HEURISTIC OPTIMIZATION ALGORITHM BASED LINE BALANCING IN A FUZZY ENVIRONMENT

BY

FERDOUS SARWAR

DEPARTMENT OF INDUSTRIAL AND PRODUCTION ENGINEERING
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
DHAKA, BANGLADESH

APRIL 2007

CERTIFICATE OF APPROVAL

The thesis titled "Heuristic Optimization Algorithm Based Line Balancing in a Fuzzy Environment" submitted by Ferdous Sarwar, Roll No: 040508004 P, Session: April, 2005 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of **Master of Science in Industrial and Production Engineering** on April 11, 2007.

Board of Examiners

Dr. M. Ahsan Akhtar Hasin Professor Department of Industrial and Production Engineering BUET, Dhaka.	Chairman (Supervisor)
Dr. A. K. M. Masud Associate Professor Department of Industrial and Production Engineering BUET, Dhaka.	Member
Dr. Abdullahil Azeem Assistant Professor Department of Industrial and Production Engineering BUET, Dhaka.	Member
Dr. Nikhil Ranjan Dhar Professor and Head Department of Industrial and Production Engineering BUET, Dhaka.	Member (Ex-officio)
Dr. Md. Mostofa Akbar Associate Professor Department of Computer Science and Engineering BUET, Dhaka.	Member (External)

Department of Industrial and Production Engineering
Bangladesh University of Engineering and Technology

April, 2007

CANDIDATE'S DECLARATION

It is hereby declared that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

16/04/07

Ferdous Sarwar

To the Almighty

To my family

ACKNOWLEDGEMENT

All praises to Allah, the most benevolent and the Almighty, for His boundless grace in successful completion of this thesis.

I would like to express my sincere respect and gratitude to my thesis supervisor, Dr. M. Ahsan Akhtar Hasin, Professor of the Department of Industrial and Production Engineering (IPE), Bangladesh University of Engineering and Technology (BUET), Dhaka, for his thoughtful suggestions, constant guidance and encouragement throughout the progress of this research work.

I also express my profound thanks and gratitude to Dr. Nikhil Ranjan Dhar, Professor and Head, Department of IPE, BUET, Dr. A. K. M. Masud, Associate Professor, Department of IPE, BUET, Dr. Abdullahil Azeem, Assistant Professor, Department of IPE, BUET and Dr. Md. Mostofa Akbar, Associate Professor, Department of CSE, BUET for their support and kind interest in this research.

I am grateful to my younger brother Mr. Shahriar Rouf, undergraduate student, CSE, BUET for helping me out with computer coding.

I am very much thankful to Engr. Faruque-ul-Islam, Executive, Planning & Co-ordination, RahimAfrooz Batteries Limited (RBL) and Ms Alina Mozumder, Sr. Officer, Planning & Co-ordination, RBL, for their cordial encouragement and sincere help during the data collection phase in case study.

I am grateful to all of my colleagues of the Department of Industrial and Production Engineering (IPE), Bangladesh University of Engineering and Technology (BUET), Dhaka for their cooperation and motivation during the study.

Finally, I would like to extend my sincere thanks to my parents and beloved wife whose continuous inspiration, sacrifice and support encouraged me to complete the thesis successfully.

ABSTRACT

Assembly Line Balancing (ALB) is one of the important problems of production management. As small improvements in the performance of the system can lead to significant monetary consequences, it is of utmost importance to develop practical solution procedures that yield high-quality design decisions with minimal computational requirements. Due to the NP-hard nature of the ALB problem, heuristics are generally used to solve real life problems. The constraints and parameters of fuzzy nature exist in line balancing problem. Fuzzy optimization can be implemented effectively in solving ASLBP. Fuzzy sets or fuzzy numbers can appropriately represent imprecise parameters, and can be manipulated through different operations on fuzzy sets or fuzzy numbers. Since imprecise parameters are treated as imprecise values instead of precise ones, the process will be more powerful and its results more credible. An efficient heuristic to solve the fuzzy single-model ALB problem has been presented in this research work. The proposed heuristic is a Genetic Algorithm (GA) with a special chromosome structure that is efficient to handle fuzzy job time through the evolution process. Elitism is also implemented in the model by using fitness function value. In this context, the proposed approach can be viewed as a unified framework which combines several new concepts of GA in the algorithmic design.

TABLE OF CONTENTS

Acknowledgement			vi
Abstract			vii
Table of Contents			viii
List of Tables			xiii
List of Figures			xiv
List of Abbreviations			xv
List of Symbols			xvi
CHAPTER 1		**INTRODUCTION**	1-2
1.1		General Introduction	1
1.2		Rationale of the Study	1
1.3		Objective of the Study	2
1.4		Methodology	2
CHAPTER 2		**LITERATURE REVIEW**	3-17
2.1		Introduction	3
2.2		Simple Assembly Line Balancing Problem (SALBP)	3
2.3		Heuristic Methods in Line Balancing	4
2.3.1		Constructive Procedures	4
	2.3.1.1	Priority Rule Based Procedure for SALBP-1	5
	2.3.1.2	Single Pass Decision Rules	7
	2.3.1.3	Arcus' Biased Sampling Procedure (Arcus)	8
2.3.2		Backtracking Decision Rules	8
	2.3.2.1	Hoffman's Enumeration Procedure	9
	2.3.2.2	Hoffman's Modified Enumeration Procedure	9
	2.3.2.3	Dar-El's Line Balancing Heuristic	10
2.3.3		Optimal Decision Seeking Rules	10
	2.3.3.1	Branch and Bound Methods	10
	2.3.3.2	Integer Programming	10
	2.3.3.3	Dynamic Programming	11

		2.3.3.4	Multiple Solutions Technique (MUST)	11
	2.3.4		Local Search and Metastrategies	11
		2.3.4.1	Tabu Search for SALBP-1	12
		2.3.4.2	Simulated Annealing (SA) Procedures	12
		2.3.4.3	Ant-Colony Algorithm Approach	13
		2.3.4.4	Genetic Algorithm	13
	2.3.5		Fuzzy Models	15
	2.3.6		Cost and Profit-Oriented Objectives	14
		2.3.6.1	Cost-Oriented Models	16
		2.3.6.2	Profit-Oriented Models	17
CHAPTER 3			**FUZZY LOGIC**	18-38
3.1			Introduction	18
3.2			Crisp Sets and Characteristic Functions	19
	3.2.1		Characteristic Functions	20
3.3			Fuzzy Set Theory	20
	3.3.1		Definition	20
	3.3.2		Fuzzy Subsets	21
	3.3.3		Fuzzy Sets Vs. Crisp Sets	23
	3.3.4		Fuzzy Sets and Membership Functions	24
	3.3.5		Characteristic Functions and Membership Function: A Comparison	25
	3.3.6		The Notation of Fuzzy Sets	26
3.4			Fuzzy Sets: Basic Concepts	28
3.5			Fundamental Operations of Fuzzy Sets	28
	3.5.1		Union of Fuzzy set A and B	29
	3.5.2		Intersection of Fuzzy set A and B	29
	3.5.3		Complement of Fuzzy set A and B	29
	3.5.4		The Law of Excluded Middle	29
	3.5.5		The Law of Contradiction	29
	3.5.6		Equality and Inclusion of Fuzzy set	30

3.8			Decomposition Principle	31
3.9			Fuzzy Numbers and Extension Principle	32
3.10			Fuzzy Numbers	33
3.11			Fuzzy Interval	33
3.12			Arithmetic Operations of Fuzzy Numbers	33
3.13			Application of Fuzzy Logic	34
3.14			Fuzzy Ranking Methods	35
	3.14.1		Signed Distance Ranking of The Level λ Fuzzy Numbers	36
CHAPTER 4			**GENETIC ALGORITHM**	39-52
4.1			Introduction	39
4.2			History	39
4.3			GA Procedure	40
4.4			Pseudo-Code Algorithm	41
4.5			Initialization	41
4.6			Encoding a Chromosome	42
	4.6.1		Binary Encoding	42
	4.6.2		Permutation Encoding	42
	4.6.3		Value Encoding	43
	4.6.4		Tree Encoding	43
4.7			Crossover	43
	4.7.1		Crossover Techniques	43
		4.7.1.1	One Point Crossover:	44
		4.7.1.2	Two Point Crossover	44
		4.7.1.3	Cut and Splice	45
		4.7.1.4	Uniform Crossover and Half Uniform Crossover	45
	4.7.2		Crossover for Ordered Chromosomes	45
	4.7.3		Crossover Biases	46

	4.7.3.1	Partially Matching Crossover (PMX)	46
	4.7.3.2	Order Crossover (OX)	47
	4.7.3.3	Cycle Crossover (CX)	48
4.8		Fitness Function	49
4.8.1		Fitness Scaling	49
4.9		Selection	50
4.10		Reproduction	50
4.11		Termination	51
4.12		Variant	52
CHAPTER 5		**PROBLEM FORMULATION**	53-61
5.1		Introduction	53
5.2		Mathematical Model	55
5.3		Genetic Algorithm Searching Method	57
5.4		Standard Encoding	58
5.5		Initial Population	59
5.6		Crossover Operator	59
5.7		Mutation Operator	60
5.8		Selection Method	60
5.9		Fitness Functions	61
5.10		Stopping Condition	61
CHAPTER 6		**RESULT ANALYSIS**	62-71
6.1		Computational Result	62
6.2		Benchmarking with EUREKA	64
6.3		Benchmarking with SALOME	65
6.4		Benchmarking with Hybrid GA	66
6.5		Fuzzy ALBP	67
6.6		Comparison of Eight Methods	68
6.7		Convergence Analysis	69

CHAPTER 7	**CASE STUDY**	72-77
7.1	Introduction	72
7.2	Overview of Problem Area	73
7.3	Some Important Definitions	74
7.4	Specific Problem Area: Assembly line in RBL-2	75
7.4.1	Assembly Operations	75
7.5	Result	77
	CONCLUSION AND RECOMMENDATION	78
8.1	Conclusion	78
8.2	Recommendation	78
References		79
Appendices		
	Appendix A: Benchmark data with EUREKA	88
	Appendix B: Benchmark data with SALOME	90
	Appendix C: Benchmark data with Hybrid GA Algorithm	99
	Appendix D: Program Code	101

LIST OF TABLES

Table No	Title	Page No
2.1	Types of SALBP	4
2.2	Priority Rules	5
3.1	Degree of OVERWEIGHT and MODERATE Height	24
3.2	The value of characteristic functions in crisp sets	25
3.3	Value of membership functions in fuzzy sets	25
6.1	Problem Characteristics	63
6.2	Benchmarking with EUREKA	64
6.3	Benchmarking with SALOME	65
6.4	Benchmarking with Hybrid GA	66
6.5	Fuzzy Task Time	67
6.6	Comparison of Eight Methods on the 70 Task Problems	68
7.1	Precedence Graph	76
7.2	Production Quantity of Battery	76
7.3	Fuzzy Task Time	77

LIST OF FIGURES

Figure No	Title	Page No
2.1	Difference between Crisp and Fuzzy Sets	21
2.2	Degree of Tallness	22
3.2	Crisp Sets of Height	25
3.3	Fuzzy Sets of Height	26
3.4	α-cut and Decomposition Principle	31
3.5	Example of Extension Principle	32
3.6	Level λ Fuzzy Set	37
4.1	One Point Crossover	44
4.2	Two Point Crossover	44
4.3	Cut and Splice	45

LIST OF ABBREVIATION

Biased Probabilistic Choice (BPC)

Branch and Bound (B & B)

Fitness Function Value (FFV)

Genetic Algorithm (GA)

Most Immediate First (MIF)

Membership Function (MF)

Nondeterministic Polynomial-time Hard (NP-Hard)

Order Strength (OS)

Random Choice (RC)

Simple Assembly Line Balancing Problem (SALBP)

Simulated Annealing (SA)

Time Variation (TV)

Triangular Fuzzy Number (TFN)

Traveling Salesman Problem (TSP)

LIST OF SYMBOLS

Membership Grade of *X* In Fuzzy Set *A*	$\mu_A(x)$
Universal Quantifier	$(\forall)$
Alpha Cut of Fuzzy Set *A*	$({}^{\alpha}A)$
Supremum	(sup)
Set of all Nonnegative Real Numbers	(R^{+})
Characteristic Functions of Crisp Set A	(χ_A)
Strong Alpha Cut	$({}^{\alpha+}A)$
Level Set of Fuzzy Set *A*	$\wedge(A)$

CHAPTER 1

INTRODUCTION

1.1 General Introduction

An assembly line is a flow-oriented production system where the productive units performing the operations, referred to as stations, are aligned in a serial manner. Line balancing problem deals with the assignment of tasks to workstations. The assembly line includes a series of workstations, where product items are processed. To produce a product, it is required to process a set of tasks (jobs). These tasks must follow a given processing order called precedence relationship. The assembly line could be dedicated to produce for a single product model or multiple product models. The most common line balancing problem is Single-model assembly line balancing problem with deterministic/stochastic/fuzzy processing time (SALBP).

1.2 Rationale of the Study

In a typical line balancing problems, the requirement is often to distribute the tasks to workstations such that a certain objective (number of workstations, total cost, production rate, etc.) is optimized and precedence relationship is not violated. The workstation time, which is the sum of times of all tasks assigned to that workstation, must not exceed the given cycle time. The processing time of tasks are also given. In general, the line balancing problem has several variances. The variety could come from the requirement, objective, or the form of processing time or the structure of the lines. The requirement of the problem is not only to allocate tasks to workstations but also to sequence product models to be assembled in the designing batch/mixed-model lines or determine optimal batch sizes for batch-model configuration. The objective could be other criteria deferent than number of workstations such as minimization of cycle time for a given number of workstations, minimization of balance delay time, etc. A deference between the cycle time and workstation time is called idle time The sum of idle time for all workstations is called balance delay time. These objectives could also be taken into account simultaneously in the form of multi-criteria optimization problem. The processing time could be given in deterministic terms or from the stochastic processes or in the form of vagueness of fuzzy sets.

1.3 Objectives of the Study

The objectives of this research can be outlined as follows –

1. Analyze the existing assembly line balancing process (ASLBP-1)
2. Address uncertainty by incorporating fuzzy task times and fuzzy constraints
3. Develop a heuristic optimization algorithm for ASLBP-1 for single line.
4. Benchmark the proposed algorithm with an existing algorithm

1.4 Outline of Methodology

In order to carry out this study, steps that have been adopted are mentioned below:–

i. Analyze the existing assembly line balancing process (ASLBP-1)

ii. Develop a heuristic optimization algorithm for ASLBP-1 for single line

iii. Benchmark the proposed algorithm with three existing algorithm

iv. Data collection:

   To implement the developed algorithm, data was collected from RahimAfrooz Batteries Ltd at Ziranibazar, Gazipur. The required data type includes-

   - Number of work stations that all battery variants under consideration pass.
   - Number of identical machines available in each stage.
   - Approximate fuzzy processing time required for each variant of batteries at each stage.
   - Approximate fuzzy cycle time for each type of battery.

v. Finally, analyzing the performance of the developed algorithm by running the computer program for different parameters.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The line balancing problem is one of the most traditional problems which has evolved from the concept of division of labor, and became popular because of Henry Ford's famous "T-model". Despite its long history of development, line balancing study is still an attractive research topic today due to its relevancy to the everyday manufacturing and the diversity of system configurations. According to different system configurations, assembly line can be classified as single-model line, mixed-model line, and multi-model line. Single-model line only assembles one product, while multiple products are assembled in either mixed or multi-model line, but intermediate set-up is required in the latter case. In addition to serial line assembly, flexibility can be improved by the introduction of parallelism, including parallel lines, parallel stations, and parallel tasks.

2.2 Simple Assembly Line Balancing Problem (SALBP)

Most of the research in assembly line balancing has been devoted to modeling and solving the simple assembly line balancing problem (SALBP). This classical single-model problem contains the following main characteristics [1, 2]:

- mass-production of one homogeneous product;
- given production process;
- paced line with fixed cycle time c;
- deterministic (and integral) operation times tj;
- no assignment restrictions besides the precedence constraints,
- serial line layout with m stations;
- all stations are equally equipped with respect to machines and workers;
- maximize the line efficiency $E = t_{sum} / (m \times c)$ with total job time t_{sum}

Table 2.1: Types of SALBP

No. of stations (m)	Cycle Time, c	
	Given	Minimize
Given	SALBP-F	SALBP-2
Minimize	SALBP-1	SALBP-E

Several problem versions arise from varying the objective as shown in Table 2.1. SALBP-F is a feasibility problem which is to establish whether or not a feasible line balance exists for a given combination of m and c. SALBP-1 and SALBP-2 have a dual relationship, because the first minimizes m given a fixed c, while the second minimizes c (maximizes the production rate) given m. SALBP-E is the most general problem version maximizing the line efficiency thereby simultaneously minimizing c and m considering their interrelationship.

2.3 Heuristic Methods in Line Balancing

A large variety of heuristic approaches to different versions of SALBP have been proposed in the last decades. While constructive procedures constructing one or more feasible solution(s) were developed until the mid nineties, improvement procedures using metastrategies like tabu search and genetic algorithms have been in the focus of researchers in the last decade.

2.3.1 Constructive Procedures

The majority of constructive procedures has been proposed for SALBP-1 and is based on priority rules, others are restricted enumerative procedures. The most recent comprehensive surveys of those approaches are given by Talbot et al. [3] and Scholl [2].Furthermore, Boctor [4] and Ponnambalam et al. [5] also presented a detail survey on the constructive procedures.

2.3.1.1 Priority Rule Based Procedure for SALBP-1

Those procedures use priority values computed for the different tasks based on the task times and the precedence relations given. Some of the most effective ones are given in Table 2.1 [3, 6, 7]. In any case, the tasks are sorted according to non-increasing priority values to get a priority list.

Table 2.2: Priority Rules

Name	*Priority values*
MaxT	Tasktime t_j
MaxPW	Positional weight
MaxF	Number of followers j
MaxTL	Tasktime over slack
MaxTS	Cumulated positional weight
MaxCPW	Tasktime over latest station

By analogy with exact solution procedures, two construction schemes are relevant for priority rule based approaches. They differ with respect to the manner in which the tasks to be assigned are selected out of the set of available tasks [7].

- **Station-Oriented Procedures:** They start with the first station (k = 1). The following stations are considered successively. In each iteration, a task with highest priority which is assignable to the current station k is selected and assigned. When station k is loaded maximally, it is closed, and the next station k + 1 is opened. For the rule MaxPW, this procedure is called ranked positional weight technique by Helgeson and Birnie [8].

- **Task-Oriented Procedures:** Among all available tasks, one with highest priority is chosen and assigned to the earliest station to which it is assignable. Depending on whether the set of available tasks is updated immediately after assigning a task or after assigning all currently available tasks, task-oriented methods can be subdivided into immediate-update-first and general-first-fit methods [6, 9].

Theoretical analyses show that both schemes obtain the same solution when the used priority rule is strongly monotonous, i.e., the priority value of any task j is smaller than that of each predecessor h ∈ Pj. This is, e.g., true for MaxPW, MaxF, and MaxCPW [2]. Computational experiments indicate that, in general, station- oriented procedures get better results than task-oriented ones though no theoretical dominance exists [7]. These classical priority rules based procedures work unidirectionally in forward direction and construct a single feasible solution. Improvements are obtained by following approaches:

- **Flexible Bidirectional Construction**

The stations to be loaded are considered in forward and backward direction, simultaneously [7]. That is, a station-oriented procedure considers the earliest and the latest unloaded station at a time. Besides selecting a (forward or backward assignable) task by some priority rule the (earliest or latest) station to be considered next is chosen. Task-oriented procedures simultaneously consider forward and backward available tasks and always choose the one with highest priority. Both approaches require defining reversed priority rules [2]. Dynamic priority rules iteratively adapt the priorities depending on the current partial solutions [4, 7] For example, MaxTS can be applied dynamically (in a uni/bidirectional procedure) by modifying the earliest and latest stations according to the assignments made. Multi-pass heuristics repeatedly apply different or stochastic priority rules in order to find several solutions the best of which is taken [3, 10, 11, 12].

- **Flexible Rule Application**

Such procedures try to identify priority rules best suited to solving a certain problem instance. This is done randomly, on the basis of experiences with former rule applications and by exploiting problem structures [13, 14, 15].

- **Reduction Techniques**

Baybars [16] proposes a priority based procedure which involves heuristically reducing the problem size by some logical tests. Furthermore, Tonge [17], Freeman and Swain [18], and Fleszar and Hindi [19] proposed some alternative reduction techniques.

- **Combined Solutions**

SALBP-1 can be interpreted as a shortest path problem with exponential numbers of nodes and arcs. Each feasible solution can be represented by a path in such a graph. Therefore, Pinto et al. [20] describe a two-stage solution approach. In the first step, a number of feasible solutions are determined by a multi-pass heuristic. These solutions are used to construct a subgraph of the complete graph in the second step of the procedure. For this subgraph, a shortest path problem is solved. That is, the outlined approach tries to combine parts of several feasible solutions in order to obtain an improved complete solution.

2.3.1.2 Single Pass Decision Rules

In this category, 13 single pass, single attribute , priority dispatch scheduling rules such as maximum ranked positional weight technique [8], maximum number of immediate followers [21], maximum task time first [22] etc. are included. Each of the decision rules consists of a simple, computationally efficient, list-processing procedure that assigns tasks to work stations according to a task's computed priority.

Operationally, in the implementation of each of these procedure, a task is first assigned a numerical priority specifies by the logic of the heuristic decision rule. Then tasks that are both precedence and cycle time feasible are placed on an available list. The task on the available list with the highest priority is assigned first. The available list is updated to reflect the possible addition of task that are now precedence feasible, and the amount of time available to be assigned to tasks in the work station is reduced by the task time of the assigned task. This process continues for a station until no more tasks can be assigned to it. The assignment process then continues to the next station, and so on, until all tasks have been assigned to some work station. When the final task has been assigned, a complete balance has been obtained.

2.3.1.3 Arcus' Biased Sampling Procedure (ARCUS)

The Arcus procedure uses a biased sampling approach to generate feasible sequence of tasks for assignment to a workstation [23]. A fit list, consisting of those tasks can be assigned to a work station is constructed and weights are assigned to each task. Tasks so assigned are removed from the fit list, and a new fit list, consisting of the tasks which can currently be assigned to a work station is constructed. The process continues until all tasks have been assigned to some work station.

2.3.2 Backtracking Decision Rules

Most of the Simple Assembly Line Balancing (SALB) techniques, which consider only cycle time and precedence constraints, are modified to accommodate the various practical constraints and converted to GALB techniques. The heuristic decision rules are list processing procedures that consider a single attribute of each work task for assignment to work station.

2.3.2.1 Hoffman's Enumeration Procedure

Hoffman [24] used a special zero-one matrix and index vector to implement the enumeration process, which results in a very simple computer code. Starting with station one, a precedent feasible list of tasks is maintained from which the combination of tasks which will minimize station idle time is found via complete enumeration. These tasks are assigned to station one. The process continues with station number two using an updated precedent feasible list. The procedure works unidirectionally in a station-oriented manner. In each iteration, a load with minimal idle time is generated for station k. That is, a single branch of a station-oriented B&B procedure is constructed. Nevertheless, it may require considerable computation times, because it has to examine all possible station loads of a current sub problem.

2.3.2.2 Hoffman's Modified Enumeration Procedure

The original Hoffman approach considers stations in numerical orders. It has a tendency to concentrate idle time in the later stations. In order to overcome the difficulty, Gerhlin and Patterson [25] proposed a very slight modification to the original Hoffman procedure: instead of determine the minimum idle time solution at each work station, determine one that is 'acceptably' close to minimum. The modification accepts a load for the currently considered station if a certain amount of idle time is not exceeded. The accepted portion of idle time depends on the balance delay time (total available idle time) for the theoretical minimum number LM1 of stations and can be controlled by a parameter. An extension of the Hoffmann heuristic which works bidirectionally is proposed by Fleszar and Hindi [19]. This heuristic is combined with a number of bound arguments and reduction techniques and, thus, has become one of the most effective of the available heuristics for SALBP-1.

2.3.2.3 Dar-El's Line Balancing Heuristic

Dar-EL developed MALB [26] as a heuristic variant of his earlier optimal seeking iterative procedure [27]. His optimal seeking procedure is based upon the Rank Positional Weight Heuristic method of Helgeson and Birnie [8], enhanced with a backtracking algorithm that generates all feasible sequences of task assignments.

2.3.3 Optimal - Seeking Decision Rules

Optimal seeking decision rules dominate the heuristic ones if they are given enough computation time. But in case of a computational time constraint, Optimal- seeking decision rules do not perform as well as more sophisticated heuristic procedures.

2.3.3.1 Branch and Bound Methods

Magazine and Wee [28] produced excellent results for the type 1 line balancing problem with their branch and bound procedure. With their method, each node in the solution tree corresponds to a feasible set of tasks assignment to a particular work station, where all nodes at the same depth in the tree refer to the same station number. Starting with node zero, descendent nodes from a node of depth (d) are generated, which are maximal feasible assignments of tasks to station (d+1).

2.3.3.2 Integer Programming

Talbot and Patterson [29] presented integer programming approach. The basic algorithm is a depth first, implicit enumeration, backtracking procedure to which various search, fathoming and backtracking decision rules are applied. They used two variations. The first variation included contains network cuts, where search and backtracking are controlled with the heuristic decision rule. The second variation dos not use any cut associated fathoming rules.

2.3.3.3 Dynamic Programming

Schrage and Baker [30] have proposed an efficient method for implementing the dynamic programming approach of Held et al. [31] through improved procedure for generating feasible subsets, and for labeling. Magazine and Wee [78] programmed and tested the Schrage and Baker approach for solving the Type 1 line balancing problem. Magazine and Wee concluded that their branch and bound solution procedure is preferred to dynamic programming for solving these types of line balancing problems, both with regard to computation time and computer storage required.

2.3.3.4 Multiple Solutions Technique (MUST)

Dar-El and Rubinovitch [32] proposed MUST, a multiple solution technique, which employs exhaustive enumeration to generate all solutions, or some subset of them, for solving the type 2 line balancing problem. As a result of experiments by Dar-el and Rubinovitch [32], it was demonstrated that MUST dominates MALB.

2.3.4 Local Search and Metastrategies

Local search (or improvement) procedures try to improve a given feasible solution by iteratively transforming it into other feasible solutions. Such transformations are referred to as moves. Solutions which may be obtained from a given solution S by means of a single move are called neighboring solutions or neighborhood of S. Traditionally, local search heuristics try to find a sequence of moves which produces a trajectory of successively improved solutions and terminate in a local optimum which might be far from optimality. This difficulty is overcome by modern metastrategies like tabu search [33] and simulated annealing [34].

2.3.4.1 Tabu Search for SALBP-1

Developing a TS procedure for SALBP-1 is not very straightforward. This is due to the fact that only three situations can occur after a move: (1) the number m of stations is unchanged (swapping two real tasks), (2) an additional station m is required (shifting a task in an empty station), (3) one station is empty (shifting the only task in this station to another one). No problem arises in case (3). However, in most iterations a large number of (1) or (2) moves have to be evaluated which have only two different objective function values m and m + 1. In this situation finding a promising search direction is rather complicated. Scholl and Voß [7] concluded that applying their TS procedure for SALBP-2 within a lower bound search (called dual strategy) is the best way out of this dilemma. Chiang [35] proposes a TS procedure similar to the SALBP-2 approach of Scholl and Voß [7] but uses a surrogate objective function that maximizes the sum over the squared station times. While minimizing the number of stations, it additionally favors solutions containing some heavily loaded stations to those solutions having more smoothly loaded ones. This effect successively directs the search to solutions where saving a station in a single move is probable. Computational experiments indicate that both approaches are successful on principle. However, Chiang [35] reports only limited results for the simplest data set on hand which are not very meaningful. Scholl and Voß find out that their dual strategy is competitive to exact procedures (applied as a restricted enumeration) in case of short computation times but is not superior to SALOME-1 in finding good feasible solutions quickly. In opposite to SALBP-2, the quality of the initial solution seems to be important for the quality of the best solution found. A further TS procedure for SALBP-1 is proposed by Lapierre et al. [36] and tested on an arbitrary subset of the test problems available. For these instances it compares favorably with Chiang's approach.

2.3.4.2 Simulated annealing (SA) procedures

Heinrici [37] proposes an SA procedure for SALBP-2 which is based on shifts and swaps. An SA approach for a stochastic variant of SALBP-1 is proposed by Suresh and Sahu [38]. McMullen and Frazier [39] propose a SA procedure for a generalization of SALBP-1 with respect to parallel stations, stochastic task times and alternative objectives.

2.3.4.3 Ant-Colony Algorithm Approach

Bautista and Pereira [40] present an ant algorithm for SALBP-1 which is based on priority rule based procedures. McMullen and Tarasewich [41] propose an ant algorithm for a generalization of SALBP with respect to parallel stations, stochastic task times, multiple objectives and mixed-model production.

2.3.4.4 Genetic Algorithm

GAs are adaptive methods which can be used to solve optimization problems. They are based on genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and survival of the fittest. In nature, individuals with the highest survival rate have relatively a large number of offsprings; that is, the genes from the highly adapted or fit individuals spread to an increasing number of individuals in each successive generation. The strong characteristics from different ancestors can sometimes produce super-fit offspring, whose fitness is greater than that of either parent. In this way, species evolve to become more adapted to their environment. Holland [42] showed that a computer simulation of this process of natural adaptation could be employed for solving optimization problems. Goldberg [43] presented a number of applications of GAs to search, optimization and machine learning problems.

In general, the power of GA comes from the fact that the technique is robust, and can deal with a wide range of problem areas. Although GA is not guaranteed to find the optimal solution, it generally finds good solutions with reasonable computational requirements. To the best of our knowledge, there are only three published papers in literature which solve ALB problem using GA; two of them work on the deterministic (SMD) problem and the other works on the stochastic problem (SMS).

The first attempt was made by Leu et al. [44]. In this study, the authors use solutions of heuristic procedures in the initial population. They also demonstrate the possibility of balancing assembly lines with multiple criteria and side constraints such as, allocating a task in a station by itself. According to the authors, the GA approach has two advantages: (i) GAs search a population rather than a single point and thus increases the odds that the algorithm will not be trapped in a local optimum since many solutions are considered concurrently, and (ii) GA fitness functions may take any form (i.e., unlike gradient methods that have differentiable evaluation functions) and several fitness functions can be utilized simultaneously.

In the second study, Anderson and Ferris [45] showed the effective use of GAs in the solution of combinatorial optimization problems, working specifically on the ALB problem. The authors first describe a fairly standard implementation for the ALB problem. Then an alternative parallel version of the algorithm for use on a message passing system is introduced. Their aim is not to demonstrate the superiority of a GA over the traditional methods, but rather to give some indications for the potential use of this technique in combinatorial optimization problems.

Thus, the authors do not compare the GA with well known heuristics, but only with a neighborhood search scheme with multiple restarts in which the GA is found to be better than this method. Suresh et al. [46] used a GA to solve the SMS version of the ALB problem. The ability of GAs to consider a variety of objective functions is regarded as the major feature of GAs. A modified GA working with two populations, one of which allows infeasible solutions, and exchange of specimens at regular intervals is proposed for handling irregular search spaces, i.e., the infeasibility problem due to precedence relations. The authors claim that a population of feasible solutions would lead to a fragmented search space, thus increasing probability of getting trapped in local minima. They also state that infeasible solutions can be allowed in the population only if genetic operators can lead to feasible solutions from an infeasible population. Since a purely infeasible population may not lead to a feasible solution in this particular problem, two alternative populations, one purely feasible and one allowing a fixed percentage of infeasible chromosomes, are combined in a controlled pool to facilitate the advantages of both of them. Certain chromosomes are exchanged at regular intervals between the two populations; the exchanged chromosomes have the same rank of fitness value in their own populations. The results of the experiments indicate that the GA working with two populations gives better results than the GA with one feasible population.

2.3.5 Fuzzy Models

In the classical mathematical formulation of SALBP, the relevant data are considered deterministic. But, the data of the real world problems are imprecise, vague or uncertain, and then the input data can be only estimated as within uncertainty and this uncertainty may be represented by a fuzzy number. So the problem can be solved by Fuzzy logic. The Fuzzy logic, which was introduced by Zadeh [47], has been applied to various industrial problems including production systems. The concept of fuzzy numbers is introduced to treat imprecise data, such as the processing time of each task. As long as the studies have been made in line balancing, in most cases the processing time was considered deterministic. But objective and constraints are known imprecisely in much of the real world line balancing problems and in such a situation fuzzy set theory becomes effectively functional.

Fuzzy optimization can be implemented effectively in solving ASLBP. Fuzzy sets or fuzzy numbers can appropriately represent imprecise parameters, and can be manipulated through different operations on fuzzy sets or fuzzy numbers. Since imprecise parameters are treated as imprecise values instead of precise ones, the process will be more powerful and its results more credible. Hence, SALBP-2 with fuzzy task times is considered by Tsujimura et al. [48].

2.3.6 Cost- and profit-oriented objectives

The installation of an assembly line requires large (long-term) **capital investments.** Furthermore, operating the line causes short-term **operating costs** such as wages, material, set-up, inventory and incompletion costs [2]. In case of a non-fixed production rate and different levels of production quality, these costs have to be contrasted with the **profit** attained by the line.

The installation and operating costs as well as the profits mainly depend on the cycle time and the number of stations [49] such that cost-oriented models are strongly related to SALBP-E. The latter problem is usually solved by iterating on SALBP-1 or SALBP-2 instances, respectively [50].Thus, the same procedures can likewise be used for cost or profit oriented objectives on principle. However, in some situations it is necessary to consider models which incorporate costs and/or profits explicitly. This is especially true when the balancing problem is connected with the decision problem of selecting processing or equipment alternatives.

2.3.6.1 Cost-oriented models

Rosenberg and Ziegler [51] assume that the operation of a station k causes a *wage rate* w_k per time unit that is equal to the maximum wage rate of all tasks that are assigned to that station. The objective is to minimize the aggregate wage rate over all stations, while the number of stations is a variable. Production costs per product unit are obtained by multiplying that rate with the given cycle time. The considered objective is equivalent to minimizing the number of stations, if all tasks have the same wage rate. Hence, the problem is a direct generalization of SALBP-1. Rosenberg and Ziegler describe and evaluate priority rule based heuristics, where some of the rules available for SALBP-1 are extended to allow for smoothing the wage rates within each station [50].

Amen [52] extends the problem by additionally considering station related **costs of capital**, i.e., each station is assumed to require a constant pre-specified investment.

Amen presents an exact branch-and-bound procedure which extends respective procedures for SALBP-1 [50] for this problem which uses a station-oriented construction scheme and a laser search strategy based on a topological task labeling. The enumeration is restricted by means of (global and local) lower bounds extending such for SALBP-1 and dominance rules, where the maximal load rule which is essential for solving SALBP-1 is shown to be inappropriate for the cost-oriented problem. Therefore, only weaker versions of this rule and some other SALBP-1 based rules are applied [53].

For the same problem, Amen [54] develops station-oriented priority rule based procedures with cost-oriented dynamic priority rules and compares them to existing ones using a large set of randomly generated problem instances. The new rule which controls the idle time and the difference of wage rates in a station ("best change of idle cost") performs best. Further improvements are obtained by approaches which use several priority rules. The best results are reported for a restricted version of the branch-and-bound procedure outlined above which is based on successively solving small problems each representing a feasible subset of remaining tasks. Malakooti [55] and Malakooti and Kumar [56] consider a multi-objective ALBP with capacity- and cost-oriented objectives and propose different solution approaches including generation of efficient alternatives, interactive approaches and goal programming.

2.3.6.2 Profit-oriented models

The cost-oriented models may be extended by additionally considering profits. The model of Rosenblatt and Carlson [57] includes fixed selling prices, material costs as well as wages and equipment costs. This model is extended by Martin [58] for the case of unpaced lines with buffers, where inventory related cost components are relevant.

CHAPTER 3

FUZZY LOGIC

3.1 Introduction

The thought of fuzzy logic declared in 1965 by Lotfi Zadeh [47]. He claimed that human reasoning is approximate rather than precise in nature. In 1974, Ebraham Mamdani used fuzzy logic to control a simple stream engine for the first time [59]. F. Smidth of Denmark applied fuzzy logic to the control of a cement kiln in 1980. This is the first industrial application of fuzzy logic. In the early 1980s, fuzzy logic was applied to home electronics products and the nontechnical people became aware of fuzzy systems. Life is full of uncertainties. For instance, if the weather is decribed in terms of the exact percentage of cloud cover, it will be too complex. Therefore, people say that it is sunny which is more uncertain and less precise but more useful.

Fuzzyness is particularly vagueness related to human linguistics and thinking. Such words as 'pretty' or 'young' are quite subjective and depend on situations. Thus, fuzzy logic can manage such vagueness mathematically. The applications of fuzzy logic expand rapidly from control to knowledge processing. In recent years, non-engineering applications such as social and environmental systems have been tested. Fuzzy logic enables us to make applications effective.

Many decision-making and problem-solving tasks are too complex to be understood quantitatively, however, people succeed by using knowledge that is imprecise rather than precise. Fuzzy set theory, originally introduced by Lotfi Zadeh in the 1960's, resembles human reasoning in its use of approximate information and uncertainty to generate decisions. It was specifically designed to mathematically represent uncertainty and vagueness and provide formalized tools for dealing with the imprecision intrinsic to many problems. By contrast, traditional computing demands precision down to each bit. Since knowledge can be expressed in a more natural by using fuzzy sets, many engineering and decision problems can be greatly simplified.

Fuzzy logic emerged into the mainstream of information technology in the late 1980's and early 1990's. Fuzzy logic is a departure from classical Boolean logic in that it implements soft linguistic variables on a continuous range of truth values which allows intermediate values to be defined between conventional binary. It can often be considered a superset of Boolean or "crisp logic" in the way fuzzy set theory is a superset of conventional set theory. Since fuzzy logic can handle approximate information in a systematic way, it is ideal for controlling nonlinear systems and for modeling complex systems where an inexact model exists or systems where ambiguity or vagueness is common. A typical fuzzy system consists of a rule base, membership functions, and an inference procedure. Today, fuzzy logic is found in a variety of control applications including chemical process control, manufacturing, and in such consumer products as washing machines, video cameras, and automobiles.

3.2 Crisp Sets and Characteristic Functions

Here is an example to explain crisp sets and characteristic functions. Let's assume a tennis club and its members are defined ad follows:

X=members= {P, Q, R, S, T, U}

A=female members {Q, R, T}

B=student members {Q, R, S}

Union, intersection and complement of A and B are as follows:

$A \cup B = \{Q, R, S, T\}$

$A \cap B = \{Q, R\}$

$\overline{A} = \{P, S, U\}$

$\overline{B} = \{P, T, U\}$

3.2.1 Characteristic Functions

A represent a crisp set on the universe X. Its characteristic function χ_A can be defined by mapping.

$$\chi_A : X \rightarrow \{0,1\} \text{ as}$$

$$\chi_A(X) = \begin{cases} 1 & x \in X \\ 0 & x \notin X \end{cases}$$

It shows that if the element x belongs to **A**, then χ_A is 1 and if it doesn't belong to **A**, χ_A is 0 (zero). This concept is very important in fuzzy sets.

3.3 Fuzzy Set Theory

Fuzzy sets are an extension of classical set theory and are used in fuzzy logic. In classical set theory the membership of elements in relation to a set is assessed in binary terms according to a crisp condition — an element either belongs or does not belong to the set. By contrast, fuzzy set theory permits the gradual assessment of the membership of elements in relation to a set; this is described with the aid of a membership function $\mu \rightarrow [0, 1]$. Fuzzy sets are an extension of classical set theory since, for a certain universe, a membership function may act as an indicator function, mapping all elements to either 1 or 0, as in the classical notion.

3.3.1 Definition

Specifically, a fuzzy set $\tilde{A}$ on a classical set X is defined as follows:

$$\tilde{A} = \{(x, \mu_A(x)) | x \in X\}$$

The membership function $\mu_A(x)$ quantifies the grade of membership of the elements x to the fundamental set X. An element mapping to the value 0 means that the member is not included in the given set, 1 describes a fully included member. Values strictly between 0 and 1 characterize the fuzzy members.

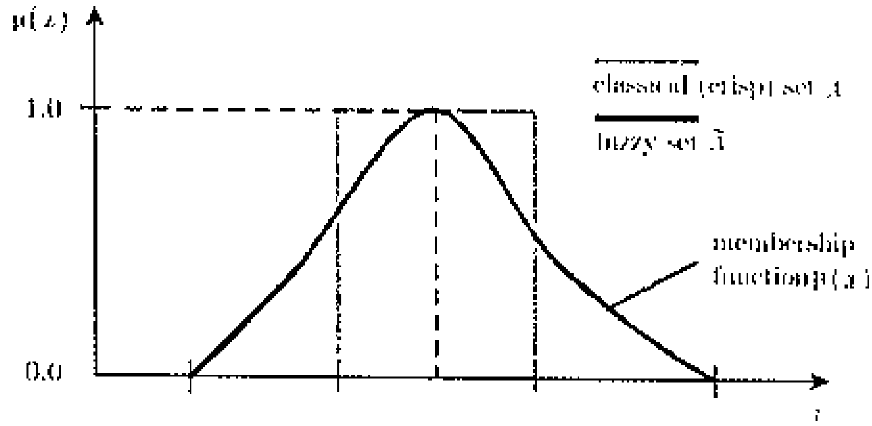


Figure 2.1: Difference between crisp and fuzzy sets

The following holds for the functional values of the membership function $\mu_A(x)$

$$\mu_A(x) \geq 0 \quad \forall x \in X$$
$$\sup_{x \in X}[\mu_A(x)] = 1$$

Fuzzy set brings a clear solution to deal with vague expression such as " a set of tall people" and " the people living close to Dhaka" which are not able to be denoted by convetional set theory. The expression such as " the set of people more than 1.90 in height" or " the people living in Dhaka" can be defined exactly by conventional sets. These are called "crisp sets" in fuzzy set theory.

3.3.2 Fuzzy Subsets

There is a strong relationship between Boolean logic and the concept of a subset. There is a similar strong relationship between fuzzy logic and fuzzy subset theory.A subset U of a set S can be defined as a set of ordered pairs, each with a first element that is an element of the set S, and a second element that is an element of the set { 0, 1 }, with exactly one ordered pair present for each element of S. This defines a mapping between elements of S and elements of the set {0, 1}. The value zero is used to represent non-membership, and the value one is used to represent membership. The truth or falsity of the statement

x is in U

is determined by finding the ordered pair whose first element is x. The statement is true if the second element of the ordered pair is 1, and the statement is false if it is 0.

A fuzzy subset F of a set S can be defined as a set of ordered pairs, each with a first element that is an element of the set S, and a second element that is a value in the interval [0, 1], with exactly one ordered pair present for each element of S. This defines a mapping between elements of the set S and values in the interval [0, 1]. The value zero is used to represent complete non-membership, the value one is used to represent complete membership, and values in between are used to represent intermediate degrees of membership.

The set S is referred to as the universe of discourse for the fuzzy subset *F*. Frequently, the mapping is described as a function, the (membership function) of *F*. The degree to which the statement *x* is in *F* is true is determined by finding the ordered pair whose first element is *x*. The degree of truth of the statement is the second element of the ordered pair. That's a lot of mathematical complexity, so here's an example. Let's talk about people and "tallness". In this case the set S (the universe of discourse) is the set of people. Let's define a fuzzy subset TALL, which will answer the question "to what degree is person x tall?" To each person in the universe of discourse, one has to assign a degree of membership in the fuzzy subset TALL. The easiest way to do this is with a membership function based on the person's height. A graph of this looks like:

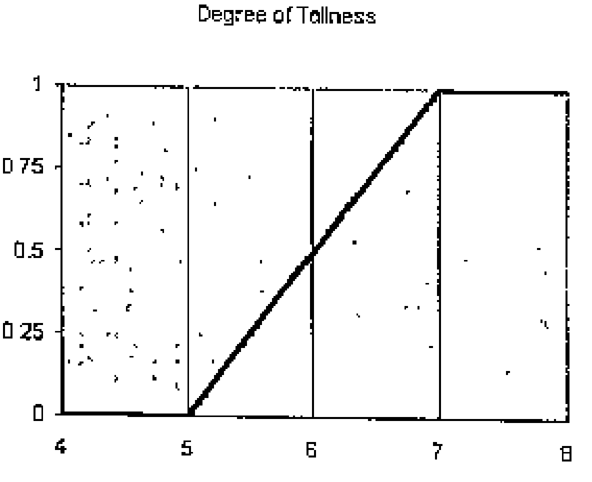


Figure 2.2: Degree of Tallness

The function will be:

$$tall(x) = \begin{cases} 0 & \text{if } height(x) < 5ft. \\ (height(x) - 5ft.)/2ft & \text{if } 5ft <= height(x) <= 7ft \\ 1 & \text{if } height(x) > 7ft. \end{cases}$$

3.3.3 Fuzzy Sets vs. Crisp Sets

Crisp sets are the sets that are used most in our life. In a crisp set, an element is either a member of the set or not. It is defined in such a away as to dichotomize the individuals in some given universe of discourse into two groups: members and nonmembers. A sharp, unambiguous distinction exists between the members and nonmembers of the set.

Fuzzy sets, on the other hand, allow elements to be *partially* in a set. Each element is given a degree of membership in a set. This membership value can range from 0 (not an element of the set) to 1 (a member of the set). It is clear that if one only allowed the extreme membership values of 0 and 1, that this would actually be equivalent to crisp sets. A membership function is the relationship between the values of an element and its degree of membership in a set. An example of membership functions are shown in Figure 1. In this example, the sets are numbers that are negative large, negative medium, negative small, near zero, positive small, positive medium, and positive large. The value, μ [0, 1], is the amount of membership in the set. Each membership function maps elements of a fuzzy set A is denoted by μ_A, that is,

$$\mu_A: X \rightarrow [0, 1]$$

Each fuzzy set is completely and uniquely defined by one particular membership function.

3.3.4 Fuzzy Sets and Membership Functions

In the given example:

A = The set of overweight people

B = The set of people of moderate height

Using Venn diagram to express these fuzzy sets is inconvenient way because the concepts of overweight and moderate height are different from person to person and depend on the situation. The degree of overweight can vary from a little heavy to extremely heavy. In this example, a real number between 0 and 1 is used to a degree .The degree 1 means the person completely belongs to the set of A and 0 degree denotes the person doesn't belong to the set of A.

Table 3.1: Degree of OVERWEIGHT and MODERATE Height

SET	A	B	C	D	E	F
OVERWEIGHT	0.5	0.9	0.3	0.4	0.7	0.6
MOD. HEIGHT	0.4	0.1	0.5	0.7	0.9	0.8

Fuzzy sets can be assumed to be an extension of crisp sets. Therefore, membership functions are the extension of characteristic functions.

A fuzzy set A on the universe X is a set defined by a membership function μ_A representing a mapping

$$\mu_A: X \rightarrow [0, 1]$$

The closer the value of μ_A (X) to 1, the higher the grade of membership of the element x in fuzzy set A. If μ_A (X)=1, the element X completely belongs to the fuzzy set A. If μ_A (X) =0, X does not belong to A at all.

3.3.5 Characteristic Functions and Membership Function: A Comparison

Table 3.2: The Value of Characteristic Functions in Crisp Sets

	Height(cm)	Low	Middle	High
A	179	0	1	0
B	171	0	1	0
C	168	1	0	0

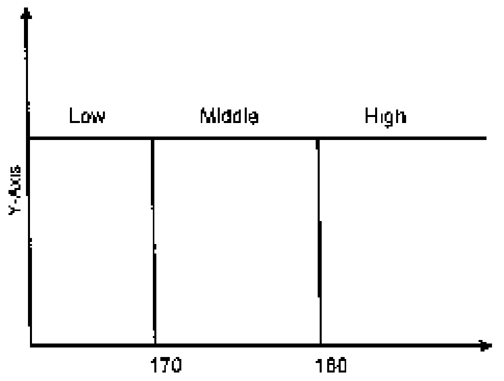


Figure 3.2: Crisp Sets of Height

From the table, A and B belong to the "middle" height set and C belongs to the " low" set although the difference in height of B and C is only 3 cm while the difference between A and B is 8 cm.

Table 3.3: Value of Membership Functions in Fuzzy Sets

	Height (cm)	Low	Middle	High
A	179	0	0.4	0.6
B	171	0.4	0.6	0
C	168	0.7	0.3	0

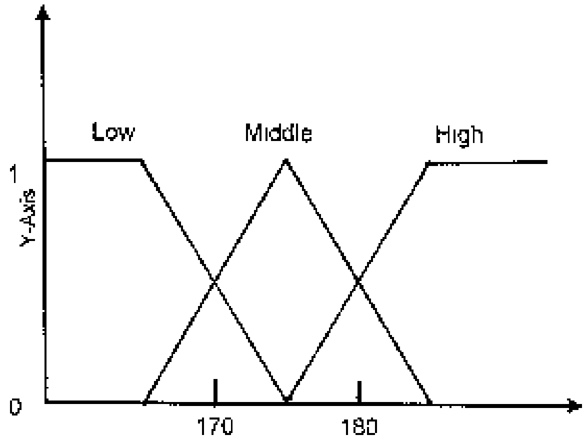


Figure 3.3: Fuzzy Sets of Height

This table indicates that A belongs to the "middle" set in the grade of 0.4 and to the "high" set in the grade of 0.6 whereas A does not belong to low set.

A: higher middle

B: lower middle

C: relatively low

3.3.6 The Notation of Fuzzy Sets

There are three types of fuzzy sets. They are:

1. Fuzzy sets with a discrete nonordered universe
2. Fuzzy sets with a discrete ordered universe
3. Fuzzy sets with a continuous universe

- **Fuzzy sets with a discrete nonordered universe**

Let *X*= {M, N, O} be the set of cities one may choose to live in. The fuzzy set *C*="desirable cities to live in" may be described as follows:

C= {(M, 0.9), (N, 0.8), (O, 0.6)}

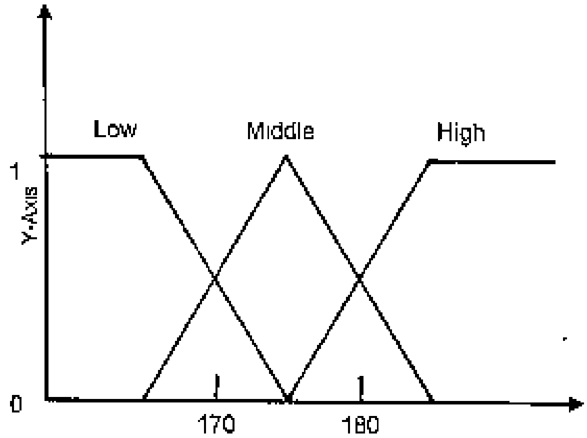


Figure 3.3: Fuzzy Sets of Height

This table indicates that A belongs to the "middle" set in the grade of 0.4 and to the "high" set in the grade of 0.6 whereas A does not belong to low set.

A: higher middle

B: lower middle

C: relatively low

3.3.6 The Notation of Fuzzy Sets

There are three types of fuzzy sets. They are:

1. Fuzzy sets with a discrete nonordered universe
2. Fuzzy sets with a discrete ordered universe
3. Fuzzy sets with a continuous universe

- **Fuzzy sets with a discrete nonordered universe**

Let X= {M, N, O} be the set of cities one may choose to live in. The fuzzy set C="desirable cities to live in" may be described as follows:

C= {(M, 0.9), (N, 0.8), (O, 0.6)}

Apparently the universe of discourse X is discrete and it contains nonordered objects- in this case, three cities. As one can see, the foregoing membership grades listed above are quite subjective; anyone can come up with three different but legitimate values to reflect his or her preference.

- **Fuzzy sets with a discrete ordered universe**

Let X= {0, 1, 2, 3, 4, 5, 6} be the set of members of children a family may choose to have. Then the fuzzy set A="sensible number of children in a family" may be described as follows:

A= {(0.0.1),(1,0.3)'(2,0.7),(3,1),4,0.7),(5,0.3),(6,0.1)}

Here X is a discrete ordered universe; the MF for the fuzzy set A is shown in the figure. Again, the membership grades of this fuzzy set are obviously subjective measures.

- **Fuzzy sets with a continuous universe**

Let $X=R^+$ be the set of possible ages for human beings. Then the fuzzy set B="about 50 years old" may be expressed as

$B= \{(x, \mu_B(x) \mid x \in X\}$

Where, $$\mu_B = \frac{1}{1+(\frac{x-50}{10})^4}$$

For simplicity of notation, an alternative way of denoting a fuzzy set is introduced. A fuzzy set A can be denoted as follows.

$$A = \begin{cases} \sum_{x_i \in X} \mu_A(x_i)/x_i, \text{if } X \textit{ is a collection of discrete object} \\ \int_X \mu_A(x)/x, \quad \textit{if } X \textit{ is a continuous space} \end{cases}$$

In the continuous expression the symbol $\int$ is used an extension of $\sum$ to the continuous world and it is not integral sign. On the lower right of $\int$ sign the name of universe is written.

3.4 Fuzzy Sets: Basic Concepts

α- cut: The α- cut of a fuzzy set A is the crisp set A that contains all the elements of the universal set X whose membership grades in A are greater than or equal to the specified value of α. Given a fuzzy set A defined on X and any number $\alpha \in [0,1]$, the α-cut, ${}^{\alpha}A$, is the crisp set ${}^{\alpha}A = \{x / A(x) \geq \alpha\}$

Strong α- cut: The strong α- cut of a fuzzy set A is the crisp set A that contains all the elements of the universal set X whose membership grades in A are greater than ~~or~~ ~~equal~~ ~~to~~ the specified value of α. Given a fuzzy set A defined on X and any number $\alpha \in [0, 1]$, the strong α-cut, ${}^{\alpha+}A$, is the crisp set ${}^{\alpha+}A = \{x / A(x) \geq \alpha\}$

Level set: the set of all levels $\alpha \in [0, 1]$ that represent distinct α-cuts of a given fuzzy set A is called a level set of A. formally, $\wedge(A) = \{\alpha / A(x) = \alpha \text{ for some } x \in X\}$ Where $\wedge$ denotes the level set of fuzzy set a defined on X.

Support: The support of a fuzzy set A is the set of all point x in X such that $\mu_A(x)>0$:

Support $(A) = \{x \mid \mu_A(x)>0\}$

Core: The core of a fuzzy set A is the set of all points x in X such that $\mu_A(x)=1$;

Core $(A) = \{x \mid \mu_A(x) = 1\}$

3.5 Fundamental Operations of Fuzzy Sets

Fuzzy union, intersection and complements are called 'the standard fuzzy operations'. The standard fuzzy operations are generalizations of the corresponding classical set operations.

3.5.1 Union of Fuzzy set A and B

The union of two fuzzy sets A and B is a fuzzy set C, written as $C=A \cup B$ or $C=A$ OR B, whose MF is related to those of A and B by

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x)$$

3.5.2 Intersection of Fuzzy set A and B

The intersection of two fuzzy sets A and B is a fuzzy set C, written as C=AWB or C=A AND B, whose MF is related to those of A and B by

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x)$$

3.5.3 Complement of Fuzzy set A and B

The complement of fuzzy set A, denoted by A ($\neg A$, NOT A), is defined as

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x)$$

Fuzzy sets have properties which are valid for crisp sets. And yet, there are some properties which are not valid for fuzzy sets.

3.5.4 The Law of Excluded Middle

$A \cup \bar{A} = x$ for crisp set but

$A \cup \bar{A} \neq x$ for fuzzy set

3.5.5 The Law of Contradiction

$A \cap \bar{A} = \phi$ for crisp set but

$A \cap \bar{A} \neq \phi$ for fuzzy set

3.5.6 Equality and Inclusion of Fuzzy set

Equality of fuzzy sets is defined as

$$A{=}B \leftrightarrow \mu_A(x) = \mu_B(x);\ \forall_X X \in X$$

Inclusion of fuzzy sets or A being a subset of B is defined as

$$A{\subset}B \leftrightarrow \mu_A(x) \le \mu_B(x)$$

Example

Age	5	10	20	30	40	50	60	70	80
Young	1	1	0.8	0.5	0.2	0.1	0	0	0
Old	0	0	0.1	0.2	0.4	0.6	0.8	1	1

	5	10	20	30	40	50	60	70	80
Young∪Old	1	1	0.8	0.5	0.2	0.1	0	0	0
Young∩Old	0	0	0.1	0.2	0.4	0.6	0.8	1	1
$\overline{\text{Old}}$	1	1	0.9	0.8	0.6	0.4	0.2	0	0

Example

	5	10	20	30	40	50	60	70	80
Old ∪ $\overline{\text{Old}}$	1	1	0.9	0.8	0.6	0.6	0.8	1	1
Old ∩ $\overline{\text{Old}}$	0	0	0.1	0.2	0.4	0.4	0.2	0	0

3.8 Decomposition Principle

Using α-cuts a membership function $\mu_A(x)$ can be decomposed into an infinite number of rectangular membership functions. When these rectangular membership functions and max-operation are aggregated, the original fuzzy set A can be obtained

$$\mu_A(x) = \max\,[\alpha \wedge XA_\alpha(x)]$$

Here $XA_\alpha(x)$ is a characteristic equation of the set A_α

The difference between strong and weak α-cuts centers on whether they include equality. Here are the illustrations of both α-cut and decomposition sets.

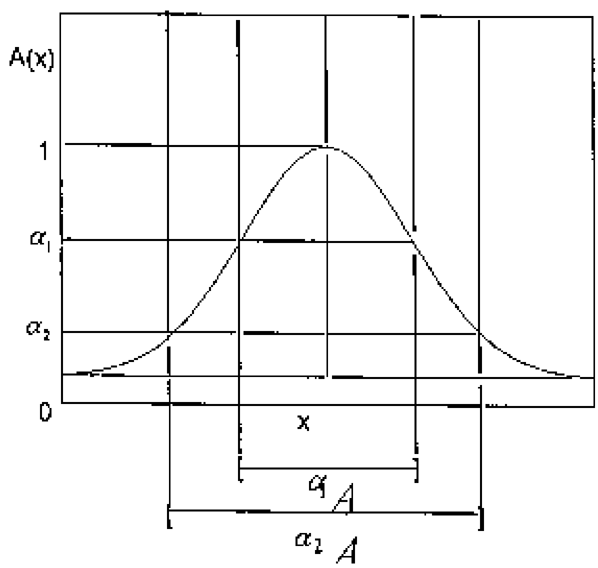


Figure 3.4: α-cut and Decomposition Principle

In this figure, $^{\alpha}A$ is an example of an α-cut. The idea of the decomposition principle is also illustrated in the figure. Let the characteristic function of a weak α cut $XA_\alpha(X)$ for an α (α ∈ (0,1]). Define a rectangular membership function that satisfies $\alpha \wedge XA_\alpha(X)$. Changing the value of α in the interval of α ∈ (0,1] the similar operation is repeated and an infinite number of rectangular membership functions is found. The decomposition principle tells us that the membership function of the original fuzzy set A can be expressed by the max operation of the previously obtained rectangular membership functions.

This is defined by $\mu_A(X) = \text{Max}\ [\ \alpha \wedge XA_\alpha(X)]\ (\alpha \in (0,1])$

3.9 Fuzzy Numbers and Extension Principle

When there is a relation y=3x+2 between x and y, the value of y for x=4 can be calculated by (3 x 4) +2 = 14.Then, how the value of y can be calculated by a fuzzy set such as x = "about y"? The extension principle gives a method for this.

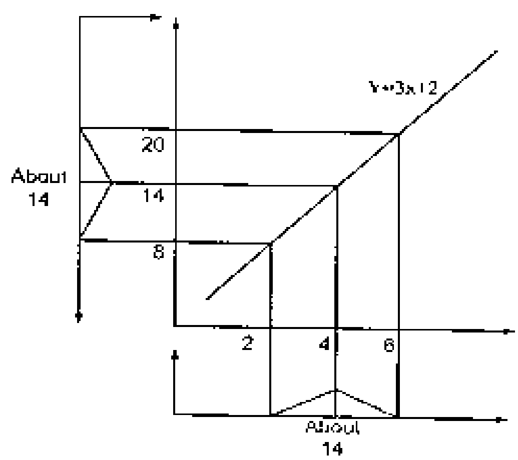


Figure 3.5: Example of extension principle

The process of calculation can be interpreted as 3 x "about 4" + 2 = "about 12" + 3 = "about 14"

Let A be the fuzzy set that gives "about 4" such as A=0.5 / 3 + 1.0 / 4 + 0.5 / 5

Also define x_1 =3, x_2=4, x_3=5 so that y_i=3x_i+2, i =1, 2, 3

$$
\begin{aligned}
f(A) &= \sum_{i=1}^{3} \mu_A(y_i)/y_i \\
&= \sum_{i=1}^{3} \mu_A(y_i)/(3x_i + 2) \\
&= 0.5/(3\times 3+2) + 1.0/(3\times 4+2) + 0.5(3\times 5+2) \\
&= 0.5/11 + 1.0/14 + 0.5/17 \\
&= \text{"}\mathit{about}\ 14\text{"}
\end{aligned}
$$

3.10 Fuzzy Numbers

A fuzzy number is a convex, normalized fuzzy set $\widetilde{A} \subseteq \Re$ whose membership function is at least segmentally continuous and has the functional value $\mu_A(x) = 1$ at precisely one element.

If a fuzzy set A on the universe R of real numbers satisfies the following conditions, it is caled a fuzzy number.

i. A is a convex fuzzy set;
ii. There is only one x_0 that satisfies $\mu_A(x_0) = 1$;
iii. μ_A is continuous in an interval.

If a fuzzy number A satisfies the following condition, it is called a flat fuzzy number.

$(m_1, m_2) \in R; \quad m_1 < m_2$

$\mu_A(x) = 1 \ \forall_x \in [m_1, m_2]$

3.11 Fuzzy Interval

A fuzzy interval is an uncertain set $\widetilde{A} \subseteq \Re$ with a mean interval whose elements possess the membership function value $\mu_A(x) = 1$. As in fuzzy numbers, the membership function must be convex, normalized, and at least continuous in segments.

3.12 Arithmetic Operations of Fuzzy Numbers

Fuzzy arithmetic is based on two properties of fuzzy numbers

1. each fuzzy sets, and thus also each fuzzy number, can fully and uniquely be represented buy it's α-cuts
2. α-cuts of each fuzzy number are closed intervals of real numbers for all $\alpha \in (0,1]$

Let * denotes any of the four arithmetic operations on closed intervals: addition +, subtraction -, multiplication •, and division /.

Then, $[a,b]*[d,e]=\{f*g \mid a \leq f \leq b,\ d \leq g \leq e\}$

As results, the four arithmetic operations on closed intervals are

$[a,b]+[d,e]= [a+d, b+e]$

$[a,b]-[d,e]= [a-e, b-d]$

$[a,b] \cdot [d,e]=[\min(ad, ae, bd, be), \max(ad, ae, bd, be)$

$[a,b]/[d,e]=[\min(a/d, a/e, b/d, b/e), \max(a/d, a/e, b/d, b/e)$, provide that $0 \notin [d,e]$

Let $\tilde{A}=(a,b,c,\lambda)$, $\tilde{A}=(d,e,f,\lambda) \in F_N \in (\lambda)$, $k \in R$ and assume that $C=(p,q,r,\lambda) \in F_N(\lambda)$, then

$$(a)\ \widetilde{C} = \widetilde{A} + \widetilde{B} = (a+d, b+e, c+f; \lambda)$$

$$(b)\ \widetilde{C} = \widetilde{A} - \widetilde{B} = (a-f, b-e, c-d; \lambda)$$

$$(c)\ \widetilde{C} = \widetilde{A} \times \widetilde{B} = (\min(ad, af, cd, cf), be, \max(ad, af, cd, cf); \lambda)$$

$$(d)\ \widetilde{C} = \widetilde{A} \div \widetilde{B} = (\min(a/d, a/f, c/d, c/f), b/e, \max(a/d, a/f, c/d, c/f); \lambda)\widetilde{0}$$
$$= (0,0,0) \notin \widetilde{B}$$

$$(e)\ \text{If } k > 0, \widetilde{C} = \widetilde{k}_1(.)\widetilde{A} = (ka, kb, kc; \lambda)$$

$$(f)\ \text{If } k < 0, \widetilde{C} = \widetilde{k}_1(.)\widetilde{A} = (kc, kb, ka; \lambda)$$

$$(g)\ \text{If } k = 0, \widetilde{C} = \widetilde{k}_1(.)\widetilde{A} = (0,0,0; \lambda)$$

3.13 Application of Fuzzy Logic

Fuzzy Set Theory defines Fuzzy Operators on Fuzzy Sets. The problem in applying this is that the appropriate Fuzzy Operator may not be known. For this reason, Fuzzy logic usually uses IF/THEN rules, or constructs that are equivalent, such as fuzzy associative matrices.

Rules are usually expressed in the form IF variable IS set THEN action. For example, an extremely simple temperature regulator that uses a fan might look like this IF temperature IS very cold THEN stop fan IF temperature IS cold THEN turn down fan IF temperature IS normal THEN maintain level IF temperature IS hot THEN speed up fan.

Notice there is no "ELSE". All of the rules are evaluated, because the temperature might be "cold" and "normal" at the same time to differing degrees. The AND, OR, and NOT operators of boolean logic exist in fuzzy logic, usually defined as the minimum, maximum, and complement; when they are defined this way, they are called the Zadeh operators, because they were first defined as such in Zadeh's original papers. So for the fuzzy variables x and y:

NOT x = (1 - truth(x))

x AND y = minimum(truth(x), truth(y))

x OR y = maximum(truth(x), truth(y))

There are also other operators, more linguistic in nature, called hedges that can be applied. These are generally adverbs such as "very", or "somewhat", which modify the meaning of a set using a mathematical formula.

In application, the programming language Prolog is well geared to implementing fuzzy logic with its facilities to set up a database of "rules" which are queried to deduct logic. This sort of programming is known as logic programming.

3.14 Fuzzy Ranking Methods

To handle the fuzzy characteristic of the FMABL problem, the fuzzy processing time is needed to be ordered and compared with fuzzy cycle time of each product model. The requirement of this ranking operator is not only simple in computation but also flexible enough to adapt with different sources of data from the real-life because the processing time and maximum cycle time could be estimated by linguistic terms (short, medium, long, etc.) or just an interval of data. In literature, several fuzzy comparison methods have been proposed such as pseudo order fuzzy preference model [60], new fuzzy-weighted average [61], and signed distance method [62]. Among these methods the signed distance method is suitable for fuzzy time comparison because it is simple in computation and flexible to convert from the interval data. Furthermore, fuzzy arithmetic needs to be performed to calculate fuzzy times in our heuristic However, there are many fundamental problems with fuzzy arithmetic. Its operation is principally based on the extension principle and/or interval arithmetic. The problem of extension principle is that it is computationally too complex. Another fundamental problem with fuzzy arithmetic is the existence and uniqueness problem of the fuzzy numbers. For example, even for the simple operation of fuzzy subtraction of two fuzzy numbers, the results may not exist. At this point, one may believe that fuzzy arithmetic based on interval operations could be better because it is easy to calculate. However, a difficulty with fuzzy arithmetic using interval principles is the non-linear representation of results. Thus it could lead to increase the computational complexity. There is one solution to overcome this difficulty by approximating the operation's results in a linear form.

3.14.1 Signed Distance Ranking of the Level λ Fuzzy Numbers

Yao and Wu presented signed distance method of ranking triangular fuzzy numbers [62]. They presented the following definitions:

Definition 1. If the memberships function of the fuzzy set A on $\boldsymbol{R}$ is

$$\mu_A(x)=\begin{cases} \lambda(x-p)/(q-p), & p\leq x\leq q\,, \\ \lambda(r-x)/(r-q), & q\leq x\leq r\,, \\ 0, & \text{otherwise} \end{cases}$$

$p<q<r$, then A is called a level λ fuzzy number, $0\leq \lambda \leq 1$

$A=(p, q, r, \lambda)$, let, $F_N(\lambda)$ be the family of all level λ fuzzy numbers

$F_N(\lambda)=\{(p, q, r, \lambda)\mid p<q<r;\ p, q, r\in R\},\ 0<\lambda\leq 1$

Definition 2. Signed Distance

For each a, 0∈R, the definition of signed distance d_0 from a to 0 is expressed by $d_0(a,0)=a$

If a>0, $d_0(a,0)$ implies that a is on the right hand side of 0 with distance $d_0(a,0)=a$

If a<0, $d_0(a,0)$ implies that a is on the left hand side of 0 with distance $-d_0(a,0)=-a$

Which is called the signed distance of 'a' which is measure from 0.

Definition 3. The signed distance of TFN $\tilde{A}$= (p, q, r) is defined as

$$d(\tilde{A},0)=\frac{1}{2}\int_0^1[p+(q-p)\alpha+r-(r-q)\alpha]d\alpha=\frac{1}{4}(2q+p+r).$$

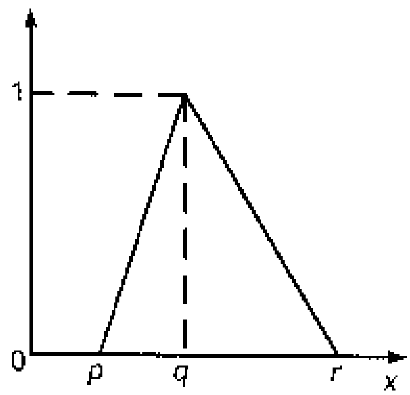


Fig 3.6: Level λ Fuzzy Set

Let $\tilde{A}$= (p, q, r, λ) ∈ $F_N(\lambda)$ the signed distance of $\tilde{A}$ from 0_1(y axis) is

$$d(\tilde{A},\tilde{0}_1)=\frac{1}{\lambda}\int_0^\lambda\frac{1}{2}\left[p+r+(2q-p-r)\frac{\alpha}{\lambda}\right]d\alpha$$

$$=\frac{1}{4}(2q+p+r)$$

Definition 4. For each $\lambda \in (0, 1]$, the ranking of the level λ fuzzy numbers in $F_N(\lambda)$ is defined as

$$\tilde{B} \prec \tilde{A} \text{ iff } d(\tilde{B},\tilde{0}_1) < d(\tilde{A},\tilde{0}_1)$$
$$\tilde{B} \approx \tilde{A} \text{ iff } d(\tilde{B},\tilde{0}_1) = d(\tilde{A},\tilde{0}_1)$$
$$\tilde{B} \succ \tilde{A} \text{ iff } d(\tilde{B},\tilde{0}_1) < d(\tilde{A},\tilde{0}_1)$$

Proposition 1. For $\tilde{A}, \tilde{B}, \tilde{C} \in F_N(\lambda)$

a. If $\tilde{A} \prec\approx \tilde{B}$ and $\tilde{B} \prec\approx \tilde{A}$ then $\tilde{A} \approx \tilde{B}$

b. If $\tilde{A} \prec \tilde{B}$ and $\tilde{B} \prec \tilde{C}$ then $\tilde{A} \prec \tilde{C}$

c. In $F_N(\lambda)$, there is only one $\tilde{A} \prec \tilde{B}$ and $\tilde{B} \approx \tilde{A}, \tilde{A} \prec \tilde{B}$ holds

The signed distance method can be easily implemented to rank different types of fuzzy numbers. In this case, the ranking method of TFN is used.

CHAPTER 4

GENETIC ALGORITHM

4.1 Introduction

A genetic algorithm (or short GA) is a search technique used in computing to find true or approximate solutions to optimization and search problems. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover (also called recombination).

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype or the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly mutated) to form a new population. The new population is then used in the next iteration of the algorithm.

4.2 History

Computer simulations of evolution started with Nils Aall Barricelli [63]. Barricelli was simulating the evolution of automata that played a simple card game. Starting in 1957, the Australian quantitative geneticist Alex Fraser published a paper on simulation of artificial selection of organisms with multiple loci controlling a measurable trait [64]. From these beginnings, computer simulation of evolution by biologists became more common in the early 1960s, and the methods were described in books by Fraser and Burnell [65] and Crosby [66].

Although Barricelli had also used evolutionary simulation as a general optimization method, genetic algorithms became a widely recognized optimization method as a result of the work of John Holland in the early 1970s [42]. His work originated with studies of cellular automata, conducted by Holland and his colleagues at the University of Michigan. Research in GAs remained largely theoretical until the mid-1980s, when The First International Conference on Genetic Algorithms was held at The University of Illinois. As academic interest grew, the dramatic increase in desktop computational power allowed for practical application of the new technique. In 1989, The New York Times writer John Markoff wrote about Evolver, the first commercially available desktop genetic algorithm. Custom computer applications began to emerge in a wide variety of fields, and these algorithms are now used by a majority of Fortune 500 companies to solve difficult scheduling, data fitting, trend spotting and budgeting problems, and virtually any other type of combinatorial optimization problem.

4.3 GA procedure

A typical genetic algorithm requires two things to be defined.

1. A genetic representation of the solution domain,
2. A fitness function to evaluate the solution domain

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size that facilitates simple crossover operation. Variable length representations were also used, but crossover implementation is more complex in this case. Tree-like representations are explored in Genetic programming and free-form representations are explored in Human Based Genetic Algorithm (HBGA).

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once the genetic representation and the fitness function are defined, GA proceeds to initialize a population of solutions randomly, and then improve it through repetitive application of mutation, crossover, and selection operators.

4.4 Pseudo-code Algorithm

1. Choose initial population
2. Evaluate the fitness of each individual in the population
    a. Repeat
    b. Select best-ranking individuals to reproduce
    c. Breed new generation through crossover and mutation (genetic operations) and give birth to offspring
    d. Evaluate the individual fitness of the offspring
3. Replace worst ranked part of population with offspring
4. Until terminating condition is met

4.5 Initialization

Initially many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, covering the entire range of possible solutions (the search space). Occasionally, the solutions may be "seeded" in areas where optimal solutions are likely to be found.

4.6 Encoding a Chromosome

The chromosome should contain information about the solution it represents.

4.6.1 Binary Encoding

One way of encoding is a binary string. The chromosome could look like this:

Chromosome 1	1101100100110110
Chromosome 2	1101011000011110

Each bit in the string can represent some characteristic of the solution or it could represent whether or not some particular characteristic was present. Another possibility is that the chromosome could contain just 4 numbers where each number is represented by 4 bits (the highest number therefore being 15.)

4.6.2 Permutation Encoding

Permutation encoding can be used in ordering problems, such as the traveling salesman problem or a task ordering problem. Every chromosome is a string of numbers, which represents number in a sequence. In the TSP each number would represent a city to be visited.

Chromosome 1	1 4 7 9 6 3 5 0 2 8
Chromosome 2	9 3 2 5 8 1 6 0 4 7

4.6.3 Value Encoding

Direct value encoding can be used in problems where some complicated values, such as real numbers, are used and where binary encoding would not suffice. While value encoding is very good for some problems, it is often necessary to develop some specific crossover and mutation techniques for these chromosomes.

Chromosome 1	A B E D B C A E D D
Chromosome 2	N W W N E S S W N N

In chromosome 1 above, A could represent a particular task, B another, etc. For chromosome 2 N could be north, S south, and thus could be the path through a maze.

4.6.4 Tree Encoding

Tree encoding is used to actually have programs or expressions evolve. In tree encoding every chromosome is a tree of some objects, such as functions or commands in the programming language. LISP is often used for this because programs in LISP can be represented in this form and then be easily parsed as a tree.

4.7 Crossover

Crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is an analogy to reproduction and biological crossover, upon which genetic algorithms are based.

4.7.1 Crossover Techniques

Many crossover techniques exist for organisms which use different data structures to store themselves.

4.7.1.1 One Point Crossover

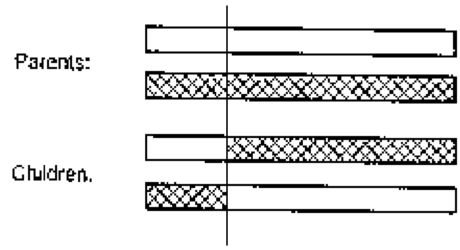


Figure 4.1: One Point Crossover

A crossover point on the parent organism string is selected. All data beyond that point in the organism string is swapped between the two parent organisms. The resulting organisms are the children.

4.7.1.2 Two Point Crossover

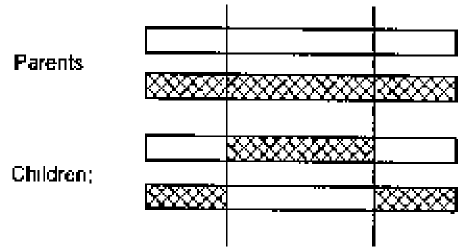


Figure 4.2: Two Point Crossover

Two point crossover calls for two points to be selected on the parent organism strings. Everything between the two points is swapped between the parent organisms, rendering two child organisms. Two point crossover shows better result in elitist process. The variety of genes is greater in two point crossover than single point crossover.

4.7.1.3 Cut and Splice Crossover

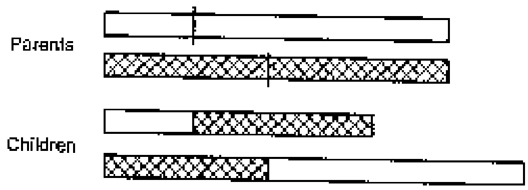


Figure 4.3: Cut and Splice Crossover

Another crossover variant, the "cut and splice" approach, results in a change in length of the children strings. The reason for this difference is that each parent string has a separate choice of crossover point.

4.7.1.4 Uniform Crossover and Half Uniform Crossover

In both these schemes: the two parents are combined to produce two new offspring. In the uniform crossover scheme (UX) individual bits in the string are compared between two parents. The bits are swapped with a fixed probability, typically 0.5.In the half uniform crossover scheme (HUX); exactly half of the mismatching bits are swapped. Thus first the Hamming distance (the number of differing bits) is calculated. This number is divided by two. The resulting number is how many of the bits that do not match between the two parents will be swapped.

4.7.2 Crossover for Ordered Chromosomes

Depending on how the chromosome represents the solution, a direct swap may not be possible. One such case is when the chromosome is an ordered list, such as an ordered list the cities to be traveled for the traveling salesman problem. A crossover point is selected on the parents. Since the chromosome is an ordered list, a direct swap would introduce duplicates and remove necessary candidates from the list. Instead, the chromosome up to the crossover point is retained for each parent. The information after the crossover point is ordered as it is ordered in the other parent. For example, if our two parents are ABCDEFGHI and IGAHFDBEC and our crossover point is after the fourth character, then the resulting children would be

ABCDIGHFE and IGAHBCDEF. Other possible methods include the edge recombination operator and partially mapped crossover.

4.7.3 Crossover biases

For crossover operators which exchange contiguous sections of the chromosomes (e.g. k-point) the ordering of the variables may become important. This is particularly true when good solutions contain building blocks which might be disrupted by a non-respectful crossover operator.

Partially matching crossover (PMX)

PMX may be viewed by a crossover of permutations that guarantees that all positions are found exactly once in each offspring i.e. each offspring receives a full set of genes followed by the corresponding filling in of alleles from their parents. PMX proceeds as follows 1) the two chromosomes are aligned 2) two crossing sites are selected uniformly at random along the strings, defining a matching section.3) The matching section is used to affect a cross through position-by-position exchange operation4) Alleles are moved to their new positions in the offspring

The following illustrates how PMX works:

Let us consider a traveling salesman problem. There are eight cities where the salesman has to travel. This is a problem in discrete or combinatorial optimization.

Under PMX, two strings (permutations and their associated alleles) are aligned, and two crossing sites are picked uniformly at random along the strings. These two points define a matching section that is used to affect a cross through position-by-position exchange operation:

There are two strings:

A= 9 8 4 |5 6 7| 1 3 2 10   B= 8 7 1 |2 3 10| 9 5 4 6

PMX proceeds by position wise exchange. First, mapping string B to string A, the 5 and the 2, the 3 and the 6, and the 10 and 7 exchange places. Similarly mapping string A to string B, the 5 and the 2, the 6 and the 3, and the 7 and the 10 exchange places. Following PMX two offspring are left, A′ and B′:

A′= 9 8 4| 2 3 10 | 1 6 5 7

B′= 8 10 1| 5 6 7 | 9 2 4 7

where each string contains ordering information partially determined by each of its parents.

Order Crossover (OX)

The order crossover operator starts off in a manner similar to PMX. Let the example strings A and B consider again. First a matching section is selected (for comparison, the matching section of PMX example is chosen):

A= 9 8 4 |5 6 7 | 1 3 2 10

B= 8 7 1 |2 3 10 | 9 5 4 6

Like PMX, each string maps to constituents of the matching section of its mate. Instead of using point-by-point exchanges to effect the mapping as PMX does, order crossover uses a sliding motion to fill the holes left by transferring the mapped positions. For example, when string B maps to string A, the cities 5, 6, and 7 will leave holes (marked by an H) in the string:

A= 9 8 4 |5 6 7 | 1 H H H

B= 8 H 1 |2 3 10 | 9 H 4 H

These holes are filled with the matching section city names taken from the mate. Performing this operation and completing the complementary cross the offspring A′ and B′ are obtained as follows:

A′=5 6 7 | 2 3 10 | 1 9 8 4

B′=2 3 10 | 5 6 7 | 9 4 8 1

Although PMX and OX are similar, they process different kinds of similarities.

Cycle Crossover (CX)

The cycle crossover operator is a cross of a different color. Cycle crossover performs recombination under the constraint that each gene comes from the one parent or the other. To see how this is done example tours C and D are started with below:

C=9 8 2 1 7 4 5 10 6 3

D=1 2 3 4 5 6 7 8 9 10

Instead of choosing crossover sites, the process is started at the left and a city is chosen from the first parent:

C′=9----------------

Since every city is to be taken from one of the two parents, city 1 will be taken from string C because of the 1 in position of string D.

C′=9----------1-------------

This selection requires that city 4 will be selected from string C. the process continues until the following pattern is left:

C′=9 2 3 1 5 4 7 8 6 10   D′=1 8 2 4 7 6 5 10 9 3

4.8 Fitness Function

A fitness function is a particular type of objective function that quantifies the optimality of a solution (that is, a chromosome) in a genetic algorithm so that that particular chromosome may be ranked against all the other chromosomes. Optimal chromosomes, or at least chromosomes which are more optimal, are allowed to breed and mix their datasets by any of several techniques, producing a new generation that will (hopefully) be even better. Another way of looking at fitness functions is in terms of a fitness landscape, which shows the fitness for each possible chromosome.

An ideal fitness function correlates closely with the algorithm's goal, and yet may be computed quickly. Speed of execution is very important, as a typical genetic algorithm must be iterated many, many times in order to produce a usable result for a non-trivial problem.

Definition of the fitness function is not straightforward in many cases and often is performed iteratively if the fittest solutions produced by GA are not what is desired. In some cases, it is very hard or impossible to come up even with a guess of what fitness function definition might be. Interactive genetic algorithms address this difficulty by outsourcing evaluation to external agents (normally humans)

4.8.1 Fitness Scaling

Recall that the two undesirable characteristics of FPS are:

- Premature convergence: Early on, a few super-individuals come to dominate selection
- Stagnation: Later on, selective pressure "disappears"

Fitness scaling offers a way to alleviate both of these problems. There are 3 general scaling methods:

a) Linear scaling

b) Sigma truncation

c) Power law scaling

4.9 Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by a fitness function) are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as this process may be very time-consuming.

Most functions are stochastic and designed so that a small proportion of less fit solutions are selected. This helps keep the diversity of the population large, preventing premature convergence on poor solutions. Popular and well-studied selection methods include roulette wheel selection and tournament selection.

4.10 Reproduction

The next step is to generate a second generation population of solutions from those selected through genetic operators: crossover (also called recombination), and/or mutation. In genetic algorithms, mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation.

The classic example of a mutation operator involves a probability that an arbitrary bit in a genetic sequence will be changed from its original state. A common method of implementing the mutation operator involves generating a random variable for each bit in a sequence. This random variable tells whether or not a particular bit will be modified.

The purpose of mutation in GAs is to allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution. This reasoning also explains the fact that most GA systems avoid only taking the fittest of the population in generating the next but rather a random (or semi-random) selection with a weighting toward those that are fitter.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each child, and the process continues until a new population of solutions of appropriate size is generated.

These processes ultimately result in the next generation population of chromosomes that is different from the initial generation. Generally the average fitness will have increased by this procedure for the population, since only the best organisms from the first generation are selected for breeding, along with a small proportion of less fit solutions, for reasons already mentioned above.

4.11 Termination

This generational process is repeated until a termination condition has been reached. Common terminating conditions are

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above.

4.12 Variants of GA

The simplest algorithm represents each chromosome as a bit string. Typically, numeric parameters can be represented by integers, though it is possible to use floating point representations. The basic algorithm performs crossover and mutation at the bit level. Other variants treat the chromosome as a list of numbers which are indexes into an instruction table, nodes in a linked list, hashes, objects, or any other imaginable data structure. Crossover and mutation are performed so as to respect data element boundaries. For most data types, specific variation operators can be designed. Different chromosomal data types seem to work better or worse for different specific problem domains.

Other approaches involve using arrays of real-valued numbers instead of bit strings to represent chromosomes. Theoretically, the smaller the alphabet, the better the performance, but paradoxically, good results have been obtained from using real-valued chromosomes.

A slight, but very successful variant of the general process of constructing a new population is to allow some of the better organisms from the current generation to carry over to the next, unaltered. This strategy is known as elitist selection.

It can be quite effective to combine GA with other optimization methods. GA tends to be quite good at finding generally good global solutions, but quite inefficient at finding the last few mutations to find the absolute optimum. Other techniques (such as simple hill climbing) are quite efficient at finding absolute optimum in a limited region. Alternating GA and hill climbing can improve the efficiency of GA while overcoming the lack of robustness of hill climbing.

CHAPTER 5

PROBLEM FORMULATION

5.1 Introduction

An **assembly line** consists of **(work) stations** k=1,..., m arranged along a conveyor belt or a similar mechanical material handling equipment. The workpieces (jobs) are consecutively launched down the line and are moved from station to station. At each station, certain operations are repeatedly performed regarding the **cycle time c** (maximum or average time available for each work cycle). The cycle time c determines the **production rate** which is 1/c.The decision problem of optimally partitioning (balancing) the assembly work among the stations with respect to some objective is known as the **Assembly Line Balancing Problem (ALBP).**

Manufacturing a product on an assembly line requires partitioning the total amount of work into a set of elementary operations named **tasks** V = {1,...,n}. Performing a task j takes a **task time** t_j and requires certain equipment of machines and/or skills of workers. Due to technological and organizational conditions **precedence constraints** between the tasks have to be observed. These elements can be summarized and visualized by a **precedence graph**. It contains a node for each task, node weights for the task times and arcs for the precedence constraints.

Any type of ALBP consists in finding a feasible line balance, i.e., an assignment of each task to a station such that the precedence constraints and further restrictions are fulfilled The set Sk of tasks assigned to a station k (=1,...,m) constitutes its station load, the cumulated task time is called station time. When a fixed common cycle time c is given, a line balance is feasible only if the station time of neither station exceeds c. In case of t(Sk) < c, the station k has an idle time of c – t(Sk) time units in each cycle. A simple lower bound on the minimal number of stations is $LB1=\lceil t_{sum}/c \rceil$ ($\lceil x \rceil$ denotes the smallest integer not being smaller than x)

The installation of an assembly line is a long-term decision and usually requires large capital investments. Therefore, it is important that such a system is designed and balanced so that it works as efficiently as possible. Besides balancing a new system, a running one has to be re-balanced periodically or after changes in the production process or the production program have taken place. Because of the long-term effect of balancing decisions, the used **objectives** have to be carefully chosen considering the strategic goals of the enterprise. From an economic point of view **cost and profit related objectives** should be considered. However, measuring and predicting the cost of running a line over months or years and the profits achieved by selling the products assembled is rather complicated and error-prone. A usual alternative objective consists in maximizing the line utilization which is measured by the **line efficiency** as the productive fraction of the line's total operating time and directly depends on the cycle time **c** and the number of stations **m**.

Balance delay time is another important factor. It is the amount of idle time on production assembly lines caused by the uneven division of work among operators or stations. It is related to the extent and way the total task is subdivided. In this mathematical model the problem of balance delay is treated empirically and analytically. Empirical studies show that high balance delay is associated with a wide range of work-element times and a high degree of line mechanization.

5.2 Mathematical Model

The processing time and cycle time are of fuzzy nature. The objective is to distribute n tasks among m stations in the assembly line so that minimum number of station with minimum idle time can be achieved and the outcome will be a highly balanced line i.e. the smoothness index will be lowest. The inputs are fuzzy cycle time and fuzzy task time.

$$\textit{Objective Function}: \min \sum_{i=1}^{m} \sum_{j=1}^{n} [x_i + (\tilde{c} - \tilde{s}_i) y_{ij} + \sqrt{(\tilde{c} - s_i)^2 y_{ij}}]$$

subject to,

$$\sum_{i=1}^{m} \sum_{j=1}^{n} \tilde{t}_j y_{ij} \le \tilde{c};\ i = 1,2,3......,m............(1)$$

$$\sum_{i=E_j}^{l} y_{ij} = 1;\quad j = 1,2,3........,n.....................(2)$$

$$\sum_{j=1}^{m} y_{ij} y_{ik} - (\sum_{i=1}^{p} y_{ij} - \sum_{i=p+1}^{m} y_{ik}) = 0..............(3)$$

$$\tilde{c} - \sum_{i=1}^{m} \sum_{j=1}^{n} \tilde{t}_j y_{ij} \ge 0...............................(4)$$

$$\text{Efficiency}, \tilde{\eta} = \frac{\tilde{T}}{x_{best} \times \tilde{c}}$$

Here,

x_i = No of station

$\tilde{c}$ = Fuzzy Cycle Time

$\tilde{t}_i$ = Fuzzy Task Time (TFN)

$\tilde{s}_i$ = Fuzzy Station Time for Station i

y_{ij} = 1; if task j is assigned to station i

= 0; otherwise

i= Work station index, i=1, 2, 3..... , m

j= Task index, j=1, 2, 3,....., n

Task j is assigned to station $i \in [E_j, L_j]$ iff $P_j^* \subseteq s_1 \cup ..... \cup s_i$ and $\tilde{t}(s_i) + \tilde{t}_i \le \tilde{c}$

(1) can be written as $\langle s_j, l_j, r_j\rangle y_{ij} \leqslant \langle t,u,v\rangle$

if $A=\langle s_1, l_1, r_1\rangle$ and $B=\langle s_2, l_2, r_2\rangle$

$A \leqslant B$ iff $s_1<s_2$, $s_1-l_1<s_2-l_2$, $s_1+r_1<s_2+r_2$

So, from (1), it can be expressed as

$$\sum_{i=1}^{m}\sum_{j=1}^{n} s_j y_{ij} \le t \qquad (1)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} (s_j - l_j) y_{ij} \le (t-u) \qquad (2)$$

$$\sum_{i=1}^{m}\sum_{j=1}^{n} (s_j + r_j) y_{ij} \le (t+v) \qquad (3)$$

$$\sum_{k=E_i}^{L_i} y_{ij} = 1;\ j = 1,2,3,\ldots\ldots,n \qquad (4)$$

$$\sum_{i=1}^{m} y_{ij} y_{ik} - \left(\sum_{i=1}^{p} y_{ij} - \sum_{i=p+1}^{m} y_{ik} \right) = 0 \qquad (5)$$

$$\tilde{c} - \sum_{i=1}^{m}\sum_{j=1}^{n} \tilde{t}_j y_{ij} \ge 0 \qquad (6)$$

5.3 Genetic Algorithm Searching Method

Genetic algorithms (GA) are a general concept for solving complex optimization problems which is based on manipulating a population of solutions by genetic operators like selection, recombination and mutation [43]. In order to adapt the general approach to SALBP (or a generalized ALBP), two main difficulties have to be resolved:

• For manipulating solutions by means of genetic operators, they have to be encoded in form of 'chromosomes' each of which consists of a sequence of genes. Several encoding schemes are possible each having pros and cons concerning the type of applicable genetic operators. In particular, pertaining feasibility of manipulated solutions is a critical issue.

• The objective function of SALBP-1 is not operational for guiding the search to promising parts of the solution space, because it does not give a strong distinction between the solution's fitness: Usually there are a few optimal solutions which require the minimal number m* of stations and many others "around them" most of which may require m* + 1 (or some more) stations. That is, a population might consist of solutions all having the same or a few different objective values such that selecting the most promising ones is not obvious. So the fitness function value is modified to search the highly balanced line with minimum idle time. The sequential steps of the proposed algorithm are as follows:

Step 1: Ranking the fuzzy task times according to signed distance value.

Step 2: Generation of initial parent strings.

Step 3: Generation of offspring using crossover procedure.

Step 4: Generation of initial population pool of randomly constructed solutions (a solution is represented by a problem-specific structure of characters or bits) using scramble mutation.

Step 5: Selection of two solutions based on their fitness function value and generation of new solutions using crossover procedures which are supposed to provide inheritance of some basic properties of parent structures by the offsprings.

Step 6: Mutation of child structures with controlled mutation rate M_r which implies exchange of number of elements between two randomly selected positions in a structure.

Step 7: Decoding/evaluation of the child structures to obtain the objective function values.

Step 8: Selection procedure including comparison of child solutions with the worst solutions in the population and replacement of the worst solution by the new one if it is better.

Step 9: Termination of the algorithm after repetition of step 5 to step 8 R times, where R is an initially specified parameter.

5.4 Standard Encoding

The first step in constructing a genetic algorithm is defining a genetic representation (encoding). Having a good representation that can well describe problem-specific characteristics is crucial since it significantly affects all the subsequent steps of the GA. Three string representations applicable to ALB problems are introduced [67].

(1) Workstation-oriented representation: if task i is assigned to workstation j, the station number, j, is placed at the i-th position in the string.

(2) Sequence-oriented representation: all tasks are sequentially listed in the order that the tasks are assigned to workstations.

(3) Partition-oriented representation: separators are introduced in the sequence-oriented representation to partition tasks into workstations.

Workstation-oriented representation can handle all types of ALBP. The other two representations are applicable only to the Type-2 and Type-E problems where the number of workstations is pre-specified. It provides considerable flexibility in choosing genetic operators. Many genetic operators that have been developed for sequencing problems are available, and the representation makes it possible for them to be adapted to ALB problems. The other two representations require complicated additional operations, such as the use of penalty functions. In the proposed algorithm, the workstation-oriented encoding method has been used.

The chromosome is defined as a vector containing the labels of the stations to which the tasks 1, . . .,n are assigned [67]. When standard crossovers or mutations are applied to such chromosomes, the resulting solutions are often infeasible. This aspect must be dealt with by penalizing infeasibilities or rearranging the solution by certain heuristic strategies. Kim et al. [68] achieve populations without infeasible solutions by decoding chromosomes using a procedure similar to that of Helgeson and Birnie [8]. In the proposed algorithm, only valid pools of chromosomes are considered.

5.5 Initial population

A genetic algorithm operates on a population of individual strings. Either heuristic procedures or random creations can be used to generate feasible strings that form the initial population. Anderson and Ferris [45] have mentioned that the performance of the GA scheme is not as good from the pre-selected starting population as it is from a random start. In this research, individuals in an initial population are all randomly generated. As mentioned earlier, the initial strings should maintain feasible sequences. So from the list generated on the basis of signed distance method an initial parent strings are formed. If it is infeasible then repair method is used to create a feasible string.

5.6 Crossover operator

Crossover or mutation depends on a certain probability, i.e., if the probability of recombining is 98% then the probability of mutating is 2% In this case, the crossover (recombination) operator is a variant of Davis' [69] order crossover operator. The two parents that are selected for crossover are cut at two random cutpoints. The offspring takes the same genes outside the cut-points at the same location as its parent and the genes in between the cut-points are scrambled according to the order that they have in the other parent. The major reason that makes this crossover operator very suitable for ALB is that it assures feasibility of the offspring. Since both parents are feasible, both children must also be feasible. Keeping a feasible population is a key to ALB problem since preserving feasibility drastically reduces computational effort. The genes can be swapped as a single bit or a cluster. The number of the genes to be swapped can be easily controlled by a user defined percentage of the length of the string.

5.7 Mutation operator

In the proposed algorithm, the scramble mutation operator is used. It was first proposed by Leu et al. [44]. A random cut-point is selected and the genes after the cut-point are randomly replaced (scrambled), assuring feasibility. Elitism, i.e., replacing a parent with an offspring only if the offspring is better than the parent, is applied to both the crossover and the mutation procedures.

5.8 Selection Method

In this algorithm, chromosomes are selected using customized tournament selection method .Tournament selection is one of many methods of selection in genetic algorithms which runs a "tournament" among a few individuals chosen at random from the population and selects the winner (the one with the best fitness) for crossover. Selection pressure can be easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected. In this case, the tournament size is equivalent to the total population including the parents. The best fit chromosomes are selected according to their fitness function value from a mutated pool. At first, chromosomes with minimum number of stations are selected. So the entire pool is curtailed to a pool with smaller population. Then the fitness function values of the valid chromosomes are computed and the best two chromosomes are selected.

In De Jong's [70] study of genetic algorithms in function optimization, a series of parametric studies across a five-function suite of problems suggested that good GA performs requires the choice of a high crossover probability, a low mutation probability (inversely proportional to the population size), and a moderate population size.

5.9 Fitness functions

In GA the better fit solutions survive across generations. Hence the fitness of a solution should reflect its quality with respect to the problem's objective. The selection policy should ensure survival of better fit solutions. In the proposed algorithm,

$$Fitness\ Function = \frac{\sum_{i=1}^{m}(\tilde{c} - s_i)}{m} + \sqrt{\sum_{i=1}^{m}(\tilde{c} - s_i)^2}$$

The fitness function is modified to achieve the optimum number of stations which will be the minimum with the minimum idle time and minimum smoothness index. However, given two different solutions with the same number of stations, one may be better balanced than the other. For example, a line with three stations may have stations times as 30-50-40 or 50-50-20. The 30-50-40 solution is considered to be superior (better balanced) to the 50-50-20 solution. Hence, a fitness function is used that consists of two objectives, i.e., minimizing the number of stations and obtaining balanced station .The first part of the fitness function aims to find the best balance while the second part minimizes the idle time among the solutions that have the same number of stations.

5.10 Stopping Condition

The algorithm terminates after a certain number of iterations. 50, 500 and 1000 values have been used as the number of iterations parameter.

CHAPTER 6

RESULT ANALYSIS

6.1 Computational Result

The proposed fuzzy line balancing algorithm using genetic algorithm has been coded in C and run on a Personal Computer having Intel Pentium- 4 core 2 duo/2.66 GHz and 512 Mb of RAM. The minimum requirement of running the program: Operating System Windows XP or any other Windows Operating System of equivalent file structure and having Microsoft Visual C installed.

To demonstrate the effectiveness and robustness of the approach computational results are presented that is obtained on a set of SALBP-1 problems found in literature. The program was run for different iterations: 50,500 and 1000. For all iteration, the ALBP-1 datasets are used and have been compared with EUREKA [71], SALOME [72] and Hybrid GA [73].

The algorithm has been evaluated on three sets of instances: The Talbot-Set (64 instances) [29], the Hoffman-Set (50 instances) [74] and the Scholl-Set (168 instances) [75]. The combined set consists of 269 instances (minus 13 instances which are in the Talbot-Set as well as in the Hoffmann-Set). The sources of the problems as well as a detailed description are given by Scholl [75].The basic characteristics of the problems are summarized in Table 6.1. The first column displays the author of the instance. The second column shows the number of tasks. The third and fourth column shows the minimum and maximum cycle time. The fifth column gives the number of different cycle times out of the interval, which are used to define a problem in the Talbot-Set (T), in the Hoffmann-Set (H), and in the Scholl-Set (S). The remaining columns contain the minimum processing time, the maximum processing time, the sum of processing times, the order strength in percent, and the time variability ratio t_{max}/t_{min}

Table 6.1: Problem Characteristics

Author	n	C_{min}	C_{max}	t_{min}	t_{max}	t_{sum}	OS (%)	TV
Arcus 1	83	3786	10816	233	3691	75707	59.1	15.8
Arcus 2	111	5755	17067	10	5689	150399	40.4	568.9
Bartholdi1	148	403	805	3	383	5634	25.8	127.7
Bartholdi2	148	84	170	1	83	4234	25.8	83
Bowman	8	20	20	3	17	75	75	5.7
Buxey	29	27	54	1	25	324	50.7	25
Gunther	35	41	81	1	40	483	59.5	40.0
Hahn	53	2004	4676	40	1775	14026	83.8	44.4
Heskiaoff	28	138	342	1	108	1024	22.5	108
Jackson	11	7	21	1	7	46	58.2	7
Jaeschke	9	6	18	1	6	37	83.3	6
Kilbridge	45	56	184	3	55	552	44.6	18.3
Lutz1	32	1414	2828	100	1400	14140	83.5	14
Lutz2	89	11	21	1	10	485	77.6	10
Lutz3	89	75	150	1	74	1644	77.6	74
Mansoor	11	48	94	2	45	185	60.0	22.5
Mertens	7	6	18	1	6	29	52.4	6.0
Mitchell	21	14	39	1	13	105	71.0	13.0
Mukherjee	94	176	351	8	171	4208	44.8	21.4
Roszieg	25	14	32	1	13	125	71.7	13.0
Sawyer	30	25	75	1	25	324	4.8	25.0
Scholl	297	1394	2787	5	1386	69655	58.2	277.2
Tonge	70	160	527	1	156	3510	59.4	156.0
Warnecke	58	54	11	7	53	1548	59.1	7.6
Wee-mag	75	28	56	2	27	1499	22.7	13.5

Table 6.2 Benchmarking with EUREKA

Author	No of Operations	Cycle Time	Total Processing Time	Optimal No of Stations	Ideal Minimum Idle Time	EUREKA No Of Station	Proposed Algorithm No Of Station	Total Idle Time	Idle Time (%)	Efficiency	Rel. Dev. From EUREKA[%]
Bowman	8	20	75	5	25	5	5	25	25	75.00	0 00
Merten	7	6	29	6	7	6	6	7	19.44	80.56	0.00
		7	29	5	6	5	6	13	30 95	69.05	20.00
		8	29	5	11	5	5	11	27.5	72.50	0.00
		10	29	3	1	3	4	11	27.5	72 50	33 33
		15	29	2	1	2	3	16	35.56	64.44	50.00
		18	29	2	7	2	2	7	19.44	80.56	0.00
Jaeschke	9	6	37	8	11	8	8	11	22.92	77.08	0.00
		7	37	7	12	7	8	19	33 93	66 07	14 29
		8	37	6	11	6	7	19	33.93	66.07	16.67
		10	37	4	3	4	5	13	26	74.00	25.00
		18	37	3	17	3	3	17	31.48	68.52	0.00
Jackson	11	7	46	8	10	8	9	17	26.98	73.02	12.50
		9	46	6	8	6	7	17	26.98	73 02	16.67
		10	46	5	4	5	6	14	23.33	76.67	20.00
		13	46	4	6	4	4	6	11.54	88.46	0.00
		14	46	4	10	4	4	10	17.86	82.14	0.00
		21	46	3	17	3	3	17	26 98	73 02	0.00
Mitchell	21	14	105	8	7	8	10	21	16.67	83.33	12.50
		15	105	8	15	8	9	15	12.5	87.50	0.00
		21	105	5	0	5	6	21	16.67	83.33	20.00
		26	105	5	25	5	5	25	19 23	80.77	0.00
		35	105	3	0	3	4	35	25	75.00	33 33
		39	105	3	12	3	3	12	10.26	89.74	0.00

[Rest of the table is given in Appendix A]

Table 6.3 Benchmarking with SALOME

Precedence Graph	No of Operations	Total Processing Time	Cycle Time	Optimal No of Stations	SALOME No of Station	Proposed Algorithm Optimum Station	Rel. Dev. From Salome [%]	Total Idle Time	Idle Time (%)	Eff.(%)
Arcus1	83	75707	3786	21	21	24	14.29	15157	16.68	83.32
Arcus1		75707	3985	20	20	23	15.00	16948	17.4	82.60
Arcus1		75707	4206	19	19	21	10.53	12619	14.29	85.71
Arcus1		75707	4454	18	18	20	11.11	13373	15.01	84.99
Arcus1		75707	4732	17	17	18	5.88	9469	11.12	88.88
Arcus1		75707	5048	16	16	16	0.00	5061	6.27	93.73
Arcus1		75707	5408	15	15	16	6.67	10821	12.51	87.49
Arcus1		75707	5824	14	14	15	7.14	11653	13.34	86.66
Arcus1		75707	5853	14	14	15	7.14	12088	13.77	86.23
Arcus1		75707	6309	13	13	13	0.00	6310	7.69	92.31
Arcus1		75707	6842	12	12	12	0.00	6397	7.79	92.21
Arcus1		75707	6883	12	12	12	0.00	6889	8.34	91.66
Arcus1		75707	7571	11	11	11	0.00	7574	9.09	90.91
Arcus1		75707	8412	10	10	10	0.00	8413	10	90.00
Arcus1		75707	8898	9	9	9	0.00	4375	5.46	94.54
Arcus1		75707	10816	8	8	8	0.00	10821	12.51	87.49
Arcus2	111	150399	5755	27	27	32	18.52	33761	18.33	81.67
Arcus2		150399	5785	27	27	33	22.22	40506	21.22	78.78
Arcus2		150399	6016	26	26	30	15.38	34581	19	83.33
Arcus2		150399	6267	25	25	31	24.00	43878	22.59	77.41
Arcus2		150399	6540	24	24	28	16.67	32721	17.87	82.13
Arcus2		150399	6837	23	23	26	13.04	27363	15.39	84.61
Arcus2		150399	7162	22	22	26	18.18	35813	19.23	80.77

[Rest of the table is given in Appendix B]

Table 6.4 Benchmarking with Hybrid GA

Author	No of Operations	Cycle Time	Total Processing Time	Optimal No of Stations	Hybrid GA No of Stations	Proposed Algorithm No Of Station	Total Idle Time	Idle Time (%)	Efficiency	Rel. Dev. From Hybrid GA [%]
Bowman	8	20	75	5	5	5	25	25	75.00	0.00
Merten	7	6	29	6	6	6	7	19.44	80.56	0.00
		7	29	5	5	6	13	30.95	69.05	20.00
		8	29	5	5	5	11	27 5	72.50	0 00
		10	29	3	3	4	11	27.5	72 50	33.33
		15	29	2	2	3	16	35.56	64.44	50.00
		18	29	2	2	2	7	19.44	80.56	0.00
Jaeschke	9	6	37	8	8	8	11	22.92	77.08	0.00
		7	37	7	7	8	19	33.93	66.07	14.29
		8	37	6	6	7	19	33.93	66.07	16 67
		10	37	4	4	5	13	26	74.00	25.00
		18	37	3	3	3	17	31.48	68 52	0.00
Jackson	11	7	46	8	8	9	17	26 98	73.02	12.50
		9	46	6	6	7	17	26.98	73 02	16.67
		10	46	5	5	6	14	23.33	76.67	20.00
		13	46	4	4	4	6	11.54	88.46	0.00
		14	46	4	4	4	10	17.86	82.14	0 00
		21	46	3	3	3	17	26.98	73 02	0.00
Mitchell	21	14	105	8	8	10	21	16.67	83 33	12.50
		15	105	8	8	9	15	12 5	87 50	0.00
		21	105	5	5	6	21	16.67	83.33	20.00
		26	105	5	5	5	25	19.23	80.77	0.00
		35	105	3	3	4	35	25	75.00	33.33
		39	105	3	3	3	12	10.26	89.74	0 00

[Rest of the table is given in Appendix C]

Table 6.5: Fuzzy Task Time

Task	Task Time	Task	Task Time	Task	Task Time	Task	Task Time
J_{1}	0.43 0.5 0 58	J_{21}	1.75 1.8 1.89	J_{41}	1.26 1.3 1.47	J_{61}	0 21 0 3 0.35
J_{2}	0.65 0.7 0 76	J_{22}	0.16 0.2 0.24	J_{42}	0.78 0.8 0.89	J_{62}	0 36 0.4 0.47
J_{3}	0.72 0.8 0 84	J_{23}	0.26 0.3 0.36	J_{43}	0.76 0 8 0 9	J_{63}	0 28 0.3 0.36
J_{4}	0.98 1.1 1 23	J_{24}	0 17 0.2 0.25	J_{44}	0.53 0.6 0.68	J_{64}	0.54 0.6 0.69
J_{5}	0.87 0.9 1 02	J_{25}	0.27 0.3 0.36	J_{45}	0.99 1.1 1.21	J_{65}	0.76 0.8 0.86
J_{6}	0.89 0.9 1.08	J_{26}	1 3 1.5 1.8	J_{46}	0.17 0.2 0.25	J_{66}	0.43 0.5 0 56
J_{7}	0 26 0.3 0.32	J_{27}	0 88 1.0 1.03	J_{47}	0.28 0.3 0.34	J_{67}	0.47 0.5 0 59
J_{8}	0.12 0.2 0.23	J_{28}	1 2 1.3 1.45	J_{48}	0.69 0.7 0.78	J_{68}	1.26 1.3 1 39
J_{9}	0 52 0.6 0.68	J_{29}	0 32 0.4 0.46	J_{49}	0.46 0.5 0.57	J_{69}	0 97 1.0 1.05
J_{10}	0.26 0.3 0 33	J_{30}	0.43 0.5 0.57	J_{50}	1.12 1 2 1.25	J_{70}	0.17 0.2 0.24
J_{11}	0.75 0.8 0.82	J_{31}	1 21 0.3 0.36	J_{51}	0 42 0.5 0.56	J_{71}	0.16 0.2 0 23
J_{12}	0.65 0.7 0.8	J_{32}	1.1 1.2 1.26	J_{52}	0.32 0 4 0 46	J_{72}	0.38 0 4 0.43
J_{13}	0 37 0.4 0.42	J_{33}	1.12 1 2 1.25	J_{53}	0 6 0.65 0.71	J_{73}	0.42 0.5 0.56
J_{14}	0.19 0.2 0.22	J_{34}	0.32 0.4 0.49	J_{54}	0.26 0.3 0 34	J_{74}	0.27 0 3 0.31
J_{15}	0 57 0.6 0.67	J_{35}	0 16 0 2 0.21	J_{55}	0 87 0.9 0.98	J_{75}	0.71 0.8 0.91
J_{16}	0.32 0.4 0.44	J_{36}	0 75 0.8 0.89	J_{56}	0 33 0.4 0.51	J_{76}	0.51 0.55 0 62
J_{17}	0 05 0 1 0.14	J_{37}	0.66 0 7 0 85	J_{57}	0.98 1 0 1.02	J_{77}	0.65 0.7 0.79
J_{18}	0.21 0.3 0.34	J_{38}	0 97 1.0 1.08	J_{58}	1 14 1.2 1.23	J_{78}	0.87 0.9 1 03
J_{19}	0.48 0.5 0.56	J_{39}	0.92 1.0 1.06	J_{59}	0.79 0.8 1 82	J_{79}	0.74 0.8 0 95
J_{20}	0.89 0.9 1.03	J_{40}	0.43 0.5 0.56	J_{60}	0.98 1.0 1 03	J_{80}	0.34 0.4 0 57

Parameters:

Population size: 20

Iteration: 30

Cycle time, C= (11 11 11)

Result:

Optimum Number of Station: 6

Total idle time: (7.11 13.9 17.21)

% of idle time: (10.77 21.06 26.08)

Efficiency: (73.92 78.94 89.23)

Table 6.6 Comparison of Eight Methods on the 70 Task Problems

Cycle Time	Moodie and Young	Tonge MIF	Tonge RC	Tonge BPC	Nevins	Baybars	Sabuncuoglu	Proposed Algorithm
83	48	50	50	49	47	47	48	46
86	47	47	48	47	46	46	46	45
89	44	45	46	44	43	43	44	43
92	43	43	44	43	42	42	43	43
95	42	43	43	41	40	40	41	42
170	24	24	24	23	23	23	23	25
173	24	24	24	23	22	22	23	25
176	22	24	23	22	22	22	22	25
179	22	23	23	21	21	21	22	24
182	22	23	22	21	21	21	22	23
346	11	11	12	11	11	11	11	12
349	11	11	11	11	11	11	11	12
364	11	11	11	11	11	11	11	11

The proposed algorithm has been compared with seven other algorithms [16, 21, 22, 76, 77] in solving Tonge's 70 task problem [17] with 13 cycle times. From above computational results it is clear in some cases proposed algorithm outperforms the other seven methods, even the genetic algorithm based algorithm of Sabuncuoglu [77].Though some results of other algorithms are better than the proposed algorithm, it must be mentioned here that the proposed algorithm holds an objective function with multiple objectives. So it tries to find out the number of the optimum stations having the minimum idle time and minimum smoothness index. So in the other case, it is most likely that the optimum solution of the proposed algorithm may give better result than other algorithms with respect to idle time and smoothness index that is achieved through a trade-off between minimum number of stations and minimum idle time.

6.7 Convergence Analysis

Genetic algorithms use a selection scheme to select individuals from the population to insert into a mating pool. Individuals from the mating pool are used by a recombination operator to generate new offspring, with the resulting offspring forming the basis of the next generation. As the individuals in the mating pool pass their genes on to the next generation it is desirable that the mating pool be comprised of good individuals. A selection scheme in GAs is simply a process that favors the selection of better individuals in the population for the mating pool. The selection pressure is the degree to which the better individuals are favored; the higher the selection pressure, the more the better individuals are favored. This selection pressure drives the GA to improve the population fitness over succeeding generations. The convergence rate of a GA is largely determined by the magnitude of the selection pressure with higher selection pressures resulting in higher convergence rates. Genetic algorithms are able to identify optimal or near-optimal solutions under a wide range of selection pressure.

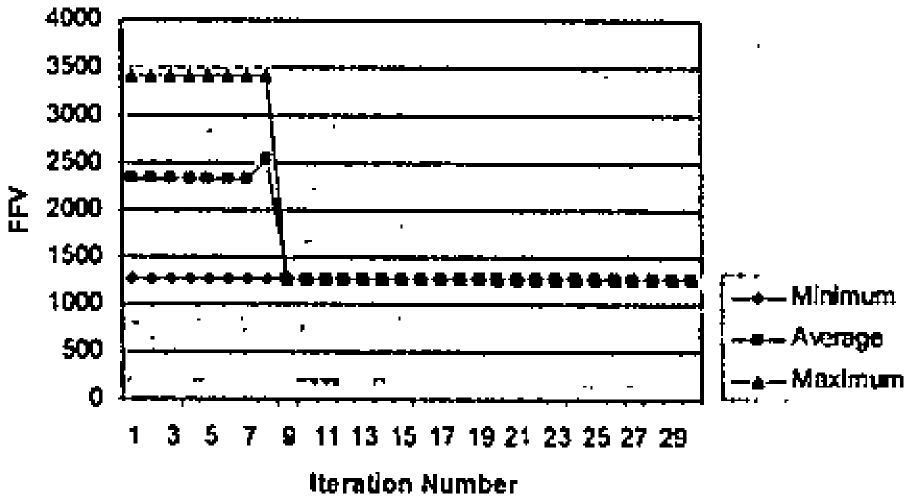
If the selection pressure is too low, the convergence rate will be slow, and the GA will unnecessarily take longer time to find the optimal solution. If the selection pressure is too high, there is an increased chance of the GA prematurely converging to an incorrect (sub-optimal) solution. In addition to providing selection pressure, selection schemes should also preserve population diversity as this helps avoid premature convergence. The proposed algorithm has been run on two options of crossover operator. One crossover operator performs the operation by swapping only one gene. The other crossover operator swaps a cluster of genes. There is an option to control the number of genes to be crossed-over by an user defined rate. The convergence analysis was done on Arcus83 dataset under several parameters. The results are depicted in the following section.

Parameters

Maximum population: 50

Iteration: 30

Crossover method: single gene two point crossover



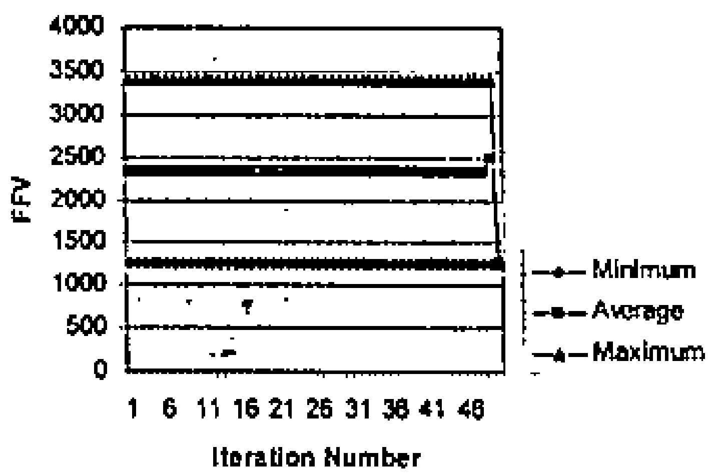
Graph 1: Convergence Analysis

Maximum population: 50

Iteration: 50

Crossover method: clustered gene two point crossover

Crossover rate: 0.001



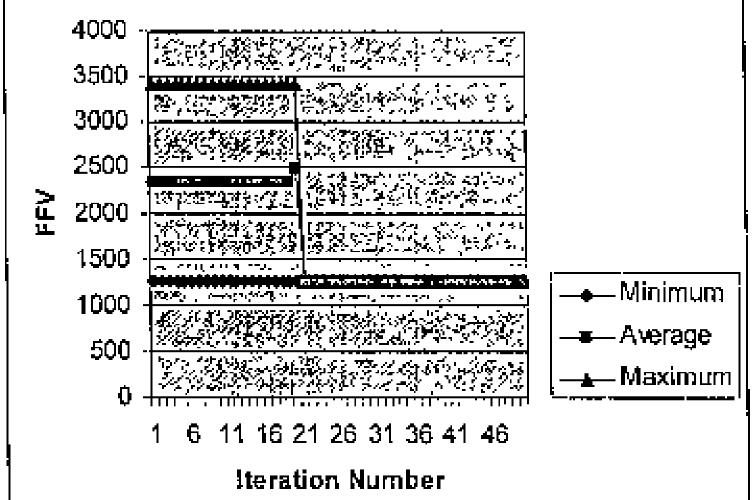
Graph 2: Convergence Analysis

Maximum population: 50

Iteration: 50

Crossover method: clustered gene two point crossover

Crossover rate: 0.001



Graph 3: Convergence Analysis

From the above results, it can be concluded that single gene crossover is better than clustered gene crossover as it converge in smaller number of iterations. The smaller the crossover rate, the lower the number of genes in a cluster and the smaller the number of iterations to converge.

CHAPTER 7

CASE STUDY

7.1 Introduction

SALBP is a classic optimization problem, having been tackled by researchers over several decades. Many algorithms have been proposed for the problem. Yet despite the practical importance of the problem, and the various efforts that have been made to tackle it, little commercially available software is available to help industry in optimizing their lines. It appears that the gap between the available results and their dissemination in today's industry is probably due to a misalignment between the academic LB problem addressed by most of the approaches, and the actual problem being faced by the industry. LB is a difficult optimization problem (even its simplest forms are NP-hard) [18]. So the approach taken by researchers has typically been to simplify it, in order to bring it to a level of complexity amenable to optimization tools. While this is a perfectly valid approach in general, in the particular case of LB it led to some definitions of the problem that ignore many aspects of the real-world problem.

To establish the robustness of the proposed algorithm, it has been implemented to solve a practical line balancing problem. In this case study, the assembly line of the RahimAfrooz Batteries Limited has been studied and balanced applying the algorithm. The data has been collected from a recently conducted time study and fuzzy jib time has been constructed for each specific job.

7.2 Overview of Problem Area

Rahimafrooz Batteries Limited (RBL) is the largest lead-acid battery manufacturer in Bangladesh and offers an extensive range of automotive & specialized industrial battery. It manufactures over 300 different types of automotive and industrial batteries. Its plant is ISO 9001 & ISO 14001 certified. It is one of the key players in South Asia in its field. RBL has also extended its product line to secure power solution with UPS, Rectifier and VRLA Batteries with collaboration of Enersys-USA, Eltek-Norway, AEES-France.

RBL has a successful story of installing solar power in the remote rural areas of Bangladesh. It has successfully installed more than 10,000 home solar systems in the remote rural areas of Bangladesh.

Rahimafrooz has state of the art manufacturing plant. It is equipped with all latest technologies with complete air treatment and lead-recycling management. RBL produces different types of batteries to meet the local and international market.

Its capacity in Automotive Battery is **660,000 (N50)** units per annum and 15 **million AH** of Industrial Battery per annum. All the products are manufactured under strict quality control and ensured by international certifications.

Its main product range includes:

- Automotive battery
- Motorcycle battery
- Appliance battery
- Deep cycle – Flat plate battery
- Industrial tubular battery
- VRLA battery
- UPS
- Solar system

It has different technical collaboration agreements with Lucas Battery Company, U.K, Technical support Group (TSG), Hawker Batteries, UK, Invensys, UK, Hawker Batteries, UK, Eltek – Norway, AEES – France to ensure the quality of battery.

Unit-2 of RBL, in Panisail, Zirani bazaar, Savar, Gazipur, is quite huge and produces mainly automotive batteries. In this thesis data has been collected from unit-2 of RBL to implement the TS-based algorithm.

7.3 Some Important Definitions

Before exploring the specific problem area the following definitions should be noted:

- Active Material: Chemically active compounds in a cell or battery that convert from one composition to another while producing current (electrical energy) or accepting current from an external circuit.
- Battery Polarity: A battery has two poles or posts. The positive battery post is usually marked POS, P, or + and is larger than the negative post which is usually marked NEG, N, or -. The polarity of the charger and battery must match to avoid damage to the battery and charger.
- Cells: The basic electrochemical current producing unit in a battery consisting of a set of positive plates, negative plates, electrolyte, separator and casing. There are six cells in a 12-volt lead-acid battery.
- Container: The polypropylene or hard rubber case that holds plates, electrolyte and separators.
- Electrolyte: A solution of sulfuric acid and water that conducts current through the movement of ions (charged particles in the electrolyte solution) between positive and negative plates. It supplies sulfate ions for reaction with the active material of both positive and negative plates.
- Plates: Flat, typically rectangular components that contain the active material and

a mechanical support structure called a grid, which also has an electrical function, carrying electrons to and from active material. Plates are either positive or negative, depending on the active material that it holds.

7.4 Specific Problem Area: Assembly line in RBL-2

There are two assembly lines in Rahimafrooz batteries ltd. (Unit-2). The configurations of the assembly lines are straight. The products are transferred from one workstation to another over roller conveyor. At each workstation different operations are performed. From a recently conducted time study, it is observed that the workstations cycle time are not equal for each workstations and the line is not balanced. So the proposed algorithm can be applied to the existing assembly line configuration and the new design can be proposed.

7.4.2 Assembly Operations

The assembly of the different types of batteries requires a total of 14 operations. The task required for the assembly operations are:

1. stacking or separating
2. group burning
3. attaching side pack and inserting in the container
4. short and polarity test
5. inter-cell welding
6. shear testing
7. sealing the container
8. burning the pole
9. brushing the pole and making it positive or negative
10. performing the leak test
11. attaching the pass tag and bar code
12. attaching the aluminum foil

13. wrapping the battery with required vent plug

14. packaging

Table 7.1: Precedence Graph

Job (J_i)	*Immediate Predecessor*
1	--
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12
14	13

Table 7.2: Production quantity of Battery

	Type of battery				*Maximum*
	VHD 1500	**PCM 29**	**PCM 27**	**N 120**	
Maximum quantity/shift	245	245	245	285	285

So in 8 hour shift or 480 minutes the workers produce 285 pieces of battery. So it needs (480/285) =1.68 min or 101 second for producing one battery. So the fuzzy cycle time for the process is (101 113 117)

From the conducted time study, the fuzzy job time for each job as follows:

Table 7.3: Fuzzy Task Time

Job (J_i)	*p*	*q*	*r*
1	48.66	52.01	53.66
2	55.66	58.23	62
3	36.66	40.9	47.33
4	18.5	20	23
5	27	30.5	36
6	23	26.8	32
7	18	20	26
8	7	20	30
9	8	10	13
10	20	22.1	28
11	6	8	10
12	18	19.5	26
13	24.5	27	30
14	30	34	37

7.4.3 Result

Applying proposed line balancing algorithm the result is:

Optimum Number of Station: 6

Total idle time: (152.01 288.96 361.02)

% of Idle Time: (21.65 42.62 59.57)

Efficiency: (48.57 57.38 74.92)

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

8.1 Conclusion

This research work has customized a genetic algorithm line balancing algorithm in a fuzzy environment. This algorithm can be employed to balance both the fuzzy and crisp environment. The objective of the algorithm is not only confined to find out the minimum number of station but also to minimize idle time and minimize smoothness index. So the outcome is a highly balanced line with minimum idle time and possible minimum number of stations. The algorithm has been benchmarked with two well known algorithms namely 'EUREKA' [71] and 'SALOME' [72]. The result shows that the proposed algorithm shows a small amount of relative deviation. The algorithm has also been compared with eight different algorithms for solving assembly line balancing problem method.

It must be mentioned that all of the previous algorithm have the objective of finding out the minimum number of stations. The proposed algorithm has multi objective and it tries to find out the optimum combination of minimum number of stations with minimum idle time and smoothness index.

8.2 Recommendations

The performance of the presented GA-based algorithm can be further analyzed in terms of CPU time by comparing it with other best known algorithms for assembly line balancing problem In the crossover operator, order crossover method has been used. Other crossover operator can be used and compared with the proposed algorithm to observe the result. If the some of the stations can be frozen after a specific number of iterations then the computation time will be lower. In this case the fuzzy time is considered Triangular fuzzy number. The algorithm can also be used to handle other type of fuzzy numbers.

REFERENCE

[1] Baybars, I., "A survey of exact algorithms for the simple assembly line balancing problem". *Management Science*, Vol. 32, pp. 909–932, 1986

[2] Scholl, A., "Balancing and sequencing assembly lines", 2nd ed. Physica, Heidelberg, 1999.

[3] Talbot, F.B., Patterson, J.H., Gehrlein, W.V., "A comparative evaluation of heuristic line balancing techniques". *Management Science*, Vol. 32, pp. 430–454, 1986.

[4] Boctor, F.F., "A multiple-rule heuristic for assembly line balancing". *Journal of the Operational Research Society*, Vol. 46, pp. 62–69, 1995.

[5] Ponnambalam, S., Aravindan, G., Mogileeswar Naidu, G., "A multi-objective genetic algorithm for solving assembly line balancing problem". *International Journal of Advanced Manufacturing Technology*, Vol. 16, pp. 341–352, 2000.

[6] Hackman, S.T., Magazine, M.J., Wee, T.S., "Fast, effective algorithms for simple assembly line balancing problems". *Operations Research*, Vol. 37, pp. 916–924, 1989.

[7] Scholl, A., Voß, S., "Simple assembly line balancing—Heuristic approaches". *Journal of Heuristics*, Vol. 2, pp. 217–244, 1996.

[8] Helgeson, W.P., Birnie, D.P., "Assembly line balancing using the ranked positional weight technique", *Journal of Industrial engineering*, Vol. 12, pp. 394-398, 1961.

[9] Wee, T.S., Magazine, M.J., "Assembly line balancing as generalized bin packing". *Operations Research Letters*, Vol. 1, No. 2, pp. 56–58. 1982.

[10] Arcus, A.L., "COMSOAL: A computer method of sequencing operations for assembly lines". *International Journal of Production Research*, Vol.4, pp. 259-277, 1966.

[11] Bonney, M.C., Schofield, N.A., Green, A., "Assembly line balancing and mixed model line sequencing". *Proceedings of the Fifth INTERNET World Congress*, Birmingham, pp.112–121, 1976.

[12] Schofield, N.A., "Assembly line balancing and the application of computer techniques". *Computers and Industrial Engineering*, Vol. 3, pp. 53–69, 1979.

[13] Tonge, F.M., "Assembly line balancing using probabilistic combinations of heuristics". *Management Science*, Vol. 11, pp. 727–735, 1965.

[14] Bennett, G.B., Byrd, J., "A trainable heuristic procedure for the assembly line balancing problem". *AIIE Transactions*, Vol. 8, pp. 195–201, 1976.

[15] Raouf, A., Tsui, C.L., El-Sayed, E.A., "A new heuristic approach to assembly line balancing". *Computers and Industrial Engineering*, Vol. 4, pp. 223–234, 1980.

[16] Baybars, I., "An efficient heuristic method for the simple assembly line balancing problem". *International Journal of Production Research*, Vol. 24, pp. 149–166, 1986

[17] Tonge, F.M., "Summary of a heuristic line balancing procedure" *Management Science*, Vol. 7, pp. 21–39, 1960.

[18] Freeman, D.S., Swain, R.W., "A heuristic technique for balancing automated assembly lines". *Research Report No. 86-6*, Industrial and Systems Engineering Department, University of Florida, Gainesville, 1986.

[19] Fleszar, K., Hindi, K.S., "An enumerative heuristic and reduction methods for the assembly line balancing problem". *European Journal of Operational Research*, Vol. 145, pp. 606–620, 2003.

[20] Pinto, P.A., Dannenbring, D.G., Khumawala, B.M., "A heuristic network procedure for the assembly line balancing problem". *Naval Research Logistics Quarterly*, Vol. 25, pp. 299–307, 1978.

[21] Tonge, F.M., "A heuristic program of assembly line balancing", Englewood Cliffs, NJ, 1961.

[22] Moodie, C.L., Young, H.H., A heuristic method of assembly line balancing for assumptions of constant or variable work element times. *Journal of Industrial Engineering*, Vol. 16, pp. 23–29, 1965.

[23] Arcus, A.L., "An analysis of a computer method of Sequencing Assembly Line Operations", Ph.D. dissertation, University of California, Berkeley, 1963.

[24] Hoffmann, T.R., "Assembly line balancing with a precedence matrix". *Management Science,* Vol. 9, pp. 551–562, 1963.

[25] Gehrlein, W.V., Patterson, J.H., "Sequencing for assembly lines with integer task times". *Management Science*, Vol. 21, pp. 1064–1070, 1975.

[26] Dar-El, E.M.,"MALB- A heuristic technique for balancing large-scale single-model assembly lines"., *AIEE Transactions.*, Vol 5, No. 4, pp 343-356, 1973.

[27] Dar-El. E.M., "Assembly line balancing - an improvement on the ranked positional weight technique", *Journal of Industrial Engineering*, Vol. 15, No. 2, pp. 73-77, 1964.

[28] Wee, T. S. and Magazine, M., "An efficient branch and bound algorithm for an assembly line balancing problem – part I: minimize the number of work stations", University of Waterloo, Ontario, Canada. 1981.

[29] Talbot, F.B., Patterson, J.H., "An integer programming algorithm with network cuts for solving the assembly line balancing problem". *Management Science*, Vol. 30, pp. 85–99, 1984.

[30] Schrage, L., Baker, K.R., "Dynamic programming solution of sequencing problems with precedence constraints". *Operations Research*, Vol. 26, pp. 444–449, 1978.

[31] Held, M., Karp, R.M., Shareshian, R., "Assembly line balancing—Dynamic programming with precedence constraints". *Operations Research*, Vol. 11, pp. 442–459, 1963.

[32] Dar-El, E.M., Rubinovitch, Y., "MUST—A multiple solutions technique for balancing single model assembly lines. *Management Science*, Vol. 25, pp. 1105–1114, 1979.

[33] Glover, F., Laguna, M., "Tabu search". Kluwer Academic Publishers, Boston, 1997.

[34] Aarts, E.H.L., Korst, J.H.M., "Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing". Wiley, Chichester, 1989.

[35] Chiang, W.-C., "The application of a tabu search metaheuristic to the assembly line balancing problem". *Annals of Operations Research*, Vol. 77, pp. 209–227, 1998.

[36] Lapierre, S.D., Ruiz, A., Soriano, P., "Balancing assembly lines with tabu search". *European Journal of Operational Research*. Vol. 168, pp 666-693, 2006.

[37] Heinrici, A., "A comparison between simulated annealing and tabu search with an example from the production planning". In: Dyckhoff, H. et al. (Eds.), *Operations Research Proceedings*, Springer, Berlin, pp. 498–503, 1994.

[38] Suresh, G., Sahu, S., "Stochastic assembly line balancing using simulated annealing". *International Journal of Production Research*, Vol. 32, pp. 1801–1810, 1994.

[39] McMullen, P.R., Frazier, G.V., "Using simulated annealing to solve a multi-objective assembly line balancing problem with parallel workstations". *International Journal of Production Research*, Vol. 36, pp. 2717–2741, 1998.

[40] Bautista, J., Pereira, J., Ant algorithms for assembly line balancing. *Lecture Notes in Computer Science* 2463, pp. 65–75, 2002.

[41] McMullen, P.R., Tarasewich, P., "Using ant techniques to solve the assembly line balancing problem". *IIE Transactions*, Vol. 35, pp. 605–617, 2003.

[42] Holland, J. H., "Adaptation in natural and artificial systems", The University of Michigan Press, Ann Arbor, MI, 1975.

[43] Goldberg, D.E., "Genetic algorithms in search, optimization and machine learning". Addison-Wesley, Reading, MA, 1989.

[44] Leu, Y.Y., Matheson, L.A., Rees, L.P., "Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria". *Decision Sciences*, Vol. 25, pp. 581–606, 1994

[45] Anderson, E.J., Ferris, M.C., "Genetic algorithms for combinatorial optimization: The assembly line balancing problem". *ORSA Journal on Computing*, Vol. 6, pp. 161–173, 1994.

[46] Suresh, G., Vinod, V.V., Sahu, S., "Genetic algorithm for assembly line balancing". *Production Planning and Control*, Vol. 7, pp. 38–46, 1996.

[47] Zadeh, L.A., "Fuzzy Sets", *Information and Control*, Vol. 8, pp 338-353, 1965.

[48] Tsujimura, Y., Gen, M., Kubota, E., "Solving fuzzy assembly-line balancing problem with genetic algorithms". *Computers and Industrial Engineering*, Vol. 29, pp. 543–547, 1995.

[49] Deckro, R.F, "Balancing cycle time and workstations", *IIE Transactions*, Vol. 21, pp. 106-111, 1989.

[50] Scholl, A., Becker, C., "State-of-the-art exact and heuristic solution procedures for simple assembly line balancing", *Jenaer Schriften zur Wirtschaftswissenschaft*. Vol. 20, University of Jena, Germany, 2003 .

[51] Rosenberg, O., Ziegler, H., A comparison of heuristic algorithms for cost-oriented assembly line balancing", *Operations Research*, Vol. 36, pp. 477-495, 1992.

[52] Amen, M., "An exact method for cost-oriented assembly line balancing". *International Journal of Production Economics*, Vol. 64, pp. 187-195, 2000

[53] Scholl, A., Becker, C., "A note on "An exact method for cost-oriented assembly line balancing". *Jenaer Schriften zur Wirtschaftswissenschaft*, Vol. 22, University of Jena, Germany, 2003

[54] Amen, M., "Heuristic methods for cost-oriented assembly line balancing: A survey". *International Journal of Production Economics*, Vol.68, pp.1-14, 2000

[55] Malakooti, B., "Assembly line balancing with buffers by multiple criteria optimization", *International Journal of Production Research*, Vol. 32, pp. 2159-2178, 1994.

[56] Malakooti, B., Kumar, A., "An expert system for solving multi-objective assembly line balancing problems", *International Journal of Production Research*, Vol. 34, pp. 2533-2552, 1996.

[57] Rosenblatt, M.J., Carlson, R.C., "Designing a production line to maximize profit", *IIE Transactions*, Vol. 17, pp. 117- 121, 1985.

[58] Martin, G.E., "Optimal design of production lines". *International Journal of Production Research*, Vol. 32, pp. 989 -1000, 1994.

[59] E.H. Mamdani, S. Assilian,"An experiment in linguistic synthesis with a fuzzy logic controller". *International journal of Man-Machine Studies*, Vol. 7, No.1, pp. 1-13, 1975.

[60] Roy, B., Vincke, "Relational systems of preference with one or more pseudo-criteria: Some new concepts and results". *Management Science*, Vol. 30, pp. 1323–1335, 1984.

[61] Vanegas, L.V., Labib, A.W., "Application of new fuzzy weighted average method to engineering design evaluation". *International Journal of Production Research*, Vol. 39, No. 6, pp. 1147–1162, 2001.

[62] Yao, J.S., Wu, K., "Ranking fuzzy numbers based on decomposition principle and signed distance". *Fuzzy Sets and Systems*, Vol. 116, pp. 275–288, 2000.

[63] Nils Aall Barricelli. "Symbiogenetic evolution processes realized by artificial methods". *Methodos*, Vol. 9, pp. 35-36, 1957.

[64] Alex S. Fraser, "Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction." *Australian Journal of Biological Sciences*, Vol. 10, pp. 484-491, 1957.

[65] Fraser, Alex and Donald Burnell, "Computer Models in Genetics", McGraw-Hill, New York, 1970.

[66] Crosby, Jack L., *Computer Simulation in Genetics,* John Wiley & Sons, London, 1973.

[67] Bautista, J., Suarez, R., Mateo, M., Companys, R., "Local search heuristics for the assembly line balancing problem with incompatibilities between tasks." *Proceedings of the 2000 IEEE International Conference on Robotics and Automation,* San Francisco, CA, pp. 2404–2409, 2000.

[68] Kim, Y.K., Kim, Y., Kim, Y.J., Two-sided assembly line balancing: A genetic algorithm approach. *Production Planning and Control*, Vol. 11, pp. 44–53, 2000.

[69] Davis, L., "Applying algorithms to epistatic domains". *Proceedings of International Joint Conference on Artificial Intelligence*, 1985.

[70] De Jong, K. A., "An analysis of the behavior of a class of genetic adaptive systems". Doctoral Dissertation, University of Michigan, 1975.

[71] Hoffmann, T.R., "EUREKA: A Hybrid System for Assembly Line Balancing." *Management Science*, Vol. 38, pp. 39–47, 1992

[72] Scholl, A., Klein, R., "SALOME: A bidirectional branch and bound procedure for assembly line balancing". *INFORMS Journal on Computing*, Vol. 9, pp. 319–334, 1997.

[73] Goncalves, J. F., Almeida, J.R., "A hybrid genetic algorithm for assembly line balancing". *Journal of Heuristics*, Vol. 8, pp. 629-642, 2002.

[74] Hoffmann, T.R., "Assembly Line Balancing: A Set of Challenging Problems." *International Journal of Production Research*, Vol. 28, pp. 1807–1815, 1990.

[75] Scholl, A., "Data of Assembly Line Balancing Problems", Working Paper, TH Darmstadt, 1993.

[76] Nevins, A. J. "Assembly line balancing using best bud search". *Management Science*, Vol. 18, pp. 529-539, 1972.

[77] Sabuncuoglu, I., Erel, E., Tanyer, M., "Assembly line balancing using genetic algorithms". *Journal of Intelligent Manufacturing*, Vol. 11, pp. 295–310, 2000.

Appendices

Appendix A: Benchmark Data with EUREKA Table 6.2 (continued)

Author	No of operations	Cycle Time	Total Processing Time	Optimal No of Stations	Ideal Minimum Idle Time	EUREKA no of station	Proposed Algorithm no of station	Total Idle Time	% of Idle Time	Efficiency	rel. dev. from EUREKA [%]
Heskia	28	138	1024	8	80	8	9	218	17.55	82.45	12 50
		205	1024	5	1	5	6	206	16.75	83.25	20.00
		216	1024	5	56	5	6	56	5 19	94.81	0.00
		256	1024	4	0	4	5	256	20	80.00	25.00
		324	1024	4	272	4	4	272	20.99	79 01	0.00
		342	1024	3	2	3	4	344	25.15	74.85	33 33
Sawyer	30	25	324	14	26	14	17	101	23.76	76.24	21.43
		27	324	13	27	13	16	108	25	75.00	23.08
		30	324	12	36	12	14	96	22 86	77.14	16.67
		36	324	10	36	10	11	72	18.18	81 82	10 00
		41	324	8	4	8	9	45	12 2	87.80	12.50
		54	324	7	54	7	7	54	14.29	85.71	0.00
		75	324	5	51	5	5	51	13.6	86.40	0.00
Kilbridge and Wester	45	57	552	10	18	10	11	75	11.96	88 04	10 00
		79	552	7	1	7	8	80	12.6	87.34	14.29
		92	552	6	0	6	7	92	14.29	85.71	16.67
		110	552	6	108	6	6	108	16 36	83.64	0.00
		138	552	4	0	4	5	138	20	80.00	25 00
		184	552	3	0	3	4	184	25	75 00	33.33
Tonge	70	176	3510	21	186	21	23	538	13.29	86.71	9.52
		364	3510	10	130	10	11	494	12.34	87.66	10.00
		410	3510	9	180	9	9	180	4.88	95.12	0.00
		468	3510	8	234	8	8	234	6.25	93 75	0.00

Table 6.2 (continued)

Author	No of operations	Cycle Time	Total Processing Time	Optimal No of Stations	Ideal Minimum Idle Time	EUREKA no of station	Proposed Algorithm no of station	Total Idle Time	% of Idle Time	Efficiency	rel. dev. from EUREKA [%]
		527	3510	7	179	7	7	706	16.75	95.15	0.00
Arcus	83	5048	75707	16	5061	17	17	10109	11.78	88.22	6.25
		5853	75707	14	6235	15	15	12088	13.77	86 23	7 14
		6842	75707	12	6397	12	12	6397	7.79	92 21	0.00
		7571	75707	11	7574	12	12	15145	16.67	83.33	9.09
		8414	75707	10	8433	10	10	8433	10.02	89.98	0.00
		8998	75707	9	5275	9	9	5275	6.51	93 49	0 00
		10816	75707	8	10821	8	8	10821	12.51	87.49	0.00
Arcus	111	5755	150399	27	4986	33	33	39516	20.81	79.19	17.86
		8847	150399	18	8847	20	20	26541	15	85.00	11.11
		10027	150399	16	10033	18	17	20060	11.77	88.23	6.25
		10743	150399	15	10746	16	16	21489	12.5	87 50	6 67
		11378	150399	14	8893	15	15	20271	11.88	88.12	7.14
		17067	150399	9	3204	10	10	20271	11.88	88.12	11 11
									Average Relative Deviation (%)		11.50

Appendix B: Benchmark Data with SALOME Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal no of stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Arcus2		150399	7520	21	21	23	9.52	22561	13.04	86.96
Arcus2		150399	7916	20	20	23	15 00	31669	18	82.61
Arcus2		150399	8356	19	19	22	15.79	33433	18.19	81.81
Arcus2		150399	8847	18	18	20	11.11	26541	15	85.00
Arcus2		150399	9400	17	17	19	11.76	28201	15.79	84.21
Arcus2		150399	10027	16	16	19	18.75	40114	21.06	78.94
Arcus2		150399	10743	15	15	16	6.67	21489	12.5	87.50
Arcus2		150399	11378	14	14	15	7.14	20271	11.88	88 12
Arcus2		150399	11570	13	13	15	15.38	23151	13.34	86 66
Arcus2		150399	17067	9	9	10	11 11	20271	11.88	88.12
Barthol2	148	4234	84	51	51	63	23 53	1058	19.99	80.01
Barthol2		4234	85	50	51	63	23 53	1121	20.93	79.07
Barthol2		4234	87	49	49	63	28.57	1247	22.75	77.25
Barthol2		4234	89	48	49	60	22.45	1106	20.71	79.29
Barthol2		4234	91	47	48	60	25.00	1226	22.45	77.55
Barthol2		4234	93	46	47	59	25.53	1253	22.84	77 16
Barthol2		4234	95	45	46	58	26.09	1276	23 16	76.84
Barthol2		4234	97	44	45	55	22.22	1101	20.64	79.36
Barthol2		4234	99	43	44	53	20 45	1013	19.31	80.69
Barthol2		4234	101	42	42	53	26.19	1119	20 9	79.10
Barthol2		4234	104	41	41	52	26 83	1174	21.71	78.29
Barthol2		4234	106	40	40	50	25.00	1066	20.11	79.89
Barthol2		4234	109	39	39	48	23.08	998	19.07	80 93
Barthol2		4234	112	38	38	46	21.05	918	17.82	82 18
Barthol2		4234	115	37	37	45	21.62	941	18 18	81.82
Barthol2		4234	118	36	36	43	19.44	840	16.55	83.45
Barthol2		4234	121	35	35	42	20.00	848	16.69	83.31

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Barthol2		4234	125	34	34	41	20.59	891	17.39	82.61
Barthol2		4234	129	33	33	40	21.21	926	17.95	82.05
Barthol2		4234	133	32	32	39	21.88	953	18.37	81.63
Barthol2		4234	137	31	31	38	22.58	972	18 67	81.33
Barthol2		4234	142	30	30	36	20.00	878	17.18	82 82
Barthol2		4234	146	29	29	34	17.24	730	14 71	85.29
Barthol2		4234	152	28	28	32	14.29	630	12 95	87.05
Barthol2		4234	157	27	27	31	14.81	633	13.01	86.99
Barthol2		4234	163	26	26	29	11.54	493	10.43	89.57
Barthol2		4234	170	25	25	28	12.00	526	11.05	88.95
Barthold	148	5634	403	14	14	16	14.29	814	12.62	87.38
Barthold		5634	434	13	13	14	7.69	442	7.27	92 73
Barthold		5634	470	12	12	13	8 33	476	7.79	92.21
Barthold		5634	513	11	11	12	9 09	522	8.48	91.52
Barthold		5634	564	10	10	11	10 00	570	9.19	90.81
Barthold		5634	626	9	9	10	11.11	626	10	90.00
Barthold		5634	705	8	8	9	12.50	711	11.21	88.79
Barthold		5634	805	7	7	8	14.29	806	12.52	87.48
Bowman	8	75	20	5	5	5	0 00	25	25	75.00
Buxey	29	324	27	13	13	15	15.38	81	20	80 00
Buxey		324	30	12	12	14	16.67	96	22 86	77.14
Buxey		324	33	11	11	12	9.09	72	18.18	81.82
Buxey		324	36	10	10	10	0.00	36	10	90.00
Buxey		324	41	8	8	9	12 50	45	12 2	87.80
Buxey		324	47	7	7	8	14.29	52	13.83	86.17
Buxey		324	54	7	7	7	0 00	54	14 29	85.71

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Gunther	35	483	41	14	14	16	14.29	173	26.37	73.63
Gunther		483	44	12	12	15	25.00	177	26.82	73.18
Gunther		483	49	11	11	12	9.09	105	17.86	82.14
Gunther		483	54	9	9	11	22.22	111	18.69	81.31
Gunther		483	61	9	9	9	0.00	66	12.02	87.98
Gunther		483	69	8	8	8	0.00	69	12.5	87.50
Gunther		483	81	7	7	7	0.00	84	14.81	85.19
Hahn	53	14026	2004	8	8	9	12 50	4010	22.23	77.77
Hahn		14026	2338	7	7	8	14.29	4678	25.01	74.99
Hahn		14026	2806	6	6	6	0.00	2810	16.69	83.31
Hahn		14026	3507	5	5	5	0.00	3509	20.01	79.99
Hahn		14026	4676	4	4	4	0.00	4678	25.01	74.99
Heskiaoff	28	1024	138	8	8	10	25.00	356	25.8	74.20
Heskiaoff		1024	205	5	5	6	20.00	206	16.75	83 25
Heskiaoff		1024	216	5	5	6	20 00	272	20.99	79 01
Heskiaoff		1024	256	4	4	5	25 00	256	20	80.00
Heskiaoff		1024	324	4	4	4	0 00	272	20.99	79.01
Heskiaoff		1024	342	3	3	4	33.33	344	25.15	74 85
Jackson	11	46	7	8	8	9	12.50	17	26.98	73.02
Jackson		46	9	6	6	7	16.67	17	26.98	73 02
Jackson		46	10	5	5	6	20.00	14	23.33	76.67
Jackson		46	13	4	4	4	0.00	6	11.54	88 46
Jackson		46	14	4	4	4	0 00	10	17.86	82 14
Jackson		46	21	3	3	3	0.00	17	26.98	73.02
Jaeschke	9	37	6	8	8	8	0 00	11	22.92	77 08
Jaeschke		37	7	7	7	8	14.29	19	33.93	66.07
Jaeschke		37	8	6	6	7	16.67	19	33.93	66.07
Jaeschke		37	10	4	4	5	25.00	13	26	74.00

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Jaeschke		37	18	3	3	3	0.00	17	31.48	68.52
Kilbridge	45	57	56	10	10	12	20.00	120	17.86	8.48
Kilbridge		57	57	10	10	11	10.00	75	11.96	9.09
Kilbridge		57	62	9	9	11	22.22	130	19.96	8.36
Kilbridge		57	69	8	8	9	12.50	69	11.11	9.18
Kilbridge		57	79	7	7	8	14.29	80	12.66	9.02
Kilbridge		57	92	6	6	7	16.67	92	14.29	8.85
Kilbridge		57	110	6	6	6	0.00	108	16.36	8.64
Kilbridge		57	111	5	5	6	20.00	114	17.12	8.56
Kilbridge		57	138	4	4	5	25.00	138	20	8.26
Kilbridge		57	184	3	3	4	33.33	184	25	7.74
Lutz1	32	14140	1414	11	11	12	9.09	2828	16.67	83.33
Lutz1		14140	1572	10	10	11	10.00	3152	18.23	81.77
Lutz1		14140	1768	9	9	9	0.00	1772	11.14	88.86
Lutz1		14140	2020	8	8	8	0.00	2020	12.5	87.50
Lutz1		14140	2357	7	7	7	0.00	2359	14.3	85.70
Lutz1		14140	2828	6	6	6	0.00	2828	16.67	83.33
Lutz2	89	485	11	49	49	61	24.49	186	27.72	72.28
Lutz2		485	12	44	44	57	29.55	199	29.09	70.91
Lutz2		485	13	40	40	52	30.00	191	28.25	71.75
Lutz2		485	14	37	37	49	32.43	201	29.3	70.70
Lutz2		485	15	34	34	46	35.29	205	29.71	70.29
Lutz2		485	16	31	31	42	35.48	187	27.83	72.17
Lutz2		485	17	29	29	40	37.93	195	28.68	71.32
Lutz2		485	18	28	28	33	17.86	109	18.35	81.65
Lutz2		485	19	26	26	31	19.23	104	31.66	82.34
Lutz2		485	20	25	25	29	16.00	95	16.38	83.62

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Lutz2		485	21	24	24	28	16 67	103	17.52	82.48
Lutz3	89	1644	75	23	23	25	8.70	231	12.32	87.68
Lutz3		1644	79	22	22	25	13 64	231	12.32	83.24
Lutz3		1644	83	21	21	25	19.05	331	16.76	79.23
Lutz3		1644	87	20	20	23	15.00	357	17.84	82.16
Lutz3		1644	92	19	19	21	10.53	288	14.91	85.09
Lutz3		1644	97	18	18	20	11.11	296	15.26	84.74
Lutz3		1644	103	17	17	19	11.76	313	15.99	84.01
Lutz3		1644	110	15	15	18	20.00	336	16.97	83.03
Lutz3		1644	118	14	14	16	14.29	244	12.92	87.08
Lutz3		1644	127	14	14	15	7 14	261	13.7	86.30
Lutz3		1644	137	13	13	14	7.69	274	14 29	85.71
Lutz3		1644	150	12	12	12	0.00	156	8.67	91 33
Mansoor	11	185	48	4	4	5	25.00	55	22 92	77.08
Mansoor		185	62	3	3	4	33 33	63	25.4	74.60
Mansoor		185	94	2	2	3	50 00	97	34.4	65 60
Mertens	7	29	6	6	6	6	0.00	7	19 44	80 56
Mertens		29	7	5	5	6	20.00	13	30 95	69.05
Mertens		29	8	5	5	5	0.00	11	27.5	72 50
Mertens		29	10	3	3	4	33.33	11	27 5	72 50
Mertens		29	15	2	2	3	50.00	16	35.56	64.44
Mertens		29	18	2	2	2	0.00	7	19.44	80 56
Mitchell	21	105	14	8	8	10	25.00	35	25	75.00
Mitchell		105	15	8	8	8	0.00	15	12.5	87.50
Mitchell		105	21	5	5	6	20.00	21	16.67	83.33
Mitchell		105	26	5	5	5	0.00	25	19.23	80.77
Mitchell		105	35	3	3	4	33.33	35	25	75.00

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Mitchell		105	39	3	3	3	0.00	12	10.26	89.74
Mukherje	94	4208	176	25	25	29	16.00	896	17.55	82.45
Mukherje		4208	183	24	24	28	16 67	916	17.88	82.12
Mukherje		4208	192	23	23	26	13 04	784	15.71	84.29
Mukherje		4208	201	22	22	25	13.64	817	16.26	83.74
Mukherje		4208	211	21	21	24	14.29	856	16.9	83.10
Mukherje		4208	222	20	20	22	10.00	676	13.84	86 16
Mukherje		4208	234	19	19	22	15.79	940	18.26	81 74
Mukherje		4208	248	18	18	20	11.11	752	15.16	84 84
Mukherje		4208	263	17	17	18	5.88	526	11 11	88.89
Mukherje		4208	281	16	16	17	6.25	569	11 21	88.09
Mukherje		4208	301	15	15	16	6.67	608	12.62	87.38
Mukherje		4208	324	14	14	14	0.00	328	7.23	92.77
Mukherje		4208	351	13	13	13	0.00	355	7.78	92.22
Roszieg	25	125	14	10	10	12	20 00	43	25 6	74.40
Roszieg		125	16	8	8	10	25.00	35	21.88	78.13
Roszieg		125	18	8	8	9	12.50	37	22.84	77.16
Roszieg		125	21	6	6	8	33.33	43	25.6	74 40
Roszieg		125	25	6	6	6	0 00	25	16 67	83 33
Roszieg		125	32	4	4	5	25.00	35	21 88	78.13
Sawyer	30	324	25	14	14	17	21.43	101	23.76	76 24
Sawyer		324	27	13	13	16	23.08	108	25	75.00
Sawyer		324	30	12	12	14	16.67	96	22.86	77.14
Sawyer		324	33	11	11	13	18 18	105	24.48	75.52
Sawyer		324	36	10	10	12	20.00	108	25	75.00
Sawyer		324	41	8	8	10	25.00	86	20.98	79 02
Sawyer		324	47	7	7	8	14.29	52	13.83	86.17
Sawyer		324	54	7	7	7	0.00	54	14 29	85.71

Table 6.3 (continued)

Precedence Graph	No of Operations	Total Processing Time	Cycle Time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Sawyer		324	75	5	5	5	0.00	51	13.6	86.40
Scholl	297	69655	1394	50	51	59	15.69	12591	15.31	84.69
Scholl		69655	1422	50	50	57	14.00	11399	14.06	85.94
Scholl		69655	1452	48	48	55	14.58	10205	12.78	87.22
Scholl		69655	1483	47	47	54	14.89	10427	13.02	86.98
Scholl		69655	1515	46	46	52	13.04	9125	11.58	88.42
Scholl		69655	1548	46	46	52	13.04	10841	13.47	86.53
Scholl		69655	1584	44	44	51	15.91	11129	13.78	86.22
Scholl		69655	1620	44	44	50	13.64	11345	14.01	85.99
Scholl		69655	1659	42	42	48	14.29	9977	12.53	87.47
Scholl		69655	1699	42	42	46	9.52	8499	10.87	89.13
Scholl		69655	1742	40	40	45	12.50	8735	11.14	88.86
Scholl		69655	1787	39	39	44	12.82	8973	11.41	88.59
Scholl		69655	1834	38	38	42	10.53	7373	9.57	90.43
Scholl		69655	1883	37	37	41	10.81	7548	9.78	90.22
Scholl		69655	1935	36	36	40	11.11	7745	10.01	89.99
Scholl		69655	1991	35	35	38	8.57	6003	7.93	92.07
Scholl		69655	2049	34	34	37	8.82	6158	8.12	91.88
Scholl		69655	2111	33	33	36	9.09	6341	8.34	91.66
Scholl		69655	2177	32	32	35	9.38	6540	8.58	91.42
Scholl		69655	2247	31	31	34	9.68	6743	8.83	91.17
Scholl		69655	2322	30	30	33	10.00	6971	9.1-	90.90
Scholl		69655	2402	29	29	31	6.90	4807	6.46	93.54
Scholl		69655	2488	28	28	31	10.71	7473	9.69	90.31
Scholl		69655	2580	27	27	29	7.41	5165	6.9	93.10
Scholl		69655	2680	26	26	28	7.69	5385	7.13	92.82
Sawyer		324	75	5	5	5	0.00	51	13.6	86.40
Scholl		69655	2787	25	25	27	8.00	5594	7.43	92.57

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. from SALOME [%]	Total Idle Time	% of Idle Time	Eff.
Tonge	70	3510	160	23	23	26	13.04	650	15.63	84 38
Tonge		3510	168	22	22	25	13.64	690	16.43	83.57
Tonge		3510	176	21	21	25	19.05	890	20 23	79.77
Tonge		3510	185	20	20	23	15.00	745	17.51	82.49
Tonge		3510	195	19	19	22	15.79	780	18.18	81.82
Tonge		3510	207	18	18	21	16.67	837	19.25	80.75
Tonge		3510	220	17	17	19	11.76	670	16.03	83.97
Tonge		3510	234	16	16	18	12.50	702	16.67	83.33
Tonge		3510	251	14	14	16	14.29	506	12.6	87.40
Tonge		3510	270	14	14	15	7.14	540	13 33	86.67
Tonge		3510	293	13	13	14	7 69	592	14.43	85.57
Tonge		3510	320	11	11	12	9 09	330	8 59	91.41
Tonge		3510	364	10	10	11	10.00	494	12.34	87.66
Tonge		3510	410	9	9	9	0 00	180	4 88	95.12
Tonge		3510	468	8	8	8	0.00	234	6.25	93.75
Tonge		3510	527	7	7	7	0.00	179	4.85	95.15
Wamecke	58	1548	54	31	31	37	19.35	450	22 52	77.48
Wamecke		1548	56	29	29	35	20.69	412	21.02	78.98
Wamecke		1548	58	29	29	36	24.14	540	25.86	74.14
Wamecke		1548	60	27	27	35	29.63	552	26.29	73.71
Wamecke		1548	62	27	27	34	25.93	560	26.57	73.43
Wamecke		1548	65	25	25	31	24 00	467	23.18	76.82
Wamecke		1548	68	24	24	30	25 00	492	24 12	75.88
Wamecke		1548	71	23	23	28	21.74	440	22.13	77 87
Wamecke		1548	74	22	22	28	27.27	524	25.29	74 71
Wamecke		1548	78	21	21	24	14.29	324	17.31	82.69
Wamecke		1548	82	20	20	23	15.00	338	17.92	82.08

Table 6.3 (continued)

Precedence graph	No of Operations	total processing time	cycle time	optimal # stations	SALOME no of station	Prop. Algorithm No of Station	rel. dev. SALOME [%]	Total Idle Time	% of Idle Time
Warnecke		1548	86	19	19	23	21.05	430	21.74
Warnecke		1548	92	17	17	21	23.53	384	19.88
Warnecke		1548	97	17	17	19	11.76	295	16.01
Warnecke		1548	104	15	15	18	20.00	324	17.31
Warnecke		1548	111	14	14	16	14.29	228	12.84
Wee-mag	75	1499	28	63	63	66	4.76	349	18.89
Wee-mag		1499	29	63	63	65	3.17	386	20.48
Wee-mag		1499	30	62	62	64	3.23	421	21.93
Wee-mag		1499	31	62	62	64	3.23	485	24.45
Wee-mag		1499	32	61	61	63	3.28	517	25.64
Wee-mag		1499	33	61	61	63	3.28	580	27.9
Wee-mag		1499	34	61	61	63	3.28	643	30.02
Wee-mag		1499	35	60	60	63	5.00	706	32.02
Wee-mag		1499	36	60	60	63	5.00	769	33.91
Wee-mag		1499	37	60	60	62	3.33	795	34.66
Wee-mag		1499	38	60	60	62	3.33	857	36.38
Wee-mag		1499	39	60	60	62	3.33	919	38.01
Wee-mag		1499	40	60	60	62	3.33	981	39.56
Wee-mag		1499	41	59	59	62	5.08	1043	41.03
Wee-mag		1499	42	55	55	61	10.91	1063	41.49
Wee-mag		1499	43	50	50	58	16.00	995	39.9
Wee-mag		1499	45	38	38	48	26.32	661	30.6
Wee-mag		1499	46	34	34	48	41.18	709	32.11
Wee-mag		1499	47	33	33	45	36.36	616	29.13
Wee-mag		1499	49	32	32	41	28.13	510	25.39
Wee-mag		1499	50	32	32	40	25.00	501	25.05
Wee-mag		1499	52	31	31	34	9.68	269	15.21
Wee-mag		1499	54	31	31	33	6.45	283	15.88
Wee-mag		1499	56	30	30	32	6.67	293	16.35
							Average rel. deviation (%) =13.91		18.27

Appendix C: Benchmark Data with Hybrid GA   Table 6.4 (continued)

Author	No of operations	Cycle Time	Total Processing Time	H.GA No of Stations	Proposed Algorithm No of Station	Total Idle Time	% of Idle Time	Efficiency	rel. dev. from Hybrid GA[%]
Heskia	28	138	1024	8	9	218	17.55	82.45	12.50
		205	1024	5	6	206	16.75	83 25	20.00
		216	1024	5	5	56	5.19	94.81	0.00
		256	1024	4	5	256	20	80.00	25.00
		324	1024	4	4	272	20.99	79.01	0 00
		342	1024	3	4	344	25.15	74.85	33.33
Sawyer	30	25	324	14	17	101	23 76	76.24	21.43
		27	324	13	16	108	25	75.00	23 08
		30	324	12	14	96	22.86	77.14	16 67
		36	324	10	11	72	18.18	81.82	10 00
		41	324	8	9	45	12 2	87.80	12.50
		54	324	7	7	54	14.29	85.71	0 00
		75	324	5	5	51	13.6	86.40	0 00
Kilbridge and Wester	45	57	552	10	11	75	11 96	88.04	10.00
		79	552	7	8	80	12.6	87.34	14 29
		92	552	6	7	92	14 29	85.71	16 67
		110	552	6	6	108	16.36	83.64	0.00
		138	552	4	5	138	20	80.00	25.00
		184	552	3	4	184	25	75.00	33.33
Tonge	70	176	3510	21	23	538	13 29	86.71	9 52
		364	3510	10	11	494	12.34	87 66	10 00
		410	3510	9	9	180	4.88	95.12	0.00
		468	3510	8	8	234	6.25	93.75	0.00
		527	3510	7	7	706	16.75	95.15	0.00
Arcus	83	5048	75707	16	17	10109	11.78	88.22	6.25
		5853	75707	14	15	12088	13.77	86 23	7.14
		6842	75707	12	12	6397	7.79	92.21	0.00

Table 6.4 (continued)

Author	No of operations	Cycle Time	Total Processing Time	H.GA No of Stations	Proposed Algorithm No of Station	Total Idle Time	% of Idle Time	Efficiency	rel. dev. From Hybrid GA[%]
		7571	75707	11	12	15145	16.67	83.33	9.09
		8414	75707	10	10	8433	10 02	89.98	0 00
		8998	75707	9	9	5275	6.51	93.49	0.00
		10816	75707	8	8	10821	12.51	87.49	0.00
Arcus	111	5755	150399	27	33	39516	20.81	79.19	21.43
		8847	150399	18	20	26541	15	85.00	11.11
		10027	150399	16	17	20060	11.77	88 23	6.25
		10743	150399	15	16	21489	12.5	87.50	6 67
		11378	150399	14	15	20271	11.88	88 12	7.14
		17067	150399	9	10	20271	11.88	88.12	11.11
							Average (%)	**81.20**	11.56

APPENDIX D. COMPUTER CODE (C LANGUAGE)

```
#include<time.h>
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
#include<algorithm>
using namespace std;

#define MAX 500

#define For(i,a,b)      for(i=(a);i<(b);i++)

typedef __int64 LL;

int fac[] = {1,1,2,6,24,120,720};
int IterationNumber;

int MaximumPoolSize;
double CrossOverRate;

int n;
int pre[MAX][MAX];          //pre[i][j] = 1 if (i) precedes (j)

struct Fuzzy{
        double d,e,f;          //d < e < f
}CycleTime, Zero, BestTotalIdleTime, PercentIdleTime, Efficiency, TotalJobTime;

int BestStationNumber , nJobStation[MAX];

int npop;
struct Chromosome{
        int     index[MAX];
        int     StationNumber;
        Fuzzy  FFV;
}population[3*1000] , crossed[2];


struct Job{
        Fuzzy  time;
        double SD;
}J[MAX];

/*
```

```
        Function Prototypes
*/

int sort_descend(const void *,const void *);
int sort_ascend(const void *,const void *);

Fuzzy sqr(Fuzzy);
Fuzzy sqrt(Fuzzy);
Fuzzy operator/(Fuzzy ,Fuzzy );
Fuzzy operator/(Fuzzy ,double);
Fuzzy operator*(Fuzzy ,Fuzzy );
Fuzzy operator*(Fuzzy ,double);
Fuzzy operator*(double,Fuzzy );
Fuzzy operator+(Fuzzy ,Fuzzy );
Fuzzy operator-(Fuzzy ,Fuzzy );
bool  operator<(Fuzzy ,Fuzzy );
bool  operator>(Fuzzy ,Fuzzy );

void PrintFuzzy(Fuzzy);
int  ScanFuzzy(Fuzzy *);

int  ScanAndInit();
void PrintBox(char *);

void PS(Chromosome &);
bool Valid(Chromosome &);
bool operator==(Chromosome,Chromosome);



//getting the Station number and Fitness Function Value of a Chromosome
void Process(Chromosome &P){
        int i , ns;
        Fuzzy RemainTime;
        Fuzzy TotalIdleTime , IdleTime[MAX];
        Fuzzy sum,sum2;

        RemainTime = CycleTime;
        TotalIdleTime = Zero;

        ns = 0;
        For(i,0,n){

                if( J[ P.index[i] ].time < RemainTime )
                        RemainTime = RemainTime - J[ P.index[i] ].time;
```

```
		else{
			IdleTime[ns] = RemainTime;

			TotalIdleTime = TotalIdleTime + IdleTime[ns];
			RemainTime = CycleTime - J[ P.index[i] ].time;
			ns++;
		}

	}
	IdleTime[ns] = RemainTime;
	TotalIdleTime = TotalIdleTime + IdleTime[ns];
	ns++;

	sum = sum2 = Zero;
	For(i,0, ns){
		sum  = sum  + IdleTime[i];
		sum2 = sum2 + sqr(IdleTime[i]);
	}

	P.StationNumber = ns;
	P.FFV = sum / ns + sqrt(sum2);
}

void CrossOver(){

	int i,k,pos;
	int gene[4];

	while(1){
		k = rand() % n;
		if( crossed[0].index[k] != crossed[1].index[k] )
			break;
	}

	gene[0] = crossed[0].index[k];
	gene[1] = crossed[1].index[k];

	for(i=0;i<n;i++)
		if( crossed[0].index[i] == gene[1] )
			break;
	crossed[0].index[k] = gene[1];
	pos = i;
	for(i=pos;i<n-1;i++)
		crossed[0].index[i] = crossed[0].index[i+1];
	crossed[0].index[n-1] = gene[0];
```

```
	for(i=0;i<n;i++)
		if( crossed[1].index[i] == gene[0] )
			break;
	crossed[1].index[k] = gene[0];
	pos = i;
	for(i=pos;i<n-1;i++)
		crossed[1].index[i] = crossed[1].index[i+1];
	crossed[1].index[n-1] = gene[1];


	Process( crossed[0] );
	Process( crossed[1] );

}

void Scramble(Chromosome C , int k){
	int n1 , max_chrom , count;
	int p;

	n1 = n-k;
	if( n1 < 7 ){
		max_chrom = fac[n1];
		if(MaximumPoolSize < max_chrom)
			max_chrom = MaximumPoolSize;
	}
	else
		max_chrom = MaximumPoolSize;

	count = 0;
	p = 0;
	while(count < max_chrom && ++p < 1000){
		if( !next_permutation(C.index + k, C.index + n) )
			break;

		if( Valid(C) ){
			Process( C );
			population[npop++] = C;
		}
		count++;
	}
}

void Mutation(){
	int k;
	if( Valid( crossed[0] ) )	population[npop++] = crossed[0];
```

```
        if( Valid( crossed[1] ) )         population[npop++] = crossed[1];

        k = rand() % n;
        Scramble( crossed[0] , k );
        Scramble( crossed[1] , k );
}

Fuzzy GetTotalIdleTime(Chromosome P){

        int i , ns;
        Fuzzy RemainTime;
        Fuzzy TotalIdleTime, IdleTime[MAX];

        RemainTime = CycleTime;
        TotalIdleTime = Zero;

        ns = 0;
        For(i,0,n){

                if( J[ P.index[i] ].time < RemainTime )
                        RemainTime = RemainTime - J[ P.index[i] ].time;

                else{
                        IdleTime[ns] = RemainTime;

                        TotalIdleTime = TotalIdleTime + IdleTime[ns];
                        RemainTime = CycleTime - J[ P.index[i] ].time;
                        nJobStation[ns] = i;

                        ns++;
                }

        }
        IdleTime[ns] = RemainTime;
        TotalIdleTime = TotalIdleTime + IdleTime[ns];
        nJobStation[ns] = n;
        ns++;

        return TotalIdleTime;
}

void init(){
        Zero.d = 0;
        Zero.c = 0;
        Zero.f = 0;
        srand( time(NULL) );
```

```
}

int main(){
	int i;
	int step , k;
	Fuzzy Min,Max,Avg , Total;

	init();

	while( ScanAndInit() == 1 ){

		//determining the MAX SD based set
		For(i,0,n)
			population[0].index[i] = i;
		qsort( population[0].index , n, sizeof(int) , sort_descend);
		Process( population[0] );

		//determining the MIN SD based set
		For(i,0,n)
			population[1].index[i] = i;
		qsort( population[1].index , n, sizeof(int) , sort_ascend);
		Process( population[1] );

		For(step , 0 , IterationNumber){

			npop = 2;
			crossed[0] = population[0];
			crossed[1] = population[1];

			For(i,0,n * CrossOverRate)
				CrossOver();

			Mutation();
			sort(population , population + npop);

			Min = population[0].FFV;
			Max = population[npop-1].FFV;
			Avg = Zero;
			For(i,0,npop)
				Avg = Avg + population[i].FFV;
			Avg = Avg / ((double)(npop));

			printf("Iteration #%d\n", step+1);
			printf("Minimum FFV = ");  PrintFuzzy(Min);     printf("\n");
			printf("Average FFV = ");  PrintFuzzy(Avg);     printf("\n");
			printf("Maximum FFV = ");  PrintFuzzy(Max);     printf("\n\n");
```

```
		For(i,1,npop)
			if( !(population[0] == population[i]) )
				break;

		population[1] = population[i];

	}

	//best is population[0];
	BestStationNumber = population[0].StationNumber;
	BestTotalIdleTime = GetTotalIdleTime( population[0] );
	PercentIdleTime = 100 * BestTotalIdleTime /
							(BestStationNumber *
CycleTime);

	PrintBox("The Best Solution");
	printf("\n");

	i = 0;
	for(k=0;k < BestStationNumber;k++){

		Total = Zero;
		printf("Station %d:",k+1);
		for( ; i < nJobStation[k];i++){
			printf(" %d",population[0].index[i] + 1);
			Total = Total + J[ population[0].index[i] ].time;
		}
		printf("\n");

		printf("Processing Time:. ");
		PrintFuzzy(Total);
		printf("\n");

		printf("Fuzzy Idle Time:: ");
		Total = CycleTime - Total;
		PrintFuzzy(Total);
		printf("\n\n\n");

	}
	printf("\n");

	printf("Total Idle Time = ");
	PrintFuzzy( BestTotalIdleTime );
	printf("\n");
```

```
		printf("Percentage of Idle Time = ");
		PrintFuzzy( PercentIdleTime );
		printf("\n");

		Efficiency = 100. * TotalJobTime / (BestStationNumber * CycleTime);
		printf("Efficiency(%%) = ");
		PrintFuzzy( Efficiency );
		printf("\n");

		printf("\n\n\n\n\n\n\n");
	}
	return 0;
}

/*
	Other Functions
*/

void PrintBox(char *Text) {
	int i;
	int L = strlen(Text);

	//1st line
	printf("%c",218);
	for(i=0;i<L;i++)
		printf("%c",196);
	printf("%c\n",191);

	//2nd line
	printf("%c",179);
	printf("%s",Text);
	printf("%c\n",179);

	//3rd line
	printf("%c",192);
	for(i=0;i<L;i++)
		printf("%c",196);
	printf("%c\n",217);
}

int ScanAndInit(){
	int i,j,k;
	int before,after;

	//scanning number of jobs
	PrintBox("Input number of Jobs (0 to break)");
```

```
scanf("%d",&n);
if(!n)   return 0;

printf("\n");

//initialization of precedence table - Adjacency list
For(i,0,n)
        For(j,0,n)
                pre[i][j] = 0;

//scanning TASK info
PrintBox("Input Fuzzy-Time for each Job");
TotalJobTime = Zero;

For(i,0,n){

        ScanFuzzy( &J[i].time );
        J[i].SD = (J[i].time.d + 2*J[i].time.e + J[i].time.f) / 4.;
        TotalJobTime = TotalJobTime + J[i].time;

}
printf("\n");

//scanning CYCLE TIME
PrintBox("Input Fuzzy-Cycle-Time");
ScanFuzzy( &CycleTime );
printf("\n");

//scanning Precedence relationship
PrintBox("Input Precedence Relationship (0 0 to end)");

while(1){
        //"before" should precede "after"
        scanf("%d%d",&before,&after);

        if(!before || !after)
                break;

        before--;
        after--;

        pre[ before ][ after ] = 1;
}
printf("\n");

//all pairs       algorithm
```

```
	For(k,0,n)
		For(i,0,n)
			For(j,0,n)
				if(pre[i][k] && pre[k][j])
					pre[i][j] = 1;

	PrintBox("Input Number of Iteration");
	scanf("%d",&IterationNumber);

	PrintBox("Input Maximum Number of Offsprings in pool");
	scanf("%d",&MaximumPoolSize);

	PrintBox("Input Rate of Crossover");
	scanf("%lf",&CrossOverRate);

	return 1;
}

/*
	Sorting Job Initially
*/

int sort_descend(const void *p,const void *q){

	int a = *((int *)p);
	int b = *((int *)q);

	if( pre[a][b] )			return -1;
	if( pre[b][a] )			return 1;

	if( J[a].SD > J[b].SD )return -1;
	return 1;

}

int sort_ascend(const void *p,const void *q){

	int a = *((int *)p);
	int b = *((int *)q);

	if( pre[a][b] )			return -1;
	if( pre[b][a] )			return 1;

	if( J[a].SD < J[b].SD )return -1;
	return 1;
```

```
}

/*
        Chromosome Functions
*/
bool operator<(Chromosome A,Chromosome B){                              //return (A < B)

        if(A.StationNumber < B.StationNumber)
                return 1;

        double sA,sB;

        sA = A.FFV.d + 2*A.FFV.e + A.FFV.f;
        sB = B.FFV.d + 2*B.FFV.e + B.FFV.f;

        if(sA < sB)
                return 1;
        return 0;
}

//bool Equal(Chromosome &C1,Chromosome &C2){
bool operator==(Chromosome C1,Chromosome C2){
        int i;
        For(i,0,n)
                if(C1.index[i] != C2.index[i])
                        return 0;
        return 1;
}

bool Valid(Chromosome &C){
        int i;
        for(i=0;i<n-1;i++)
                if( pre[ C.index[i+1] ][ C.index[i] ] == 1)
                        return 0;
        return 1;
}

/*
        Fuzzy Structure Operator & Function Definitions
*/

int ScanFuzzy(Fuzzy *A){
        if( scanf("%lf%lf%lf" , &A->d , &A->e , &A->f) == 3)
                return 1;
        return 0;
}
```

```
void PrintFuzzy(Fuzzy A){
        printf("[%.2f, %.2lf, %.2lf]" , A.d , A.e , A.f),
}

Fuzzy operator/(Fuzzy A,Fuzzy B){            //return C = A / B
        Fuzzy C;
        double v[4];
        v[0] = A.d / B.d;
        v[1] = A.d / B.f;
        v[2] = A.f / B.d;
        v[3] = A.f / B.f;

        sort(v,v+4);

        C.d = v[0];
        C.e = A.e / B.e;
        C.f = v[3];

        return C;
}

Fuzzy operator/(Fuzzy A,double x){           //return C = A / x
        Fuzzy C;     C.d = A.d / x;          C.e = A.e / x;        C.f = A.f / x;
        return C;
}

Fuzzy operator*(Fuzzy A,Fuzzy B){            //return C = A * B
        Fuzzy C;
        double v[4];
        v[0] = A.d * B.d;
        v[1] = A.d * B.f;
        v[2] = A.f * B.d;
        v[3] = A.f * B.f;

        sort(v,v+4);

        C.d = v[0];
        C.e = A.e * B.e;
        C.f = v[3];

        return C;
}

Fuzzy operator*(Fuzzy A,double x){           //return C = A * x
        Fuzzy C;     C.d = A.d * x;          C.e = A.e * x;        C.f = A.f * x;
```

```
}

Fuzzy operator*(double x,Fuzzy A){          //return C = x * A
        Fuzzy C;        C.d = A.d * x;          C.e = A.e * x;          C.f = A.f * x;
        return C;
}

Fuzzy operator+(Fuzzy A,Fuzzy B){           //return C = A + B
        Fuzzy C;        C.d = A.d + B.d;        C.e = A.e + B.e;        C.f = A.f + B.f;
        return C;
}

Fuzzy operator-(Fuzzy A,Fuzzy B){           //return C = A + B
        Fuzzy C;        C.d = A.d - B.f;        C.e = A.e - B.e;        C.f = A.f - B.d;
        return C;
}

Fuzzy sqr(Fuzzy A){                                         //return square of A
        Fuzzy C;        C.d = A.d*A.d;                  C.e = A.e*A.e;                  C.f =
A.f*A.f;
        return C;
}

Fuzzy sqrt(Fuzzy A){                                    //return square root of A
        Fuzzy C;        C.d = sqrt(A.d);        C.e = sqrt(A.e);        C.f = sqrt(A.f);
        return C;
}

bool operator<(Fuzzy A,Fuzzy B){            //return (A < B)
        if(A.d < B.f && A.e < B.e && A.f < B.d)
                return 1;
        return 0;
}

bool operator>(Fuzzy A,Fuzzy B){            //return (A > B)
        if(A.d > B.f && A.e > B.e && A.f > B.d)
                return 1;
        return 0;
}
```