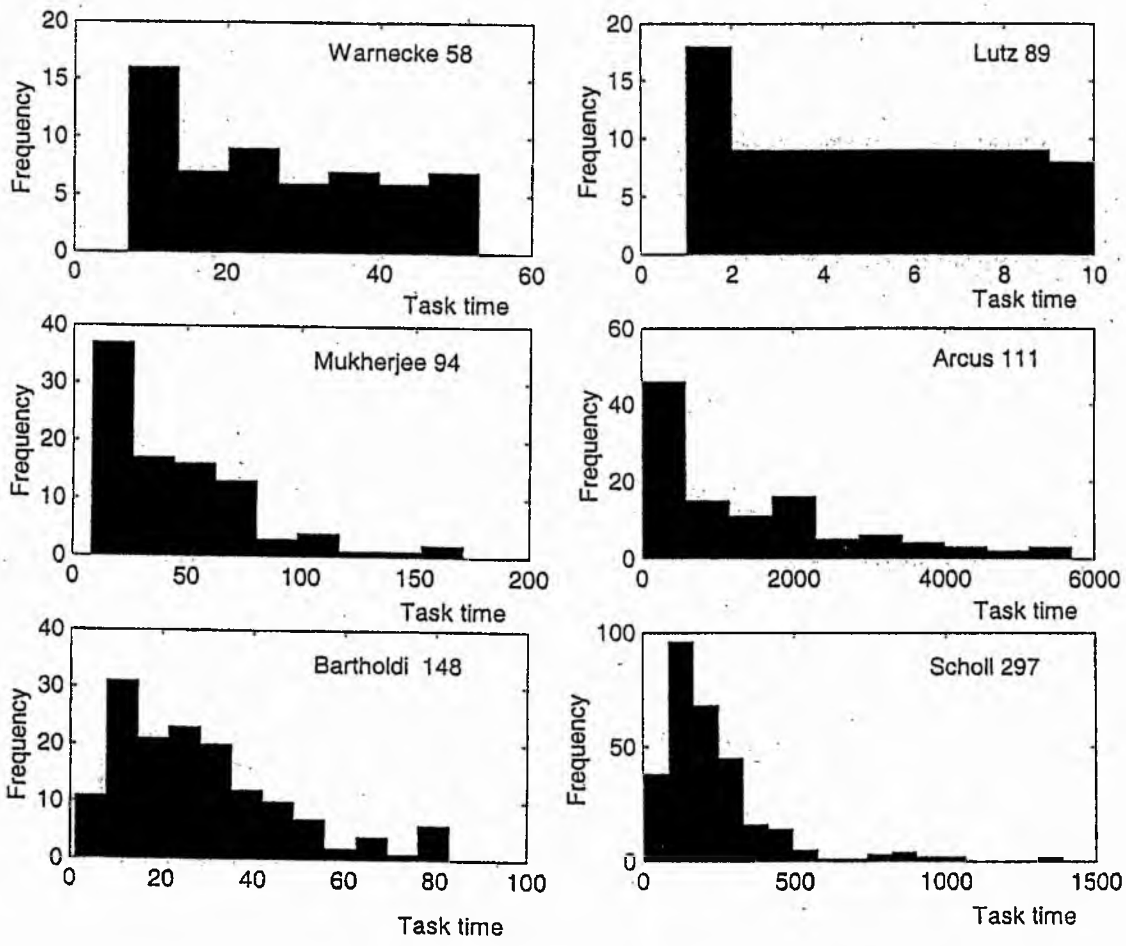


Figure (b). Variation of fitness against number of workstations

So, both Y and Z fitness components increase as the number of workstations decreases , proving the second property of the overall fitness function.

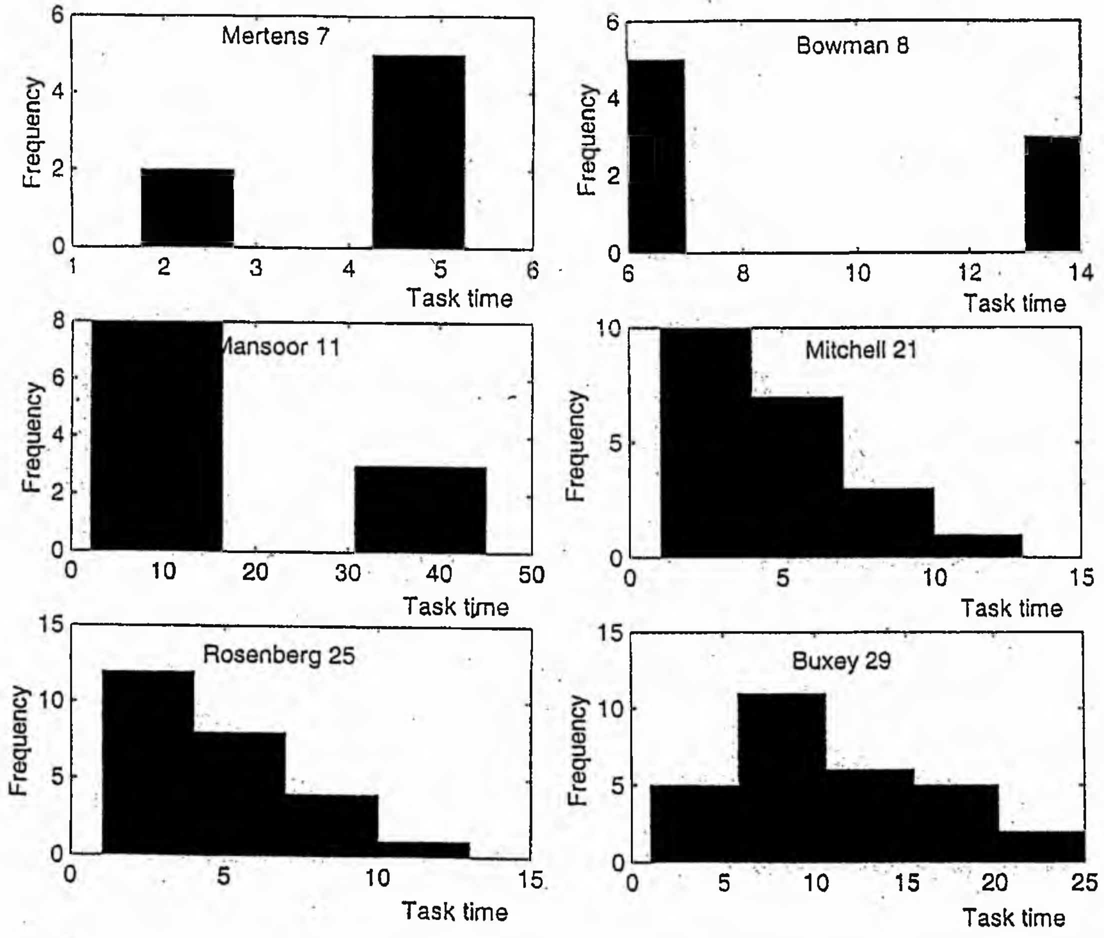
APPENDIX D

TEST PROBLEM SPECIFICATIONS



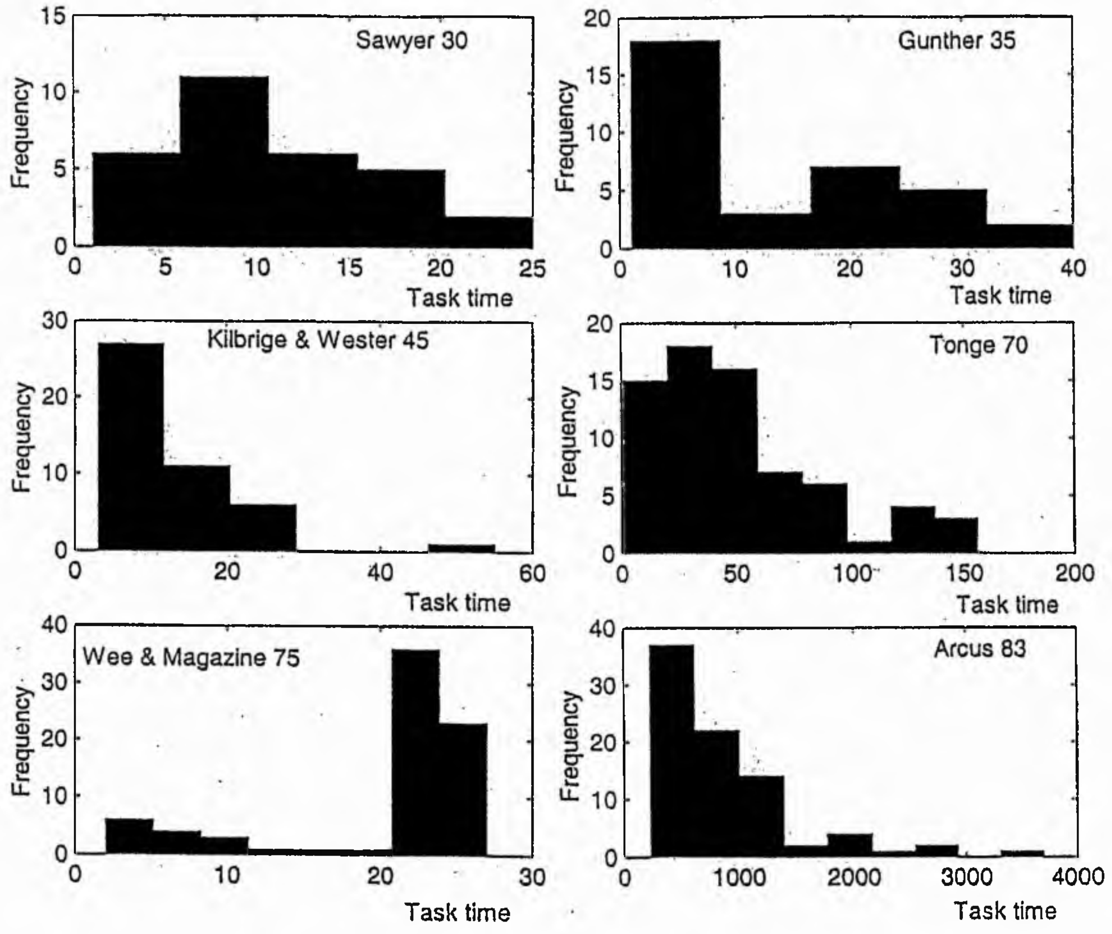
Reference	Problem ID	n	t_{mim}	t_{max}	t_{sum}	OS	TV	Cycle times
Warnecke(1971)	WA_{58}	58	7	53	1548	59.1	7.6	65
Lutz (1974)	LU_{89}	89	1	10	485	77.6	10.0	20
Mukeherjee and Basu (1964)	MU_{94}	94	8	171	4208	44.8	21.4	176,301, 351
Arcus (1963)	AR_{111}	111	10	5689	150339	40.4	568.9	6269
Bartholdi (1993)	BA_{148}	148	1	83	4234	25.8	83.0	170,425, 1000
Sholl (1999)	SH_{297}	297	5	1366	69655	58.2	277.2	2787,7000, 14000

n : number of task elements	t_{min} : minimum task time
t_{max} : maximum task time	t_{sum} :total work content
OS : Order Strength	TV : time variability ratio



Reference	Problem ID	n	t_{mim}	t_{max}	t_{sum}	OS	TV	Cycle times
Mertens (1967)	ME_7	7	1	6	29	52.4	6.0	6,8,10,15
Bowman (1960)	BO_8	8	3	17	75	75.0	5.7	20
Mansoor(1964)	MA_{11}	11	2	45	185	60.0	22.5	48,62,94
Mitchell (1957)	MI_{21}	21	1	13	105	71.0	13.0	15,26
Rosenberg and Ziegler (1992)	RO_{25}	25	1	13	125	71.7	13.0	14,16,21, 32
Buxey (1974)	BU_{29}	29	1	25	324	50.7	25.0	27,33,36, 41

n : number of task elements
t_{min} : minimum task time
t_{max} : maximum task time
t_{sum} :total work content
OS : Order Strength
TV : time variability ratio



Reference	Problem ID	n	t_{mim}	t_{max}	t_{sum}	OS	TV	Cycle times
Sawyer (1970)	SA_{30}	30	1	25	324	44.8	25.0	25,30,41
Gunther et al (1983)	GU_{35}	35	1	40	483	59.5	40.0	41,49, 54, 81
Kilbridge & Wester (1962)	KW_{45}	45	3	55	552	44.6	18.3	69,92, 138,184
Tonge (1961)	TO_{70}	70	1	156	3510	59.4	156	251,320
Wee and Magazine (1981b)	WM_{75}	75	7	53	1548	59.1	7.6	97,104, 350
Arcus (1963b)	AR_{83}	83	233	3691	75707	59.1	15.8	5408 7571

n : number of task elements
t_{max} : maximum task time
OS : Order Strength
t_{min} : minimum task time
t_{sum} :total work content
TV : time variability ratio