DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SOLVING MIXED-MODEL ASSEMBLY LINE SEQUENCING PROBLEM USING ADAPTIVE GENETIC ALGORITHMS

by

Onur Serkan AKGÜNDÜZ

September, 2008

İZMİR

SOLVING MIXED-MODEL ASSEMBLY LINE SEQUENCING PROBLEM USING ADAPTIVE GENETIC ALGORITHMS

A Thesis Submitted to the

Graduate School of Natural and Applied Sciences of Dokuz Eylül University

In Partial Fulfillment of the Requirements for the Degree of Master of Science in

Industrial Engineering, Industrial Engineering Program

by

Onur Serkan AKGÜNDÜZ

September, 2008

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"SOLVING MIXED-MODEL ASSEMBLY LINE SEQUENCING PROBLEM USING ADAPTIVE GENETIC ALGORITHMS"** completed by **ONUR SERKAN AKGÜNDÜZ** under supervision of **PROF. DR. SEMRA TUNALI** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

....

Prof. Dr. Semra TUNALI

Supervisor

....

Jury Member	Jury Member

....

Prof.Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

First and foremost I offer my sincerest gratitude to my supervisor and mentor, Prof. Dr. Semra Tunalı, who has supported me throughout my thesis with her wisdom and patience. Her inspiration, guidance and counsel were invaluable. I'm especially grateful to her for allowing me the room to work in my own way while helping me to maintain focus.

I would also like to thank the members of jury, Prof. Dr. Bahar Karaoğlan and Asst. Prof. Dr. Arslan M. Örnek, for accepting to serve on my dissertation jury in the midst of all their activities.

I would like to express my deep appreciation to my family who always give me encouragement and support at each stage of my studies. Their undying patience has given me the peace of mind needed to dedicate my efforts towards this thesis.

Finally, I would like to thank everybody who was important to the successful realization of thesis, as well as expressing my apology that I could not mention personally one by one.

Onur Serkan AKGÜNDÜZ

SOLVING MIXED-MODEL ASSEMBLY LINE SEQUENCING PROBLEM USING ADAPTIVE GENETIC ALGORITHMS

ABSTRACT

The focus of this M.Sc study is to introduce adaptive Genetic Algorithm (GA) based approaches for single- and multi-objective mixed-model assembly line sequencing problems (MMALSP), which deal with the determination of production launching orders so that the variations in part consumption rates (VPC) are minimized. In addition to this objective, minimization of total utility work (UW) and cost for sequence-dependent setups (SC) are also considered in multi-objective version of the MMALSP.

In order to solve single-objective MMALSPs, an adaptive GA based approach which incorporates adaptive parameter control techniques into a pure GA is proposed. The proposed approach, integrates an adaptive elitist strategy and a scheme for varying probability of mutation according to the feedback taken from the algorithm. Using this approach, the MMALSP is solved under the objective of minimizing VPC in a four level assembly environment, i.e. product, subassembly, component and raw material.

Later, by modifying the adaptive parameter control techniques and integrating them into a Pareto Stratum – Niche Cubicle GA, a multi-objective MMALSP with three objective functions (i.e., minimization of VPC, UW and SC) is solved. Finally, to evaluate the performance of the proposed approach, various sets of experiments have been carried out.

Keywords: Mixed-model assembly line, model sequencing, adaptive genetic algorithm, multi-objective optimization, adaptive parameter control.

KARIŞIK-MODELLİ MONTAJ HATTI SIRALAMA PROBLEMLERİNDE ADAPTİF GENETİK ALGORİTMALARIN KULLANIMI

ÖZ

Bu yüksek lisans çalışmasının esas amacı, tek- ve çok-amaçlı karışık-modelli montaj hattı sıralama problemleri (KMMHSP) için, parça kullanım oranlarındaki değişkenlikleri (PKOD) en küçükleyecek şekilde, model üretim sıralarının belirlenmesi sağlayan adaptif Genetik Algoritma (GA) yaklaşımlarını ortaya koymaktır. Çok-amaçlı problem, PKOD'a ek olarak, yardımcı işçi kullanımının (YİK) ve hazırlık sürelerinin (HS) en küçüklenmesi amaçlarını da dikkate almaktadır.

Çalışmada ilk olarak, tek-amaçlı KMMHSPlerini çözmek üzere, adaptif parametre kontrolü tekniklerini öz GA'ya uygulayan adaptif GA tabanlı bir yaklaşım önerilmiştir. Bu yaklaşım, adaptif bir elit stratejisi ve algoritmadan aldığı geribildirime göre mutasyon olasılığını düzenleyen bir yapı içermektedir. Önerilen yaklaşım kullanılarak KMMHSP problemi, son ürün, alt-montaj, bileşen ve ham madde olmak üzere dört seviye içeren bir montaj ortamında, PKOD'u en küçükleyecek şekilde çözülmüştür.

Çalışmanın devamında, daha önceden tek-amaçlı problemin çözümünde kullanılan adaptif parametre kontrol teknikleri modifiye edilerek Pareto Stratum – Niche Cubicle olarak bilinen GA'ya entegre edilmiş ve PKOD, YİK ve HS'nin en küçüklenmesini amaçlayan çok-amaçlı bir KMMHSP problemi çözülmüştür. Son olarak da, önerilen çözüm yöntemlerinin performanslarını değerlendirmek üzere çeşitli boyutlardaki problem setleri üzerinde deneyler yapılmıştır.

Anahtar Kelimeler: Karışık-modelli montaj hattı, model sıralama, adaptif genetik algoritma, çok-amaçlı optimizasyon, adaptif parametre kontrolü.

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

In today's many industries, varying customer demands and intense competition require a highly diversified product portfolio provided in a cost effective manner. In order to provide increased flexibility for product diversification, many manufacturers have upgraded their assembly lines, which were originally developed for a cost efficient mass production of a single standardized product. The current trend is to design mixed-model assembly lines (MMAL), which are capable of producing a variety of different product models simultaneously and continuously.

In a MMAL, the application of flexible workers and machinery leads to a substantial reduction in setup times and cost, so that different products with lot size of one can be jointly manufactured at the same line in intermixed sequences. In addition to the flexible resources being available, the production processes of manufactured goods require a minimum level of homogeneity (Boysen, Fliedner, & Scholl, 2007a). This is achieved by using a generic product model which is customizable by optional features.

The design of an MMAL involves several issues. The most important ones are resource planning, generic product modeling, line balancing and model sequencing. Resource planning is concerned with the selection of production means adequate for performing the assembly operations specified by the assembly planning. Balancing an assembly line means distributing work required to assemble a product among a set of work stations. In addition to the long and mid-term line balancing problems, there are short-term model sequencing problems, which aim at determining the production sequence of different models to be produced during the work shift. In this thesis study, we focus on the mixed-model sequencing problems.

Determining the sequence of launching models to the assembly line is of particular importance for efficient use of MMALs. In the literature, several objectives and methods have been proposed to judge the efficiency of different production sequences including minimizing total utility work, minimizing variation of production rates, keeping a constant rate of part usage, minimizing total setup cost, minimizing the risk of stopping a conveyor, minimizing the overall line length, leveling workloads and so on. Which objectives to employ depends on the goals of the research and/or company.

Sequencing with a single objective can be meaningful when the objective unconditionally rules over all the others. In practice, however, several objectives, often conflicting with each other, need to be simultaneously considered (Hyun, Kim, & Kim, 1998). Such conflicts make it complicated to plan and control the production activities as the sequencing decision becomes a multi-objective problem.

An important issue that complicates the sequencing problem is its combinatorial nature. Typically, an enormous number of possible production sequences exist, even for relatively small problems, so that finding the optimal solution is usually impractical. In the literature, various solution approaches are proposed including dynamic programming (Yano & Rachamadugu, 1991), linear and integer programming (Drexl & Kimms, 2001; Ventura & Radhakrishnan, 2002), goal chasing methods (Celano, Costa, Fichera, & Perrone, 2004; Mane, Nahavandi, & Zhang, 2002; Monden, 1993), branch and bound (Drexl, Kimms, & Matthießen, 2006), tabu search (McMullen, 1998; Scholl, Klein, & Domschke, 1998), simulated annealing (Cho, Paik, Yoon, & Kim, 2005; Kara, Özcan, & Peker, 2007a, 2007b; McMullen & Frazier, 2000), ant colony optimization technique (Boysen, Fliedner, & Scholl, 2007b; Gagné, Gravel, & Price, 2006), evolutionary and genetic algorithms (Hyun et al., 1998; Kim, Kim, & Kim, 2000, 2006; Mansouri, 2005; McMullen, Tarasewich, & Frazier, 2000; Ponnambalam, Aravindan, & Rao, 2003; Yu, Yin, & Chen, 2006) and several other heuristics. Among these, genetic algorithms have been shown to be quite successful in dealing with many manufacturing optimization problems. A genetic algorithm (GA) is a highly simplified computational model of biological evolution. In this study, we aim at solving mixed-model sequencing problems using GAs.

The values of GA parameters greatly determine the performance of GAs. Choosing the right parameter values, however, is a time-consuming task. Furthermore, this task may need to be repeated for different instances of the problem. A recent trend in GA based research studies is to employ adaptive or self-adaptive parameter control mechanisms to remedy this situation (Bingul, Sekmen, & Zein-Sabatto, 2000; Chang, Hsieh, & Wang, 2007; Eiben, Hinterding, & Michalewicz, 1999; Herrera & Lozano, 2003; Huang, Chang, & Sandnes, 2006; Liu, Zhou, & Lai, 2003; Shi, Eberhart, & Chen, 1999; Smith & Fogarty, 1997; Srinivas & Patnaik, 1994; Zhao, Zhao, & Jiao, 2005). During the survey of current literature, we have not noted any study employing adaptive or self-adaptive parameter control mechanisms to solve MMAL sequencing problem (MMALSP). Based on this observation, in this study, an adaptive genetic algorithm based approach is developed in order to solve the MMALSP.

This study is organized as follows. In Chapter 2, detailed background information about the genetic algorithms, multi-objective optimization and mixed-model sequencing problem are given. In order to highlight the place of this study in the current literature, we extensively surveyed the relevant studies. Chapter 3 presents both the evaluation criteria we employed for classifying the current relevant literature and also the findings of this survey study. In Section 4, we present the details of the proposed adaptive GA based approach to solve the single-objective MMALSP and compare its performance with the pure GA. In Chapter 5, an adaptive GA based approach is developed to solve the multi-objective MMALSP and various sets of experiments are carried out to evaluate its performance. Finally, concluding remarks and the future research directions are given in Chapter 6.

CHAPTER TWO
BACKGROUND INFORMATION

This section presents detailed background information about the mixed-model assembly line sequencing problem and genetic algorithms. First, the problem is explained in detail by presenting the problem statement, mathematical models, a summary of common problem parameters, line characteristics, and common objective functions employed to study this problem. Following, genetic algorithms including the related terminology, GA components and multi-objective GA approaches are presented.

2.1 Mixed-model Assembly Line Sequencing

The sequencing problem appears when variations of the same basic product are produced on the same production line. These variations imply that the processing times on the individual stations differ, dependent on the model to be processed. This type of problem is called the mixed model assembly line sequencing problem (MMALSP) and is defined by various parameters which reflect the characteristics of the stations and the production line. This chapter first presents a short history of the sequencing problem, and later gives the problem statement and a summary of common problem parameters, line characteristics, and common objective functions employed to study this problem.

The mixed-model sequencing problem was first investigated by Wester & Kilbridge (1964), and since then, a large number of researches employing a variety of approaches (i.e., exact methodologies, meta-heuristics) have been carried out. During the literature survey, we noted that while some researchers dealt with only mixed-model sequencing problem, others focused on both line balancing and sequencing problems.

One of the early studies dealing with both problems in a hierarchical framework has been carried out by Thomopoulos (1967). In this study, line balancing procedure was an adaptation of single-model line balancing techniques to mixed-model schedules. This procedure was followed by a sequencing procedure for determining the order in which models are launched to the line. The proposed approach resulted in an increase in the efficiency of the assembly line by providing near optimum solutions.

Another research that dealt with the aggregated problem of balancing and sequencing has been carried out by Merengo, Nava, & Pozetti (1999). As it was the case in Thomopoulos (1967), the authors proposed a hierarchical approach, in which they first dealt with the balancing problem. The balancing objectives were minimization of number of stations and number of incomplete units on the line, whereas sequencing algorithm aimed at keeping a constant rate of parts usage. Four balancing approaches were proposed and compared to each other. Also the results of the sequencing algorithm were compared to those of Miltenburg (1989) and shown to be better.

The evolution concept has been also introduced to the aggregated problem by several researchers (e.g. Kim et al., 2000, 2006; Miltenburg, 2002; Rekiek, De Lit, & Delchambre, 2000). In these studies, the authors have developed various evolutionary and genetic algorithm approaches to deal with the aggregated problem in both straight and U-shaped lines.

Another group of studies only focused on mixed-model sequencing problems. Dar-el & Cother (1975) proposed a new formulation for the sequencing problem under the objective function of minimizing the overall length of assembly line. The authors evaluated the effects of five factors (i.e., the number of models, the model cycle time deviation, the operator time deviation, the production demand deviation for each model, and the number of stations in the assembly line) on the overall assembly line length and suggested that the first three factors had major effects on the line length. Another finding was that for maximum efficiency in utilization of space the open-station interfaces should be preferred.

Monden (1993) defined two goals for evaluating the performance of sequencing approaches. The first one was based on leveling the load at each workstation in order to minimize the risk of stopping the conveyor, and the other one was maintaining a constant feeding rate of every model. The author stated that the latter one was more applicable to JIT production systems.

Yano & Bolat (1989) provided a literature review on sequencing in mixed-model assembly lines. They also developed a heuristic which aims to minimize total utility work and compared their results with those of two automobile manufacturers' existing algorithms.

Yano & Rachamadagu (1991) investigated the problem of sequencing jobs, each representing a combination of product options, on a paced assembly line. They developed an optimal procedure for the situation where a single station is affected by an option. They also provided a heuristic procedure for multiple stations. The procedure was compared with an existing procedure used in industry.

Miltenburg (1989) studied the sequencing problem with the objective of keeping a constant rate of part usage. It was assumed that each product requires approximately the same number and mix of parts. Hence a constant rate of usage of all parts used by the line was achieved by considering only the demand rates for the products, and ignoring the resulting part demand rates. Three algorithms were presented for an exact solution but since the algorithms' worst case complexity was exponential, two heuristics were also proposed.

Bard, Dar-El, & Shtub (1992) proposed an analytical framework for sequencing MMALs and presented the characteristics of sequencing problems including open and closed stations, launching discipline, sequencing objectives, line movement, and operator schedules. Six variants of the sequencing problem were formulated and solved under the objective of assembly line length minimization.

In recent years, the increased popularity of genetic algorithms has led researchers to propose several GA based approaches for the sequencing problem. The first research on the application of GAs to the sequencing problem in MMALs has been carried out by Kim, Hyun, & Kim (1996). This was followed by Hyun et al. (1998) who also considered the multi-objective nature of the sequencing problem. McMullen et al. (2000) combined multiple objectives into a single objective using the weighted-sum approach and solved this problem with GAs.

There are several other published researches dedicated to the MMALS problem, covering a broad range of solution methods. Boysen et al. (2007a) present a detailed classification scheme for MMALS problem which they then use to classify most of the published researches in this area. The reader may refer to this study for an extensive list of model sequencing related literature.

2.1.1 Problem Statement and Mathematical Models

Most of the existing literature mentions the optimization of line balancing and model sequencing in a consecutive order (however, they usually focus on one of the two). Once the line is balanced and the design of the line is obtained, it is necessary to achieve a reasonable, if not optimum, order for the jobs to be processed consecutively (Färber & Coves, 2005).

Sequencing problems in MMALs can be considered from different points of view. In this section, we present mathematical models of three different versions of sequencing problem which aim at minimizing the total utility work (work overload), the variation of part consumption rates and the total setup cost, respectively. All models rely on the following assumptions made by Hyun et al. (1998):

- Assembly line is a conveyor system moving at a constant speed (v_c).
- Line is partitioned into J stations.
- All stations are closed ones so that workers cannot cross stations’ boundaries.

- Minimum part set (MPS) production is used. MPS is a vector representing a product mix, such that $(d_1,...,d_m)=(D_1/h,...,D_M/h)$, where M is the total number of models, D_m is the number of products of model type m which needs to be assembled during an entire planning horizon and h is the greatest common divisor or highest common factor of $D_1, D_2,..., D_M$. This strategy operates in a cyclical manner. The number of products produced in one cycle is $I=\sum_{m=1}^{M} d_m$. Obviously h times repetition of the MPS can meet the total demand in the planning horizon.
- Products are launched onto the conveyor at a fixed rate. The launch interval (γ) is set to $T/(I \times J)$, where T is the total operation time required to produce one cycle of MPS products ($T=\sum_{j=1}^{J}\sum_{m=1}^{M} t_{jm} d_m$, where t_{jm} is the operation time for model m at station j).
- Processing times are deterministic.
- Workers' moving time is ignored.

2.1.1.1 Minimizing the Total Utility Work

During the line balancing phase, in order to avoid excessive station capacities, the cycle time is determined as an average for all models. As a consequence, the processing times of some models are higher than the cycle time, whereas that of others are lower. If several models with higher processing times follow each other at the same station, the worker will not be able to return to the left-hand border before the next work piece has arrived and thus be consecutively moved towards the right-hand border of the station. This finally results in a work overload whenever the operations of a work piece can not be finished within the station's boundaries. Depending on the exact type of boundaries considered this might necessitate one of the following reactions (Boysen et al., 2007a) :

i. the whole assembly line is stopped until all stations have finished work on their current work piece,

ii. utility workers support the operator(s) of the station to finish work just before the station's border is reached,

iii. the unfinished tasks and all successors are left out and executed off-line in special finishing stations after the work piece has left the last station of the line, or

iv. the production speed is accelerated at the risk of quality defects.

To avoid such costly compensations, mixed-model sequencing searches for sequences where those models with high processing times alternate with less work-intensive ones at each station (Wester & Kilbridge, 1964). For this purpose, models are scheduled at each station and cycle, by explicitly taking into account processing times, worker movements, station borders and further operational characteristics of the line.

Let L_j be the fixed line length of station j and U_{ij} be the amount of the utility work required for the ith product in a sequence at station j. The following model is presented by Hyun et al. (1998):

Minimize $$\sum_{j=1}^{J}\left(\sum_{i=1}^{I} U_{ij} + Z_{(i+1)j} / v_c\right) \tag{1}$$

S.T.

$$\sum_{m=1}^{M} x_{im} = 1 \quad \forall i \tag{1.1}$$

$$\sum_{i=1}^{I} x_{im} = d_m \quad \forall m \tag{1.2}$$

$$Z_{(i+1)j} = \max\left[0, \min\left(Z_{ij} + v_c \sum_{m=1}^{M} x_{im} t_{jm} - (\gamma \times v_c), L_j - (\gamma \times v_c)\right)\right] \quad \forall i, j \tag{1.3}$$

$$U_{ij} = \max\left[0, \left(Z_{ij} + v_c \sum_{m=1}^{M} x_{im} t_{jm} - L_j\right) \Big/ v_c\right] \quad \forall i, j \tag{1.4}$$

$$x_{im} = 0 \text{ or } 1 \quad \forall i, m \tag{1.5}$$

$$Z_{1j} = 0,\ Z_{ij} \geq 0 \quad \forall i, j \tag{1.6}$$

$$U_{ij} \geq 0 \quad \forall i, j \tag{1.7}$$

where Z_{ij} is the starting position of the work on the ith product in a sequence at station j, and x_{im} is 1 if the ith product in a sequence is model m; otherwise x_{im} is 0. The second term in the objective function takes into the account for the utility work that may be required at the end of a cycle. Eq. (1.1) ensures that exactly one product is assigned to each position in a sequence. Eq. (1.2) guarantees that demand for each model is satisfied. Eq. (1.3) indicates the starting position of the worker at each station j on product $i+1$ in a sequence. Utility work U_{ij} for the ith product in a sequence at station j is determined by Eq. (1.4).

2.1.1.2 Minimizing the Variation of Part Consumption Rates

Keeping the part consumptions at a constant rate is considered to be an important goal for JIT production systems. These systems rely on continual and stable part supply. Therefore, it is important to keep part demand rates as constant as possible over time. This can be achieved by minimizing the variation of actual part consumption rates from the expected ones.

g	level number (level 4: raw material, level 3: components, level 2: subassemblies, level 1: final assembly)
n_g	number of outputs at level g, where $g = 1,2,3,4$
d_{i1}	demand for product $i = 1,2,...,n_1$
t_{igl}	number of units of output i at level g used to produce one unit of product l, $i = 1,2,...,n_g$; $g = 2,3,4$; $l = 1,2,...,n_1$
$d_{ig} = \sum_{h=1}^{n_1} t_{igh} d_{h1}$	demand for output i at level g
$DT_g = \sum_{i=1}^{n_g} d_{ig}$	total demand for production at level g, where $g = 1,2,3,4$

$r_{ig} = d_{ig} / DT_g$ ratio of level g production devoted to output i, $i = 1,2,...,n_g$; $g = 1,2,3,4$.

DT_1 products must be assembled on the final assembly line during the planning horizon. Let us say that there are DT_1 consecutive stages and a product is assigned to each of these stages. This is called a schedule. The schedule is represented in the model by the following variables:

x_{i1k}	number of units of product i produced during stages $1,2,...,k$. The notation k is used throughout to denote the stage number.
$x_{igk} = \sum_{h=1}^{n_1} (t_{igh} \times x_{h1k})$	number of units of output i at level g produced during stages $1,2,...,k$.
$XT_{gk} = \sum_{i=1}^{n_g} x_{igk}$	total production at level g during stages $1,2,...,k$.
w_g	weights, $g = 1,2,3,4$.

If production were strictly synchronized with demand we would find that after k stages the total output x_{igk} of part i at level g would be $\left(XT_{gk} \times r_{ig}\right)$. However, equality is not always possible. So we strive to schedule the system as to make x_{igk} close to $\left(XT_{gk} \times r_{ig}\right)$ for each i; g and k. Equipped with these definitions and notations, Miltenburg & Sinnamon (1989) formulated the scheduling problem as follows:

Select x_{igk}; $i = 1,2,...,n_1$; $k = 1,2,...,DT_1$ to minimize the variation in parts consumption, i.e.

$$\text{Minimize} \quad \sum_{k=1}^{DT_1} \sum_{g=1}^{4} \sum_{i=1}^{n_g} w_g [x_{igk} - (XT_{gk} \times r_{ig})]^2 \qquad (2)$$

S.T.

$$XT_{1k} = k,\ k = 1, 2, \ldots DT_1 \tag{2.1}$$

$$0 \le x_{i1k} - x_{i1(k-1)} \text{ and } x_{i1k} \text{ is an integer, } i = 1,2,\ldots n_1; k = 1,2,\ldots DT \tag{2.2}$$

$$x_{i1DT_1} = d_{i1}, i = 1,2,\ldots.n_1 \tag{2.3}$$

Eq. (2.1) ensures that exactly k products are scheduled during k stages. Eq. (2.2) ensures that it is not possible to schedule less than zero units, more than one unit, or a fraction of a unit of any product. Eq. (2.3) ensures that exactly the right number of each type of model is produced during the planning horizon (Ponnambalam et al., 2003).

2.1.1.3 Minimizing the Total Setup Cost

In many industries, sequence-dependent setups are considered as an important issue in assembly operations. A setup is required each time two consecutive items in the production sequence are different. Therefore, this objective aims to minimize product changes in the production schedule by batching products as much as possible. A mathematical model considering sequence-dependent setups has been developed by Hyun et al. (1998) as follows:

Minimize $$\sum_{j=1}^{J}\sum_{i=1}^{I}\sum_{m=1}^{M}\sum_{r=1}^{M} x_{imr} c_{jmr} \tag{3}$$

S.T.

$$\sum_{m=1}^{M}\sum_{r=1}^{M} x_{imr} = 1 \quad \forall i, \tag{3.1}$$

$$\sum_{m=1}^{M} x_{imr} = \sum_{p=1}^{M} x_{(i+1)rp} \quad i = 1,\ldots, I-1, \forall r, \tag{3.2}$$

$$\sum_{m=1}^{M} x_{Imr} = \sum_{p=1}^{M} x_{1rp} \quad \forall r, \tag{3.3}$$

$$\sum_{i=1}^{I}\sum_{r=1}^{M} x_{imr} = d_m \quad \forall m, \tag{3.4}$$

$$x_{imr} = 0 \text{ or } 1 \quad \forall i, m, r, \tag{3.5}$$

where c_{jmr} is the setup cost required when the model type is changed from m to r at station j and x_{imr} is 1 if model types m and r are assigned, respectively at the ith and $(i+1)$th position in a sequence; otherwise x_{imr} is 0. Eq. (3.1) is a set of position constraints indicating that every position in a sequence is occupied by exactly one product. Eqs. (3.2) and (3.3) ensure that the sequence of products must be maintained while repeating the cyclic production. Eq. (3.4) imposes the restriction that all the demands must be satisfied in terms of MPS.

2.1.2 Problem Complexity

The total number of sequences for a mixed-model assembly sequencing problem can be computed as follows:

$$\text{Total sequences} = \frac{(\sum_{m=1}^{M} d_m)!}{\prod_{m=1}^{M} (d_m!)}. \tag{4}$$

Here M is the number of different models, m is the model type and d_m is the demand of model m. As the problem increases in size, the number of feasible solutions increases in the exponential way. Thus, problems with large number possible solutions cannot be usually solved optimality within a reasonable amount of time (Tavakkoli-Moghaddam & Rahimi-Vahed, 2006). Therefore, only a few exact procedures are proposed in the literature up to now. They either solve very restrictive problem versions or are intended to serve as reference procedures for evaluating heuristic methods (Scholl, 1999).

Also, when the multi-objective nature of the problem is considered, finding production sequences with desirable levels of all objectives is NP-hard (Hyun et al., 1998). Conventional multi-objective optimization techniques including linear programming, gradient methods, methods of inequalities, goal attainment or weighted sum approach, all have some shortcomings as pointed out by Deb (1999):

> "In dealing with multi-criterion optimization problems, classical search and optimization methods are not efficient, simply because (i) most of them cannot find multiple solutions in a single run, thereby requiring them to be applied as many times as the number of desired Pareto-optimal solutions, (ii) multiple application of these methods do not guarantee finding widely different Pareto-optimal solutions, and (iii) most of them cannot efficiently handle problems with discrete variables and problems having multiple optimal solutions."

Emulating the biological evolution mechanism and Darwin's principal on "survival-of-the-fittest", genetic algorithms have been recognized to be well suited for multi-objective optimization problems where conventional tools fail to work well (Tan, Lee, & Khor, 2002).

2.1.3 Characteristics of MMALSP

Characteristics of this problem can be grouped into four categories: (i) station characteristics, (ii) line characteristics, (iii) operational characteristics, and (iv) objective functions. The following subsections briefly summarize these characteristics.

2.1.3.1 Station Characteristics

The station environment hosts various decision factors for the MMALS problem.

2.1.3.1.1 Station Boundaries. Depending on the nature of the tasks and the physical layout of the facility, stations in the assembly line may be *closed* or *open* at the either side of a station boundary. In a *closed* station, the operator is not allowed to move out of his work area when he assembles the products. On the other hand, in an *open* station, the operator is permitted to move outside his station up to some specific limits (such as the reach of the material handling equipment or range of power tools). In no instance are the operators permitted to interfere with each other, or to service a unit simultaneously (Bard et al., 1992; Sarker & Pan, 2001).

2.1.3.1.2 Reaction on Work Overload. Whenever the operator of a station is not able to complete the assigned tasks before the work piece leaves the station (due to a restricted station length or due to the transport system), *work overload* occurs. Work overload may be compensated by the temporary employment of utility workers, stopping the line or another sanction. No matter which sanction is selected, work overload is inefficient and expensive and should be minimized (Becker & Scholl, 2006). Boysen et al. (2007a) list the reaction alternatives on imminent work overload as follows:

- In the general case, it is assumed that work overloads do not affect starting times of successive stations, so that in spite of an overload at station k, the successive station $k+1$ can start processing the work piece as soon as the work piece reaches its left station border (provided that it finished work on the preceding work piece). This is always the case if (i) work overload is compensated by the timely assignment of a utility worker, who helps to finish the work within the station's boundaries, (ii) the processing velocity is increased so that the work is finished in time or (iii) the unfinished tasks are left out and completed later at end-of-line repair shops or special in-line repair stations. Irrespective of the measure taken, the line can continue processing work pieces.
- The work piece is taken off the transportation system, e.g., for disposal or off-line completion, so that successive stations have empty cycles.
- The line is stopped as soon as an unfinished work piece reaches a station's definite border, which induces idle time at all other stations, and is, for instance, the typical compensation in the Japanese automobile industry
- Overloads are (seen to be) compensated by variable station borders. Typically, the decision on the stations' length is part of assembly line balancing and, thus, already fixed for short-term model sequencing. Nevertheless, such sequencing models can be utilized to either decide on the station extents on the basis of a representative medium-term model mix as an addition to the balancing information or as a surrogate model, e.g., for

assembly lines with open stations. A further differentiation of variable borders can be made as follows:

- An *early start model* presupposes fixed left borders of stations, so that each station may only expand in downstream direction. Each worker starts processing in the first cycle at the left border of his station (reference point 0). When the operator has finished his work he walks back and starts work, when another work piece has already been launched into the station area. Otherwise, he goes back to the beginning of the station (reference point 0) and waits for the next work piece.
- In a *late start model* the stations expand in both directions. Workers who completed their tasks move back until they reach the subsequent work piece, even if it puts them beyond their respective reference point 0, instead of incurring idle time by waiting at left station borders.

2.1.3.1.3 Processing Times. A further important characteristic defining different versions of MMALSP is the variability of task times. Whenever the expected variance of task times is sufficiently small, as in case of, e.g., simple tasks or highly reliable automated stations, the task times are considered to be *deterministic*. Considerable variations, which are mainly due to the instability of humans with respect to work rate, skill and motivation as well as the failure sensitivity of complex processes, require considering *stochastic* task times. Besides stochastic time variations, systematic reductions are possible due to learning effects or successive improvements of the production process (Becker & Scholl, 2006).

2.1.3.1.4 Concurrent Work. Concurrent work enables the worker(s) of a station to start processing although the previous station has not finished its work on the respective work piece. This necessitates open stations as well as work pieces of an appropriate size, so that workers do not impede each other (Boysen et al., 2007a).

2.1.3.1.5 Setups. A mixed-model assembly line necessitates a considerable reduction of setup times and costs. Otherwise, a production of different models in an intermixed sequence is utterly impossible. Nevertheless, short setup operations, which consume just a fraction of the cycle time, may be relevant, e.g., due to tool swaps (Boysen et al., 2007a). Setup time/cost is concerned if an additional time/cost appears to change the setup of a station in order to be able to process the next job. If the setup time/cost is independent of the model, it can be simply added to the processing time/cost (Färber & Coves, 2005).

2.1.3.1.6 Parallel Stations. Tasks with comparatively large processing times may lead to inefficient line balances, as they force the cycle time to be inefficiently large inducing idle times at other stations. Thus, it might be preferable to install parallel stations, which alternately process identical work contents. Parallel stations can be implemented in two different ways: In *spacial parallelization*, the respective stations are located side by side and are alternately fed with work pieces over a switch. In *chronological parallelization*, a number, say p, of operators or teams of operators work at the same segment of the serial line covering p sequential stations. The teams process the work pieces for p cycles such that they circulate within the line segment each team being responsible for one out of p successive work piece (Boysen et al., 2007a).

2.1.3.2 Assembly Line Characteristics

2.1.3.2.1 Number and Homogeneity of Stations. Assembly lines in real world are composed of more than one station. However, a restriction to a given number n of stations or even a single station might be of value. When there is more than one station, another characteristic emerge from this situation: homogeneity of stations. In practical cases, an assembly line may consist of stations with diverging characteristics. For instance, open and closed stations can be mixed throughout the line, which is referred to as hybrid lines. The majority of existing sequencing approaches presuppose stations with homogeneous characteristics, which is very limiting for practical applications (Boysen et al., 2007a).

2.1.3.2.2 Line Layout. Traditionally, an assembly line is organized as a *serial line*, where single stations are arranged along a (straight) conveyor belt (see Figure 2.1 a). Such serial lines are rather inflexible and have other disadvantages which might be overcome by a *U-shaped* assembly line (see Figure 2.1 b). Both ends of the line are closely together forming a rather narrow ‘‘U’’. Stations may work at two segments of the line facing each other simultaneously (crossover stations). Besides improvements with respect to job enrichment and enlargement strategies, a U-shaped line design might result in a better balance of station loads due to the larger number of task-station combinations (Becker & Scholl, 2006).

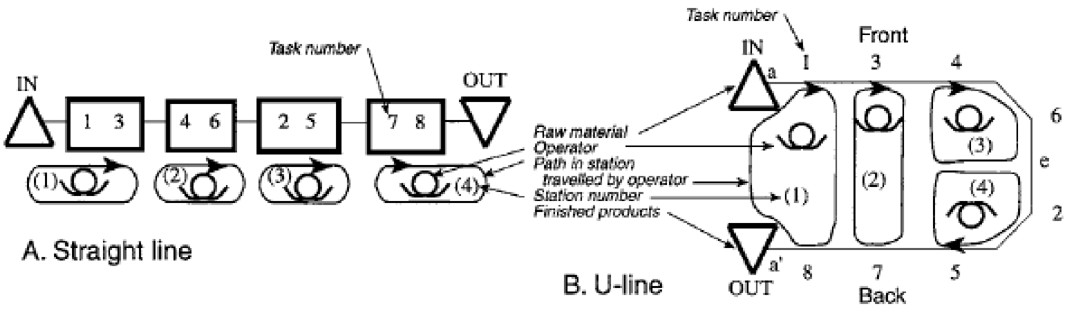


Figure 2.1 Straight and U-shaped lines (Miltenburg, 2002)

In principle, the physical layout of the line is not relevant for the sequencing decision. However, a U-line allows operators to work on more than one work piece per cycle at different positions on the line, because crossover stations have access to two legs of the U-shaped line simultaneously. This influences the sequencing problem considerably (Boysen et al., 2007a).

2.1.3.2.3 Launching Discipline. There are two main launching strategies: (i) *fixed rate launching*, (ii) *variable rate launching*. Fixed rate launching (FRL) implies that each unit, regardless of model type, is equi-spaced on the line. Here, the reciprocal of the distance between successive units equals to the cycle time. When product variations are small, often common choice is FRL due to its simplicity. With increasing product diversification, though, variable rate launching (VRL), made possible by the arrival of sophisticated controllers and sensors, is now becoming a practical alternative. VRL augments the flexibility of line operation as the launching interval can be dynamically adapted to prevent idle times and work overloads (Bard et al., 1992; Boysen et al., 2007a).

2.1.3.2.4 Worker Return Velocity. Whenever a worker finishes all operations, he needs to return upstream to start processing the next work piece. In the real world, walking a distance takes some time. Nevertheless, in a mathematical model two kinds of premises with regard to the return speed are possible (Boysen et al., 2007a):

- If workers are considerably faster than the movement of the line, return times of workers can either be neglected or treated as fixed and directly added to station times. Because the assumption of infinite return speed of workers simplifies analysis and, in many cases, is a slight relaxation of reality only, most approaches act on this assumption.
- If return times vary considerably from cycle to cycle and worker to worker, e.g., due to different processing times and, hence, different walking distances, an approach for mixed-model sequencing should take finite return speeds into account.

2.1.3.3 Operational Characteristics

Most production systems consist of multiple levels. Nevertheless, under the assumption of a final assembly which instantaneously pulls its material through the whole supply chain, it may be sufficient to solely consider the final production stage. A level production schedule at the final assembly automatically induces a level production schedule at all preceding stages, if a small lot production is directly triggered by the respective material usages. However, multiple production levels can also be considered explicitly. In this case, the deviations of all production levels are included within the objective function (Boysen et al., 2007a).

2.1.3.4 Sequencing Objectives

It has been noted during the literature survey that the researchers employed various types of objective functions for sequencing in mixed model assembly lines. The most common ones are:

- *Keeping a constant rate of part usage.* Some researchers who are particularly interested in just-in-time production systems have addressed the problem of keeping a constant rate of part usage. One basic requirement of JIT systems is continual and stable part supply. Since this requirement can be realized when the demand rate of parts is constant over time, the objective of keeping a constant rate of part usage is important to a successful operation of the systems. (Hyun et al., 1998). This objective can be achieved by matching demand with actual production.

- *Minimizing variation of production rates.* Smoothing variation of production rates is considered as a substitute for the ultimate objective of keeping a constant rate of part usage under the assumption that products require approximately equal number and mix of parts (Mansouri, 2005).

- *Minimizing work overload.* This objective minimizes the time (or space) by which station borders would be exceeded if no type of compensation (e.g., assigning utility workers, stopping the line, taking work piece off the line, etc.) was carried out. As work overload is usually avoided by the assignment of utility workers, this objective is also referred to as "minimization of total utility work" (Boysen et al., 2007a). Minimizing the utility work contributes to reducing not only labor cost but also the risk of stopping the conveyor and the required line length (Kim et al., 2000).

- *Minimizing the set-up cost/time.* A setup is required each time two consecutive items in the production sequence are different. Many assembly operations often require sequence-dependent setups. For instance, an automotive body station needs a setup when the door types are changed. (Hyun et al., 1998).

- *Minimizing line length.* This objective can be followed with variable station borders. It is a surrogate for minimizing investment cost of the transportation system, which is considered to raise in proportion to an increase in length (Boysen et al., 2007a).

- *Minimizing throughput time.* Throughput time is defined as time interval between the launching of the first work piece and the finishing of the last. This is highly correlated to the objective "minimize line length". This objective also depends on non-fixed station lengths (Boysen et al., 2007a).

- *Leveling workloads for stations on the line.* This goal is about sequencing mixed models to achieve balanced workloads over time in each assembly station (Ding, Zhu, & Sun, 2006). This objective has several benefits: establishing a sense of equity among workers, reducing line congestion and thus increasing the throughput rate (Kim et al., 2006).

- *Minimizing the duration of line stoppages.* When the line is stopped no work pieces can be completed. Thus, this objective minimizes opportunity costs for lost sales (Boysen et al., 2007a). This objective was emerged from the autonomation concept in the Toyota production system (Xiabo, 1999).

- *Minimizing total idle time.* Idle time represents unused (non-productive) capacities of machines and workers. Idle time occurs whenever a worker waits for a work piece to reach his station. Idle time in itself has its justification if unused capacities can be profitably utilized for performing other work (Boysen et al., 2007a).

2.1.3.5 Hybrid Sequencing Problems

Design of the MMALs involves several issues other than the model sequencing problems (e.g. line balancing). Although those issues have been separately considered in the literature, several hybrid approaches are also developed to deal with integrated problems of sequencing and balancing.

The short-term decision problem of model sequencing is heavily interdependent with the long- to mid-term assembly line balancing. The line balance decides on the assignment of tasks to stations and thus determines the work content and material usage per station and model. This decision constitutes the input data of model sequencing. Thus, the amount of overload resulting from a planned model sequence by itself is a measure of efficiency for the achieved line balance. That is why some authors have proposed a simultaneous consideration of both planning problems (Boysen et al., 2007a).

2.2 Genetic Algorithms

It is known that the MMAL sequencing problem falls into NP-hard class of combinatorial optimization problems and thus a large-sized problem may be computationally intractable. The computational time can be a critical factor in choosing the right method in solving this problem since real time alteration of model sequences is often necessary when demand pattern changes or part shortages occur. For this reason, in recent years meta-heuristics have been widely adopted by a number of researchers (Hyun et al., 1998). One of the most popular metaheuristics dealing with many manufacturing optimization problems is Genetic Algorithms.

A genetic algorithm (GA) is a search and optimization method, which simulates the natural behaviour of biological systems. After being introduced by John Holland (Holland, 1975), they have become increasingly popular among other heuristic methods and they have been successfully adapted to solve several combinatorial optimization problems in the literature. This popularity and success depend on their several advantages on other methods. First, unlike some other heuristic methods, which iterate on a single solution, GAs work with a population of candidate solutions. This helps GAs to be useful in problem domains that have a complex fitness landscape as recombination (crossover) is designed to move the population away from local optima that a traditional hill climbing algorithm might get stuck in. Second, the inductive nature of the GA means that it doesn't have to know any rules of the problem — it works by its own internal rules. This is very useful for complex or loosely defined problems. Other advantages include but are not limited to scalability to higher dimensional problems and ease of adjustability to the problem at hand. Haupt & Haupt (2004) lists the advantages of GAs as follows:

- Optimizes continuous or discrete variables,
- Doesn’t require derivative information,
- Simultaneously searches from a wide sampling of the cost surface,
- Deals with a large number of variables,
- Is well suited for parallel computers,
- Optimizes variables with extremely complex cost surfaces (they can jump out of a local minimum),
- Provides a list of optimum variables, not just a single solution,
- May encode the variables so that the optimization is done with the encoded variables, and
- Works with numerically generated data, experimental data, or analytical functions.

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).

In nature, the `fittest’ members of a population typically survive at higher rates compared to the `weakest’ members. These fittest members then reproduce with one another, resulting in a new generation of the population having attributes similar to that of their parents. In each new generation, mutations can also occur on an infrequent basis. In this situation, offspring develop attributes of their own, independent of their parents. Mutations can add diversity and a certain amount of robustness to the population (McMullen, 2000). These natural phenomena can be exploited to find good solutions to combinatorial optimization problems.

Genetic algorithms can be defined as a computer simulation in which a population of abstract representations (called chromosomes) of candidate solutions (called individuals) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings (real-valued, tree, etc.) are also possible. The evolution usually starts from an initial population of randomly generated individuals and proceeds throughout the 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 (crossed and possibly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. The goal here is eventually to find generations of solutions to a combinatorial optimization problem where the objective function value approaches the global optimum.

2.2.1 Biological and GA Terminology

There are several terms in the context of genetic algorithms that are used in analogy to the real terms of biology. The definitions of these terms (Mitchell, 1999) are given below.

Chromosome. All living organisms consist of cells, and each cell contains the same set of one or more *chromosomes*—strings of DNA—that serve as a blueprint for the organism.

Gene. A chromosome can be conceptually divided into *genes*— each of which encodes a particular protein. Very roughly, one can think of a gene as encoding a *trait*, such as eye color.

Allele. The different possible "settings" for a trait (e.g., blue, brown, hazel) are called *alleles*.

Locus. Each gene is located at a particular *locus* (position) on the chromosome.

Genome. Many organisms have multiple chromosomes in each cell. The complete collection of genetic material (all chromosomes taken together) is called the organism's *genome*.

Genotype. The term *genotype* refers to the particular set of genes contained in a genome. Two individuals that have identical genomes are said to have the same genotype.

Phenotype. The genotype gives rise, under fetal and later development, to the organism's *phenotype* — its physical and mental characteristics, such as eye color, height, brain size, and intelligence.

Recombination (Crossover). During sexual reproduction, *recombination* (or *crossover*) occurs: in each parent, genes are exchanged between each pair of chromosomes to form offspring.

Mutation. Offspring are subject to *mutation*, in which single nucleotides (elementary bits of DNA) are changed from parent to offspring, the changes often resulting from copying errors.

Fitness. The *fitness* of an organism is typically defined as the probability that the organism will live to reproduce (*viability*) or as a function of the number of offspring the organism has (*fertility*).

In genetic algorithms, these terms have much simpler meanings. The term *chromosome* typically refers to a candidate solution to a problem, often encoded as a bit string. The "genes" are either single bits or short blocks of adjacent bits that encode a particular element of the candidate. An allele in a bit string is either 0 or 1; for larger alphabets more alleles are possible at each locus. Crossover typically consists of exchanging genetic material between parents. Mutation consists of flipping the bit at a randomly chosen locus (or, for larger alphabets, replacing the symbol at a randomly chosen locus with a randomly chosen new symbol).

2.2.2 Components of GA

Design of a GA requires careful inspection and determination of the suitable choices for several factors including data structures, initial population formation, fitness function, selection and insertion strategies, genetic operators, genetic parameters, and termination criteria. Once they are decided, the genetic search process works as given in Figure 2.2.

Using the process flow presented in Figure 2.2, pseudocode representation for a traditional GA can be written as follows:

```
Input Parameters (MaxGeneration, PopSize, $P_C$:crossover rate, $P_M$: mutation rate)
Generate Initial Solutions
Evaluate Fitness Values, Zs
For $i$ = 1 to MaxGeneration
    For $j$ = 1 to PopSize / 2
        Choose Two Solutions (with a fitness bias)
        If Random Number < $P_C$ then Perform Crossovers
        If Random Number < $P_M$ then Perform Mutations
    Next $j$
    Determine Zs for New Solutions
    Note Best of Generation
```

```
        If $Z$(Best of Generation) is better than $Z$(Best Found thus Far) Then
            Replace Best Found thus Far with Best of Generation
        Endif
    Next $i$
    Present Best Found Thus Far
```

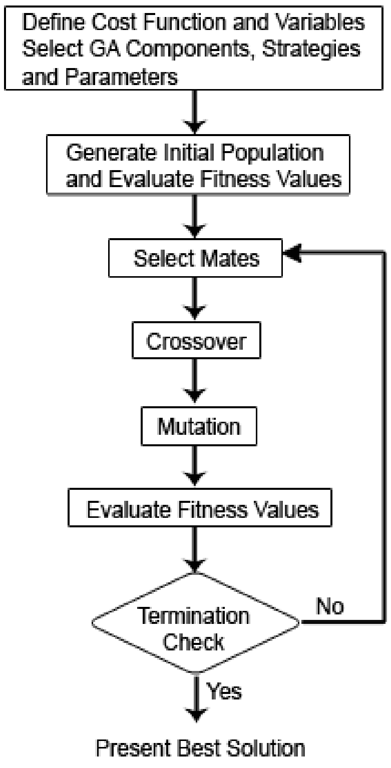

Define Cost Function and Variables
Select GA Components, Strategies and Parameters
Generate Initial Population and Evaluate Fitness Values
Select Mates
Crossover
Mutation
Evaluate Fitness Values
Termination Check
No
Yes
Present Best Solution

Figure 2.2 Flow-chart of a simple GA

The simple search procedure given in Figure 2.2 and the above pseudocode are the basis for most applications of GAs. Although the outline of the algorithm is common in most of the applications, the details of the components are determined according to the problem at hand. In the following sections, components of GA will be described in detail.

2.2.2.1. Genetic Representation

Genetic representation is a way of representing solutions (individuals) in GAs. It's accomplished by translating the solutions into chromosome representations. The chromosome format used in this translation process is called encoding.

Designing a good genetic representation that is expressive and evolvable is essential to the success of the algorithm. Most GA applications use fixed-length, fixed-order bit strings to encode candidate solutions. Variable length representations may also be used, but one must consider the trade-offs between possible gains and implementation complexity. Common genetic representations are (i) *binary* encoding, (ii) *permutation* encoding, (iii) *real-valued* encoding and (iv) *tree* encoding.

2.2.2.1.1 Binary Encoding. Binary encoding is the most common and simplest type of genetic representation. In binary encoding every chromosome is a string of bits, 0 or 1. For example:

Chromosome A → 1 0 1 0 1 1 1 1
Chromosome B → 0 0 1 1 0 1 1 0

Popularity of binary encoding can be attributed to a number of reasons. Firstly, it is the simplest representation type, and hence initial GA researchers concentrated on such encodings. Also much of the existing GA theory is based on the assumption of fixed-length, fixed-order binary encodings. In addition, heuristics about appropriate parameter settings (e.g., for crossover and mutation rates) have generally been developed in the context of binary encodings (Mitchell, 1999).

However, it should be noted that this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation operations.

2.2.2.1.2 Permutation Encoding. Permutation encoding can be used in ordering problems, such as traveling salesman problem or task ordering problem. In permutation encoding, every chromosome is a string of numbers that represent a position in a sequence. For example:

Chromosome A $\rightarrow$ 2 7 4 5 6 3 8 1
Chromosome B $\rightarrow$ 1 4 7 8 5 2 6 3

Here, the two chromosomes, A and B, present two arbitrary orderings of numbers from 1 to 8. For this specific example, there are $8!$ alternative sequences.

Although being useful for ordering problems, for certain types of crossover and mutation operations, permutation encoding may require that some corrections be made to leave the chromosome consistent (i.e. have real sequence in it).

2.2.2.1.3 Real-Valued Encoding. For many applications, it is most natural to use an alphabet of many characters or real numbers to form chromosomes. Use of binary encoding for this type of problems would be difficult. In the real-valued encoding, every chromosome is a sequence of some values. Values can be anything connected to the problem, such as real numbers, characters or any object. For example:

Chromosome A $\rightarrow$ 2.27 5.32 7.14 4.45 5.81 6.77
Chromosome B $\rightarrow$ D B A A C B D B B B A C

Mitchell (1999) indicates that several empirical comparisons between binary encodings and multiple-character or real-valued encodings have shown better performance for the latter. But the performance of the chosen encoding depends very much on the problem and the details of the GA being used.

Real-valued encoding may be a good choice for certain problems, however, for this type of encoding it is often necessary to develop some new crossover and mutation operators which are specific to the problem at hand.

2.2.2.1.4 Tree Encoding. In the tree encoding every chromosome is a tree of some objects, such as functions or commands in a programming language. For example:

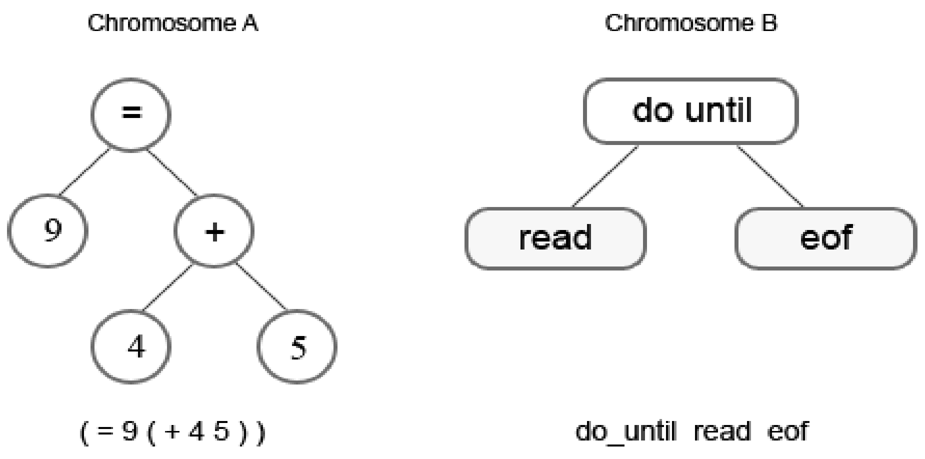


Figure 2.3 Tree encoding examples

Tree encoding is useful for evolving programs or any other structures that can be encoded in trees. Programming language LISP is often used for this purpose, since programs in LISP are represented directly in the form of tree and can be easily parsed as a tree, the crossover and mutation can be done relatively easier (Obitko, 1998).

2.2.2.2 Fitness Function

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. Here the GA searches for a set of parameter values (a chromosome) that maximize or minimize the given fitness function.

The fitness function is always problem dependent. For instance, in the traveling salesman problem, the total distance traveled is minimized, whereas, in the knapsack problem, the total value of objects that can be put in a knapsack of some fixed capacity is maximized. Therefore, the fitness functions for these two problems are the sum of total distance traveled and sum of total value of objects in the knapsack, respectively.

In the above-mentioned two problems, fitness functions can be defined mathematically, however, in some problems, it is hard or even impossible to define the fitness expression. When the form of fitness function is not known (for example, visual appeal or attractiveness) or the result of optimization is required to fit a particular user preference (for example, taste of coffee or color set of the user interface) human evaluation may be necessary. In these cases, interactive genetic algorithms which use human evaluation are usually preferred.

In following, various terms which are related to fitness function development in GAs are explained:

Fitness Landscape. In the context of population genetics, a *fitness landscape* is a representation of the space of all possible genotypes along with their fitnesses (Mithcell, 1999). A sample fitness landscape for the arbitrary function (5) is given in Figure 2.4.

$$Z = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{1+0.1(x^2+y^2)} \tag{5}$$

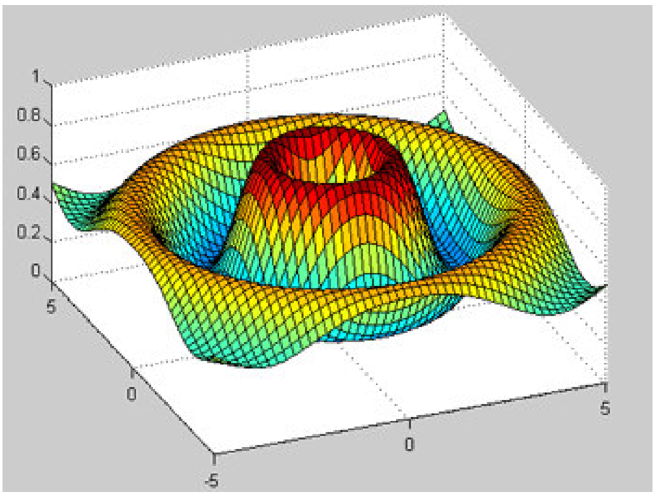


Figure 2.4 A sample fitness landscape

Such plots are called landscapes because the plot of fitness values can form "hills," "peaks," "valleys," and other features analogous to those of physical landscapes.

Evolution causes populations to move along landscapes in particular ways, and adaptation can be seen as the movement toward local peaks. Likewise, in GAs the operators of crossover and mutation can be seen as ways of moving a population around on the landscape defined by the fitness function.

Global Optimum. A global optimum is a point in the fitness landscape whose value exceeds that of any other value (or is exceeded by every other value if it is a minimization problem).

Local Optimum. A local optimum is a point in the fitness landscape that has the property that no chain of mutations starting at that point can go up without first going down (Ashlock, 2005).

Making an analogy to a mountain range, the global optimum can be thought of as the top of the highest mountain, while the local optima are the peaks of every mountain or foothill in the range. Even rocks will have associated local optima at their high points. Note that the global optimum is one of the local optima. Also, note that there may be more than one global optimum if two mountains tie for highest. GAs search such fitness landscapes for highly fit individuals.

Once an appropriate representation is chosen and the fitness function is defined, GA proceeds to initialize a population of solutions usually randomly.

2.2.2.3 Initial Population

Initially, a number of individual solutions are generated to form an initial population. The population size depends on the nature and complexity of the problem. Traditionally, the population is generated randomly, covering the entire range of possible solutions. Occasionally, the solutions may be seeded in areas where optimal solutions are likely to be found. Latter can be accomplished by several approaches such as a pre-run heuristic method, initial run of the GA, a case- or memory-based reasoning system.

After the initialization phase, the chromosomes are passed to the fitness function to determine the fitness values which are used to decide which chromosomes in the initial population are fit enough to survive and possibly reproduce offspring in the next generation.

2.2.2.4 Selection

Selection is the stage of a genetic algorithm in which the fitter individuals are chosen from a population for later breeding so that their offspring will in turn have even higher fitness.

The purpose of selection is to emphasize the fitter individuals in the population in hopes that their offspring will in turn have even higher fitness. Selection pressure is a critical parameter for GAs, and it has to be balanced with variation from crossover and mutation (the "exploitation/exploration balance"): too-strong selection means that a set of highly fit but suboptimal chromosomes will dominate the population, reducing the diversity needed for further change and progress (exploration problem), and the search will terminate prematurely; too-weak selection will result in too-slow evolution (exploitation problem).

As was the case for encodings, numerous selection schemes have been proposed in the GA literature (Mitchell, 1999). 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 to keep the diversity of the population large, preventing premature convergence on poor solutions. The following section explains the widely used selection schemes in the literature.

2.2.2.4.1 Roulette Wheel and Stochastic Universal Sampling. Holland (1975) used *fitness-proportionate* selection which chooses parents in direct proportion to their fitness. If individual *i* has fitness f_i, then its expected value (the expected number of times an individual will be selected to reproduce) is $f_i / \bar{f}$, where $\bar{f}$ is the average of the fitness values of the entire population. The most common method to implement fitness-proportionate selection is the *roulette wheel sampling*.

The analogy to a roulette wheel can be envisaged by imagining a roulette wheel in which each candidate solution represents a pocket on the wheel; the size of the pockets are proportionate to the probability of selection of the solution. Selecting *N* chromosomes from the population is equivalent to spinning the roulette wheel *N* times, as each candidate is drawn independently. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation.

Roulette wheel sampling method can be implemented as follows (Mitchell, 1999):

1. Sum the total expected value of individuals in the population. Call this sum *T*.
2. Repeat *N* times:
    a. Choose a random integer *r* between 0 and *T*.
    b. Loop through the individuals in the population, summing the expected values, until the sum is greater than or equal to *r*. The individual whose expected value puts the sum over this limit is the one selected.

It should be noted that while this method may lead elimination of candidate solutions even though they have a very high fitness value, on the other hand, it may allow the survival of some weak solutions. Allowing the survival of weak solutions can be considered as an advantage: as though a solution may be weak, it may include some component which could prove useful following the recombination process.

Although roulette wheel method statistically results in the expected number of offspring for each individual, with the relatively small populations, the actual number of offspring allocated to an individual might be far from its expected value (an extremely unlikely series of spins of the roulette wheel could even allocate all offspring to the worst individual in the population). Hence, a different sampling method — *stochastic universal sampling* (SUS) — is proposed to minimize this "spread" (the range of possible actual values, given an expected value). Rather than spin the roulette wheel *N* times to select *N* parents, SUS spins the wheel once — but with *N* equally spaced pointers, which are used to select the *N* parents (Mitchell, 1999).

However, SUS does not solve the major problems with fitness-proportionate selection. Typically, early in the search the fitness variance in the population is high and a small number of individuals are much fitter than the others. Under fitness-proportionate selection, these individuals and their descendents will multiply quickly in the population, in effect preventing the GA from doing any further exploration. This is known as "premature convergence." In other words, fitness-proportionate selection early on often puts too much emphasis on "exploitation" of highly fit strings at the expense of exploration of other regions of the search space. Later in the search, all individuals in the population become very similar (the fitness variance is low), the real fitness differences for selection diminish, and evolution grinds to a near halt, implying that the rate of evolution depends on the variance of fitnesses in the population (Mitchell, 1999).

2.2.2.4.2 Sigma Scaling. To address premature convergence problems caused by the variance of fitnesses, the researchers have experimented with several "scaling" methods—methods for mapping "raw" fitness values to expected values so as to make the GA less susceptible to premature convergence. One example is "sigma scaling", which keeps the selection pressure (i.e., the degree to which highly fit individuals are allowed many offspring) relatively constant over the course of the run rather than depending on the fitness variances in the population. Under sigma scaling, an individual's expected value is a function of its fitness, the population mean, and the population standard deviation. An example of sigma scaling would be as follows (Mitchell, 1999):

$$ExpVal(i,t) = \begin{cases} 1 + \dfrac{f(i) - \bar{f}(t)}{2\sigma(t)} & \text{if } \sigma(t) \neq 0 \\ 1.0 & \text{if } \sigma(t) = 0 \end{cases} \tag{6}$$

where *ExpVal*(*i,t*) is the expected value of individual *i* at time *t*, *f*(*i*) is the fitness of *i*, *f*(*t*) is the mean fitness of the population at time *t*, and σ (*t*) is the standard deviation of the population fitnesses at time *t*. This function, gives 1.5 expected offspring to an individual with fitness one standard deviation above the mean. If *ExpVal*(*i,t*) is less than 0, expected offspring may be reset to a small arbitrary number such as 0.1, so that individuals with very low fitness will have some small chance of reproducing.

Under sigma scaling, if a single chromosome dominates at the beginning of the run, the variability in fitnesses will also be large, and scaling by the variability will reduce the dominance. Later in the run, when populations are typically more homogeneous, scaling by this smaller variability will allow the highly fit chromosomes to reproduce.

2.2.2.4.3 Boltzman Selection. Sigma scaling keeps the selection pressure more constant over a run. But often different amounts of selection pressure are needed at different times in a run. Depending on how far along in the run the generation is, Boltzman selection scheme allows this kind of variation in the selection process by changing the selection pressure. Early on, it may be better to allow lower selection pressure, allowing the less fit chromosomes to reproduce at rates similar to the fitter chromosomes, thereby maintaining a wider exploration of the search space. Later in the run, increasing the selection pressure helps the GA to converge more quickly to the optimal solution, hopefully the global optimum.

Similar to simulated annealing approach, under Boltzmann selection, a continuously varying "temperature" controls the rate of selection according to a preset schedule. The temperature starts out high, which means that selection pressure is low (i.e., every individual has some reasonable probability of reproducing). The temperature is gradually lowered, which gradually increases the selection pressure, thereby allowing the GA to focus to the best part of the search space while maintaining the "appropriate" degree of diversity. A typical implementation given in Mitchell (1999) is to assign an expected value to each individual *i*,

$$ExpVal(i,t) = \frac{e^{f(i)/T}}{\left\langle e^{f(i)/T} \right\rangle_t} \tag{7}$$

where *T* is temperature and $\langle\ \rangle_t$ denotes the average over the population at time *t*. Experimenting with this formula will show that, as *T* decreases, the gap between the expected values of the best and worst individuals gets larger. The desire is to have this happen gradually over the course of the search, so temperature can gradually decrease according to a predefined schedule.

2.2.2.4.4 Rank Selection. In recent years, the researchers widely employed several alternative selection schemes, such as rank and tournament selection, which are proposed to overcome the problems present in fitness-proportionate selection approaches.

Rank selection, which prevents too-quick convergence, ranks the chromosomes according to their raw fitness values and then based on these rankings, assigns reproductive fitness values to them. This may be done linearly (linear ranking) or exponentially (exponential ranking). Ranking avoids the selection pressure exerted by the proportional fitness method, but it also ignores the absolute differences among the chromosome fitnesses. Moreover, ranking does not take variability into account and provides a moderate adjusted fitness measure, since the probability of selection between chromosomes ranked k and $k+1$ is the same regardless of the absolute differences in fitness.

The downside of rank selection, which is caused by slowing down the selection pressure, is that the GA will in some cases be slower in finding highly fit individuals. However, in many cases the increased preservation of diversity that results from ranking leads to more successful search than the quick convergence that can result from fitness-proportionate selection.

2.2.2.4.5 Tournament Selection. The selection schemes (explicit fitness mapping methods) described above require an intermediate step of computing a modified fitness and in the case of rank selection, a sorting process was required to put the entire population in order by rank. To avoid these time-consuming procedures, tournament selection can be used as an alternative approach. This method is similar to the rank selection in terms of the selection pressure.

There are several variants of tournament selection. In the simplest, *binary* tournament selection, pairs of individual are picked at random from the population. Whichever has the higher fitness is copied into a mating pool. This is repeated until the mating pool is full. Larger tournaments may also be used, where the best of n randomly chosen individuals is copied into the mating pool. Using larger tournaments has the effect of increasing the selection pressure, since below average individuals are less likely to win a tournament, while above average individuals are more likely to (Beasley, Bull, & Martin, 1993).

A further generalization is *probabilistic* binary tournament selection. In this, the better individual wins the tournament with probability p, where $0.5 < p < 1$. Using lower values of p has the effect of decreasing the selection pressure, since below average individuals are comparatively more likely to win a tournament. By adjusting tournament size or win probability, the selection pressure can be made arbitrarily large or small (Beasly et al., 1993).

2.2.2.4.6 Elitism. Elitism refers to the selection strategy that forces the GA to retain some number of the best individuals at each generation, protecting them against destruction through crossover, mutation, or inability to reproduce.

Those members of the population guaranteed to survive are called the *elite*. Elitism guarantees that a population with a fixed fitness function cannot slip back to a smaller maximum fitness in later generations, but it also causes the current elite to be more likely to have more children in the future causing their genes to dominate the population. Such domination can impair search of the space of genes, because the current elite may not contain the genes needed for the best possible creatures. A good compromise is to have a small number of elites (Ashlock, 2005).

Several researchers report that elitism significantly improves GA's performance.

2.2.2.4.7 Steady-State Selection and Generation Gaps. The generation gap is defined as the proportion of individuals in the population which are replaced in each generation. Most work has used a generation gap of 1, which means the whole population is replaced in each generation. In some schemes, such as the elitist schemes described above, successive generations overlap to some degree—some portion of the previous generation is retained in the new population. However, a more recent trend has favoured steady-state replacement. This operates at the other extreme: in each generation only a few (typically two) individuals are replaced.

Steady-state GAs may be a better model of what happens in nature but no evidence has been found that steady-state replacement is fundamentally better than the generational one. They are often used in evolving rule-based systems in which incremental learning (and remembering what has already been learned) is important and in which members of the population collectively (rather than individually) solve the problem at hand.

2.2.2.5 Genetic Operators

A genetic operator is a process used in genetic algorithms to maintain genetic diversity. Genetic variation is a necessity for the process of evolution. Genetic operators used in genetic algorithms are analogous to those which occur in the natural world: crossover and mutation. Here, the decision of which genetic operators to use depends greatly on the encoding strategy, fitness function and the details of the problem at hand.

2.2.2.5.1 Crossover. Crossover is a genetic operator which is used to vary the programming of a chromosome from one generation to the next. It is an analogy to reproduction and biological crossover, upon which genetic algorithms are based. It could be said that the main distinguishing feature of a GA is the use of crossover. In fact, the power of GAs arises from crossover.

Crossover causes a structured, yet randomized exchange of genetic material between solutions, with the possibility that good solutions can generate better ones (Srinivas & Patnaik, 1994). Ashlock (2005) gives the formal definition of crossover as follows:

A *crossover operator* for a set of genes G is a map

$$\text{Cross} : G \times G \rightarrow G \times G \tag{8.1}$$

or

$$\text{Cross} : G \times G \rightarrow G. \tag{8.2}$$

The points making up the pairs in the domain space of the crossover operator are termed *parents*, while the points either in or making up the pairs in the range space are termed *children (offspring)*. The children are expected to preserve some part of the parents' structure.

a. *Single-Point Crossover.* Many different approaches have been tried for crossover in GAs. The simplest form is the *single-point* crossover in which a single crossover position is chosen at random and the parts of two parents after the crossover position are exchanged to form two offspring. This means that the information for each child comes from a different parent before and after the crossover point. One of the shortcomings of this method is *positional bias*: the schemas that can be created or destroyed by a crossover depend strongly on the location of the bits in the chromosome. Single-point crossover treats some loci preferentially: the segments exchanged between the two parents always contain the endpoints of the strings, and loci near one another in the representation are kept together with a much higher probability than those that are farther apart.

b. *Two-Point Crossover.* To overcome the positional bias problems, *multiple-point* crossover methods have been developed. The most common approach is the *two-point* crossover where the segments between two randomly chosen loci are exchanged. While this method prevents the endpoint effect mentioned above, it is also less likely to disrupt schemas with large defining lengths and therefore it can combine more schemas than single-point crossover.

c. *Parameterized Uniform Crossover.* Some practitioners believe strongly in the superiority of *parameterized uniform* crossover, in which an exchange happens at each bit position with probability p (typically $0.5 < p < 0.8$). Parameterized uniform crossover has no positional bias—any schemas contained at different positions in the parents can potentially be recombined in the offspring. However, this lack of positional bias can prevent coadapted alleles from ever forming in the population, since parameterized uniform crossover can be highly disruptive of any schema (Mitchell, 1999). It must also be noted that this method is computationally expensive because of the large number of random numbers needed.

Most applications of GAs use static crossover operators, that is to say that their mechanisms, parameters, and probability of application are fixed at the beginning and constant throughout the run of the algorithm. However, there is no single choice of operators which is optimal for all problems. In fact, the optimal choice of operators for a given problem will be time-variant i.e. it will depend on such factors as the degree of convergence of the population. Based on theoretical and practical approaches, a number of authors have proposed methods of adaptively controlling the operators, usually invoking some kind of *meta-learning* algorithm, in order to try and improve the performance of the Genetic Algorithm as a function optimizer (Smith & Fogarty, 1997).

d. *Adaptive Crossover*. A technique to use in these situations is *adaptive crossover*. In adaptive crossover, each creature has its gene augmented by a *crossover template*, a string of 0's and 1's with one position for each item in the original data structure. When two parents are chosen, the crossover template from the first parent chosen is used to do the crossover. In positions where the template has a 0, items go from first parent to the first child and the second parent to the second child. In positions where the template has a 1, items go from the first parent to the second child and from the second parent to the first child. The parental crossover templates are themselves crossed over and mutated with their own distinct crossover and mutation operators to obtain the children's crossover templates. The templates thus coevolve with the creatures and seek out crossover operators that are currently useful. This can allow evolution to focus crossover activity in regions where it can help the most. The crossover templates that evolve during a successful run of an evolutionary algorithm may contain nontrivial useful information about the structure of the problem (Ashlock, 2005).

Adaptive crossover can suffer from a common problem called a *two-timescale problem*. The amount of time needed to efficiently find those fit genes that are easy to locate with a given crossover template can be a great deal less than that needed to *find* the crossover template in the first place. For some problems this will not be the case, for some it will, and intuition backed by preliminary data is the best tool currently known for telling which problems might benefit from adaptive crossover. If a problem must be solved over and over for different parameters, then saving crossover templates between runs of the evolutionary algorithm may help. In this case, the crossover templates are being used to find good representations, relative to the crossover operator, for the problem in general while solving specific cases (Ashlock, 2005).

2.2.2.5.2 Crossover For Permutation Problems. As it is the case in the mixed-model sequencing problems, sometimes an optimization process requires sorting the elements of a list in the correct order. For such problems, above-mentioned crossover operators may be inappropriate as they may produce invalid offspring. Since the chromosome is an ordered list (or there may be repeated elements of which quantities must not change), a direct swap would introduce duplicates and remove necessary candidates from the list. Consider the following two parent chromosomes for a mixed-model sequencing problem of 3 models where each model has a demand of 2.

PARENTS

$parent_1$ [A A C C B B]

$parent_2$ [A B C C B A]

A simple single-point crossover between the second and third elements of the parent chromosomes produces the offspring

SIMPLE CROSSOVER (INCORRECT)

$offspring_1$ [A A | C C B A]

$offspring_2$ [A B | C C B B]

Obviously this won't work, since $offspring_1$ contains three As and only one B while $offspring_2$ has three Bs, and only one A. Starkweather (1991) compares several possible solutions to this problem, which are briefly summarized here.

a. *Edge Recombination.* The edge recombination operator is different from other genetic sequencing operators in that it emphasizes adjacency information instead of the order or position of items in the sequence. The *edge table* used by the operator is really an adjacency table listing the connections of elements found in the two parent sequences. The edges are then used to construct offspring in such a way that the isolation of the elements in the sequence is avoided.

For example, for a traveling salesman problem of 10 cities, the tour [b a d f g e c j h i] contains the links [ba, ad, df, fg, ge, ec, cj, jh, hi, ib]. In order to preserve existent links in the two parent sequences a table is built which contains all the existent links in each parent tour. Building the offspring then proceeds as follows: (1) Select a starting element. This can be one of the starting elements from a parent, or can be chosen from the set of elements which have the fewest entries in the edge table. (2) Of the elements that have links to this previous element, choose the element which has the fewest number of links remaining in its edge table entry, breaking ties randomly. (3) Repeat step 2 until the new offspring sequence is complete.

b. *Order Crossover (OX).* As the title implies, here the goal is to preserve the relative order of elements in the sequences to be combined. With this operator, the offspring inherits the elements between the two crossover points, inclusive, from the selected parent in the same order and position as they appeared in that parent. The remaining elements are inherited from the alternate parent in the order in which they appear in that parent, beginning with the first position following the second crossover point and skipping over all elements already present in the offspring.

The following example taken from Haupt & Haupt (2004) demonstrates the order crossover procedure. Consider two parent chromosomes of length 6:

PARENTS

$parent_1$	[3 4 6 2 1 5]
$parent_2$	[4 1 5 3 2 6]

OX begins by choosing two crossover points and exchanging the integers between them. This results in holes (denoted below by *X*'s), which are left in the spaces where integers are repeated. If the crossover points are after the second and fourth integers, the first stage leaves offspring that look like

OX (First Stage)

$$\mathit{offspring}_{1L}\quad [\,X\ 4 \mid \underbrace{\mathbf{5\ 3}}_{J} \mid 1\ X\,]$$

$$\mathit{offspring}_{2L}\quad [\,4\ 1 \mid \underbrace{\mathbf{6\ 2}}_{K} \mid X\ X\,]$$

At this point the holes are pushed to the beginning of the offspring. All integers that were in those positions are pushed off the left of the chromosome and wrap around to the end of the offspring. At the same time the strings *J* and *K* that were exchanged maintain their positions:

OX (Second Stage)

$$\mathit{offspring}_{1M}\quad [\,X\ X\ \mathbf{5\ 3}\ 1\ 4\,]$$

$$\mathit{offspring}_{2M}\quad [\,X\ X\ \mathbf{6\ 2}\ 4\ 1\,]$$

For the final stage the *X*'s are replaced with strings *J* and *K*:

OX (Final Stage)

$$\mathit{offspring}_{1N}\quad [\,\underbrace{\mathbf{6\ 2}}_{K}\,\underbrace{\mathbf{5\ 3}}_{J}\ 1\ 4\,]$$

$$\mathit{offspring}_{2N}\quad [\,\underbrace{\mathbf{5\ 3}}_{J}\,\underbrace{\mathbf{6\ 2}}_{K}\ 4\ 1\,]$$

OX has the advantage that the relative ordering is preserved, although the absolute position within the string is not.

c. *Partially Mapped Crossover (PMX).* In this method, a parent and two crossover sites are selected randomly and the elements between the two starting positions in one of the parents are directly inherited by the offspring. Each element between the two crossover points in the alternate parent is mapped to the position held by this element in the first parent. Then the remaining elements are inherited from the alternate parent. Just as in the OX operator, the section of the first parent which is copied directly to the offspring preserves order, adjacency and position for that section. However, it seems that more disruption occurs when mapping the other elements from the unselected parent. Using the same parent chromosomes of the OX example, Haupt & Haupt (2004) demonstrates the PMX procedure as follows:

If crossover points are between elements 1 and 2, and 3 and 4, then string *K* from *parent*$_2$ is switched with string *J* from *parent*$_1$ as follows:

PMX (Step A)

$$\mathit{offspring}_{1A} \quad [3 \mid \underbrace{\mathbf{1\ 5}}_{J} \mid 2\ 1\ 5]$$

$$\mathit{offspring}_{2A} \quad [4 \mid \underbrace{\mathbf{4\ 6}}_{K} \mid 3\ 2\ 6]$$

All values exchanged between parents are shown in bold type. As it is seen, this process leads to making some values to double and some others to disappear. To remedy this, the switched strings, *J* and *K*, remain untouched throughout the rest of the procedure. The original doubles in *offspring*$_{2A}$ are exchanged with the original doubles in *offspring*$_{1A}$ (the original 4 in *offspring*$_{2A}$ exchanged with the original 1 in *offspring*$_{1A}$, and the original 6 in *offspring*$_{2A}$ with the original 5 in *offspring*$_{1A}$) to obtain the following final solution:

PMX (Step B)

$$\mathit{offspring}_{1B} \quad [3 \mid \mathbf{1\ 5} \mid 2\ \mathbf{4\ 6}]$$

$$\mathit{offspring}_{2B} \quad [\mathbf{1} \mid \mathbf{4\ 6} \mid 3\ 2\ \mathbf{5}]$$

Each offspring contains part of the initial parent in the same position (not highlighted numbers) and includes each integer once and only once.

d. *Cycle Crossover (CX).* This operator preserves the absolute positions of elements in the parent sequence. A parent sequence and a cycle starting point are randomly selected. The element at the cycle starting point of the selected parent is inherited by the child. The element, which is in the same position in the other parent, is searched in the selected parent. When its position is found in the selected parent, the element is inherited to the same position in the child. This continues until the cycle is completed by encountering the initial item in the unselected parent. Any elements which are not yet present in the offspring are inherited from the unselected parent. Note that cycle crossover always preserves the position of elements from one parent or the other without any disruption. The example taken from Haupt & Haupt (2004) demonstrates the CX procedure as follows:

The information exchange begins at the left of the string and the first two digits are exchanged leading to

CX (First Step)

$$\begin{array}{ll} \textit{offspring}_{1W} & [\mathbf{4}\ 4\ 6\ 2\ 1\ 5] \\ \textit{offspring}_{2W} & [\mathbf{3}\ 1\ 5\ 3\ 2\ 6] \end{array}$$

Now that the first offspring has two 4s, the second 4 is exchanged with the other offspring to get:

CX (Second Step)

$$\begin{array}{ll} \textit{offspring}_{1X} & [\mathbf{4}\ \mathbf{1}\ 6\ 2\ 1\ 5] \\ \textit{offspring}_{2X} & [\mathbf{3}\ \mathbf{4}\ 5\ 3\ 2\ 6] \end{array}$$

Since there are two 1s in the first offspring, position 5 is exchanged with the second offspring:

CX (Third Step)

$$\begin{array}{ll} \textit{offspring}_{1Y} & [\mathbf{4}\ \mathbf{1}\ 6\ 2\ \mathbf{2}\ 5] \\ \textit{offspring}_{2Y} & [\mathbf{3}\ \mathbf{4}\ 5\ 3\ \mathbf{1}\ 6] \end{array}$$

The next position to exchange is position 4 where there is a repeated 2:

CX (Fourth Step)

$offspring_{1Z}$ [**4** **1** 6 **3** **2** 5]

$offspring_{2Z}$ [**3** **4** 5 **2** **1** 6]

At this point, 2 is exchanged with the 3 and there are no repeated integers in either string, so the crossover is complete. Now, it can be seen that each offspring has exactly one of each digit, and it is in the position of one of the parents.

e. *Position Based Crossover (PBX).* This operator is intended to preserve position information during the recombination process. Several random locations in the sequence are selected along with one parent; the elements in those positions are inherited from that parent. The remaining elements are inherited in the order in which they appear in the alternate parent, skipping over all elements which have already been included in the offspring. Thus, the operator appears to be similar to OX, except that the elements copied from the selected parent come from random locations in the sequence and not from adjacent locations; although designed as a "position" operator, at preserving a position, it probably is less effective than PMX and CX.

f. *Immediate Successor Relation Crossover (ISRX).* This operator is rarely used in the literature. Kim et al. (2000) states that for MMAL sequencing problems, it has been shown experimentally that a combined use of immediate successor relation crossover (ISRX) operator and inversion operator is superior to other possible operators. ISRX is a crossover operator that uses the information of immediate successor relations in parents. An immediate successor of a gene A is the gene immediately following A. The immediate successor of the last gene is the first one, by definition. This relation can be summarized in a tabular form as shown in Table 2.1, which maintains all the immediate successors for every gene type and the ratio of the immediate successors, called gene ratio. The overall procedure of creating offspring with the ISRX operator is as follows.

Step 1: Construct the initial ISRX table, set $i = 1$, and select an initial gene type (e_i) randomly.

Step 2: Inherit e_i at the i-th position in the offspring. If a complete offspring is built, then stop; otherwise, go to the next step.

Step 3: Update the immediate successors by removing two randomly chosen elements among those having the gene type of e_i, and also update the gene ratio.

Step 4: Increase i by 1, and set e_i to the gene type such that the number of elements having the type is the largest in the row of e_{i-1}. Ties are broken by selecting the one with the smallest gene ratio. When this also ties, one gene type is randomly selected. Return to Step 2.

Suppose two parent strings are (A A B B B C C C C) and (A B C A B C B C C). The ISRX operator begins with setting up the initial ISRX table, as shown in Table 2.1 (a). The first four iterations of the procedure are provided in Table 2.1. Repeating the procedure can produce an offspring of (B C A B C C B C A). The other offspring can be obtained by starting with a different initial gene type.

Table 2.1 ISRX table

	(a)		(b)		(c)		(d)	
Gene Type	Immediate Successor	Gene Ratio	Immediate Successor	Gene Ratio	Immediate Successor	Gene Ratio	Immediate Successor	Gene Ratio
A	ABBB	2	ABB	2	ABB	2	ABB	1
B	BBCCCC	3	BCCCC	2	BCCC	2	BCCC	2
C	CCCAABCA	4	CCCAABCA	4	CCAABCA	3	CCBCA	3
Selected gene type	B (randomly selected)		C		A		B	

2.2.2.5.3 Mutation. Mutation in GAs is a genetic operator in an analogy to biological mutation, which is used to maintain genetic diversity from one generation of a population of chromosomes to the next. Its importance arises from the fact that it allows 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. It also helps to ensure that no point in the search space has a zero possibility of being examined.

Ashlock (2005) gives the formal definition of mutation operator as follows:

A *mutation operator* on a population of genes G is a function

$$\text{Mutate} : G \rightarrow G \qquad (9)$$

that takes a gene to another similar but different gene. Mutation operators are also called unary variation operators.

Crossover mixes and matches disparate creatures; it facilitates a broad search of the space of data structures accessible to a given genetic algorithm. Mutation, on the other hand, makes small changes in individual creatures. It facilitates a local search and also a gradual introduction of new ideas into the population (Ashlock, 2005). A common view in the GA community is that crossover is the major instrument of variation and innovation in GAs, with mutation insuring the population against permanent fixation at any particular locus and thus playing more of a background role. However, the appreciation of the role of mutation is growing as the GA community attempts to understand how GAs solve complex problems. Some comparative studies have been performed on the power of mutation versus crossover; however, it is not a choice between crossover and mutation but rather the balance among crossover, mutation, and selection that is all important. The most promising prospect for producing the right balances over the course of a run is to find ways for the GA to adapt its own mutation and crossover rates during a search (Mitchell, 1999).

Common mutation operators can be summarized as follows.

a. *Single-Point Mutation.* This simple operator changes the value of the string at a single position. When used with binary encoding, if the selected location contains “0”, it becomes “1” (and vice-versa).

b. *Multiple-Point Mutation.* This is basically a single-point mutation which is applied for a fixed number of locations of the chromosome.

c. *Probabilistic (Uniform) Mutation.* This operator is a flexible version of multiple-point mutation, in which every location can be exposed to a mutation with a probability of α.

d. *Lamarckian Mutation.* This is performed by looking at all possible combinations of k or fewer point mutations and using the one that results in the best fitness value.

The point-mutation operators described above are the basic mutation operators which can be easily used with simple binary genetic algorithm. The success of the operators usually depends on the nature of the problem, and therefore several other operators are proposed in the literature such as swapping and shifting.

e. *Swap Mutation.* This operator is performed by exchanging the values at two randomly selected locations.

f. *Shift Mutation.* This operation can be performed by shifting either a single value by moving it through the existing string or the values between two selected locations.

2.2.2.5.4 Inversion. Genetic Algorithm theory as proposed by Holland (1975) was founded on four operators: selection, crossover, mutation and inversion. Although Holland included inversion as one of the defining operators of GAs, in the following years it has been rarely used.

Inversion operator aims to mimic the property from nature that, in general, the function of a gene is independent of its location on the chromosome. This is performed by randomly selecting a segment on the chromosome and then inverting the order of the genes in that segment.

2.2.2.6 GA Parameters

Another important issue in implementing a genetic algorithm is how to set the values for the various parameters, such as the population size, crossover rate, mutation rate and termination criteria.

When defining a GA one needs to choose its components, such as variation operators (mutation and crossover) that suit the representation, selection mechanisms for selecting parents and survivors, and an initial population. Each of these components may have parameters, for instance: the probability of mutation, the tournament size of selection, or the population size. The values of these parameters greatly determine whether the algorithm will find a near-optimum solution and whether it will find such a solution efficiently. Choosing the right parameter values, however, is a time-consuming task. In recent years, this issue has received the attention of many researchers. Eiben et al. (1999) classifies these efforts into two major forms: *parameter tuning* and *parameter control* (Figure 2.5).

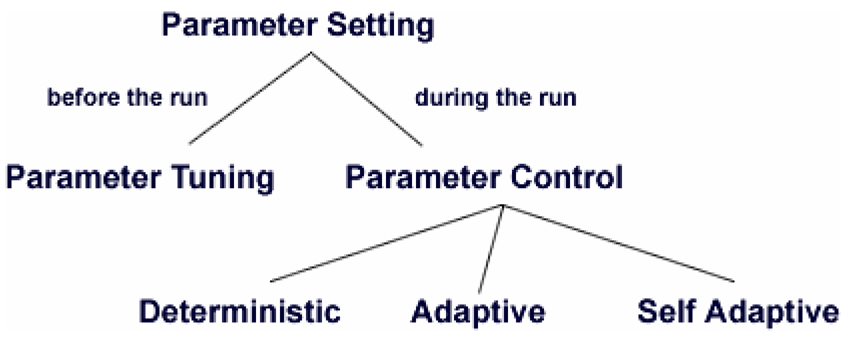


Figure 2.5 Taxonomy for parameter setting in GAs

Parameter Tuning. It refers to the commonly practiced approach that amounts to finding good values for the parameters before the algorithm is run and then keeping these values fixed while the algorithm runs. With this method, typically one parameter is tuned at a time, which may cause some suboptimal choices, since parameters often interact in a complex way. Simultaneous tuning of more parameters, however, leads to an enormous amount of experiments. The technical drawbacks to parameter tuning based on experimentation can be summarized as follows:

- Parameters are not independent, hence trying all different combinations systematically is practically impossible.
- The process of parameter tuning is time consuming, even if parameters are optimized one at a time, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them is significant.

Parameter Control. This method forms an alternative, as it amounts to starting a run with initial parameter values which are changed during the run.

A general drawback of the parameter tuning approach, regardless of how the parameters are tuned, is based on the observation that a run of a GA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast to this spirit. Additionally, it is intuitively obvious that different values of parameters might be optimal at different stages of the evolutionary process. For instance, large mutation steps can be good in the early generations helping the exploration of the search space and small mutation steps might be needed in the late generations to help fine tuning the suboptimal chromosomes. This implies that the use of static parameters itself can lead to inferior algorithm performance.

The straightforward way to treat the above problem is to use parameters whose values may change over time, that is, by replacing a parameter p by a function $p(t)$, where t is the generation counter. As indicated earlier, however, the problem of finding optimal static parameters for a particular problem can be quite difficult, and the optimal values may depend on many other factors (such as the applied recombination operator, the selection mechanism, etc.). Hence designing an optimal function $p(t)$ may be even more difficult. Another possible drawback to this approach is that the changes in parameter values are caused by a deterministic rule triggered by the progress of time, without taking any notion of the actual progress in solving the problem, i.e., without taking into account the current state of the search. Yet researchers, using such simple deterministic rules, managed to improve the performance of evolutionary algorithms on some particular problems. This can be explained simply by superiority of changing the values of parameters: suboptimal choice of $p(t)$ often leads to better results than a suboptimal choice of p.

Finding good parameter values for an evolutionary algorithm is a poorly structured, ill-defined, complex problem. But on these types of problems, GA's are often considered to perform better than other methods! It is thus seemingly natural to use a genetic algorithm not only for finding solutions to a problem, but also for tuning the (same) algorithm to the particular problem. Technically speaking, this amounts to modifying the values of parameters during the run of the algorithm by taking the actual search process into account. Basically, there are two ways to do this. Either one can use some heuristic rules which take feedback from the current state of the search and modify the values of parameters accordingly, or incorporate parameters into the chromosomes, thereby making them subject to evolution. The first option, using a heuristic feedback mechanism, allows one to change the values of parameters, based on triggers other than elapsing time, such as population diversity measures, relative improvements, absolute solution quality, etc. The second option, incorporating parameters into the chromosomes, leaves changes entirely based on the evolution mechanism. In particular, the natural selection acting on solutions (chromosomes) will drive changes in parameter values associated with these solutions. Methods for parameter control are summarized by Eiben et al (1999) as follows (see Figure 2.5):

- *Deterministic parameter control*, in which the values of GA parameters are changed according to some deterministic rule. This rule modifies the parameters deterministically without using any feedback from the search. Usually, a time-varying schedule is used, i.e., the rule will be used when a set number of generations have elapsed since the last time the rule was activated.

- *Adaptive parameter control*, in which the values of GA parameters are changed according to some form of feedback provided by the algorithm.
- *Self-adaptive parameter control*, in which the GA parameters are encoded into the chromosomes and are optimized simultaneously with a cost function. The encoded parameter values which lead to better individuals and are more likely to survive and produce offspring, will be therefore propagated to the next generations.

Between the two forms of parameter setting, parameter tuning is the more preferred one by the researchers. This can be credited to its easier implementation, but it requires a time-consuming trial-error phase before determining the suitable parameter set. The motivations behind the tested parameter sets usually depend on some facts such as the exploration – exploitation issue.

Any efficient optimization algorithm must use two techniques to find a global optimum: *exploration* to investigate new and unknown areas in the search space, and *exploitation* to make use of knowledge found at previously visited points to help finding better points. These two requirements are contradictory, and a good search algorithm must find a trade-off between the two (Beasley, 1993). For GAs, the balance between these two techniques is dictated by the values of crossover (p_c) and mutation (p_m) rates. Increasing values of p_c and p_m promotes exploration at the expense of exploitation. Moderately large values of p_c (0.5-1.0) and small values of p_m (0.001-0.05) are commonly employed values in GA practice (Srinivas & Patnaik, 1994). If the GA uses a parameter control approach, crossover and mutation rates vary with time.

Another parameter of the GA is the population size, which can be defined as the number of data structures in the evolving population. In biology it is known that small populations are likely to die out for lack of sufficient genetic diversity to meet environmental changes or because all members of the population share some defective gene. Analogous effects are possible even in simple evolutionary optimizers. On the other hand, a random initial population is usually jammed with average creatures. In the course of finding the global optimum, we may need a lot of randomness at some computational cost. There is thus a tension between the need for sufficient diversity to ensure solution and the need to avoid processing a population so large that it slows time-to-solution (Ashlock, 2005).

The last parameter in implementing the GA is the terminating condition. Common terminating conditions are as follows:

- a solution is found that satisfies minimum criteria,
- fixed number of generations are reached,
- allocated budget (computation time/money) is reached,
- a plateau such that successive iterations no longer produce better results has been reached,
- manual inspection of solutions by GA user,
- combinations of the above.

2.2.3 Hybrid GA

A hybrid GA combines the power of the GA with the speed of a local optimizer. The GA excels at gravitating toward the global optimum. It is not especially fast at finding the optimum when in a locally quadratic region. Thus the GA finds the region of the optimum, and then the local optimizer takes over to find the optimum. Haupt & Haupt (2004) classify the Hybrid GA methods as follows:

1. Running a GA until it slows down, then letting a local optimizer take over,
2. Seeding the GA population with some local optima found from random starting points in the population,

3. Running a local optimizer on the best solution or on the best few solutions and adding the resulting chromosomes to the population after every specified number of iterations.

2.2.4 Multi-objective GA

This section presents the multi-objective optimization concept and the GA approaches developed to deal with multi-objective optimization problems. First, we'll discuss the multi-objective optimization concept and compare it to single objective optimization. This is followed by the definition of Pareto optimality, which will eventually lead to the related GA approaches for this concept.

2.2.4.1 Multi-objective Optimization

A multi-objective optimization can be defined as the problem of finding a set of design variables (DV) which optimizes a set of objective functions (OF) and simultaneously satisfies a set of constraint functions. A multi-objective optimization problem can be expressed as follows (Augusto, Rabeau, Dépincé, & Bennis, 2006):

Find DV set:

$$x = (x_1, x_2, ..., x_n)^T \in (DVS), \tag{10.1}$$

which minimizes OF:

$$f(x) = (f_1(x), f_2(x), ..., f_k(x))^T \in (OFS), \tag{10.2}$$

and simultaneously satisfies constraints:

$$h_i(x) = 0, \quad i \in [1, ..., q_h] \quad \text{(equality constraints)} \tag{10.3}$$

$$g_i(x) \leq 0, \quad i \in [1, ..., q_g] \quad \text{(inequality constraints)} \tag{10.4}$$

where n is the number of DV (optimization parameters) which belong to the design variables space (DVS); k is the number of OF to be optimized; OF are included in the objective function space (OFS). The space of DV contains both discrete and continuous variables; q_h is the number of equality constraint functions; and q_g is the number of inequality constraint functions.

For multiple-objective problems, the objectives are generally conflicting, preventing simultaneous optimization of each objective. The approaches to deal with multi-objective problems can be placed into two categories: *a priori* and *a posteriori* methodologies. In Figure 2.6 (Augusto et al., 2006), working principles of single-objective a priori and multi-objective a posteriori techniques are compared to each other.

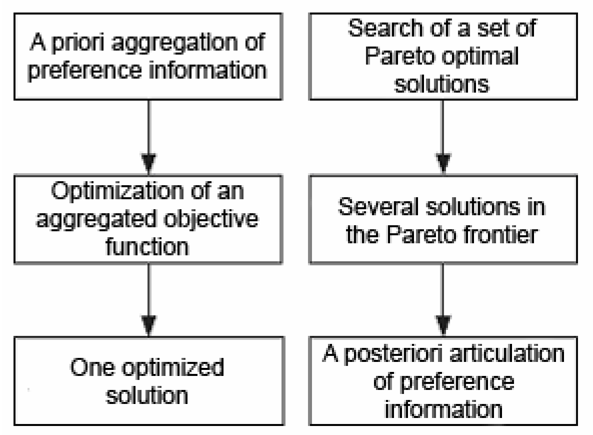

A priori aggregation of preference information
Optimization of an aggregated objective function
One optimized solution
Search of a set of Pareto optimal solutions
Several solutions in the Pareto frontier
A posteriori articulation of preference information

Figure 2.6 Single-objective optimization methods vs. multi-objective

The first set contains *single-objective optimization* techniques which consist of combining the individual objective functions into a single composite function (scalarization) or moving all but one objective to the constraint set. In the former case, determination of a single objective is possible with methods such as utility theory, weighted sum method, etc., but the problem lies in the proper selection of the weights or utility functions to characterize the decision-maker’s preferences. In practice, it can be very difficult to precisely and accurately select these weights, even for someone familiar with the problem domain. Compounding this drawback is that scaling amongst objectives is needed and small perturbations in the weights can sometimes lead to quite different solutions. In the latter case, the problem is that to move objectives to the constraint set, a constraining value must be established for each of these former objectives. This can be rather arbitrary. In both cases, an optimization method would return a single solution rather than a set of solutions that can be examined for trade-offs (Konak, Coit, & Smith, 2006). Consequently, single-objective techniques can be considered as *a priori* methodologies.

Decision-makers often prefer a set of good solutions considering the multiple objectives. This can be accomplished by the second set of *multi-objective optimization* techniques which provide a set of alternative solutions. They are based on an *a posteriori* articulation of preference information in order to make a choice within a set of optimum solutions. These solutions form the Pareto optimal solution set.

2.2.4.2 Pareto Optimality

If we note $f_1^*, f_2^*, ..., f_k^*$ as the individual optima of each respective OF, the utopian solution, $\mathbf{f}^* = (f_1^*, f_2^*, ..., f_k^*)$, is the best theoretical solution which simultaneously optimizes all the objectives. Nevertheless, this utopian solution is rarely feasible because of the existence of constraints. Often $\mathbf{f}^*$ does not belong to the OFS and we use the *Pareto frontier* to define a set of solutions instead of the optimum solution (Figure 2.7). The *Pareto-optimality* is defined as a set, where every element is a solution of the problem for which no other solutions can be better with regard to all the OF. A solution in a Pareto-optimal set cannot be considered better than the others within the set of solutions without including preference information (Augusto et al., 2006).

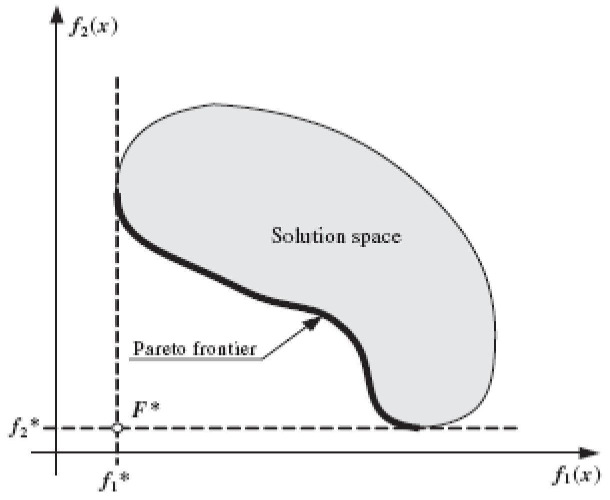


Figure 2.7 Solution space

For a minimization problem, considering two solution vectors x and y, it is said that x dominates y (also written as $x \succ y$) iff:

$$\forall i \in 1,2,...,k : f_i(x) \leq f_i(y) \tag{11.1}$$

and

$$\exists j \in 1,2,...,k : f_j(x) < f_j(y)\,. \tag{11.2}$$

A Pareto optimal set is a set of solutions that are non-dominated with respect to each other. While moving from one Pareto solution to another, there is always a certain amount of sacrifice in one objective(s) to achieve a certain amount of gain in the other(s). Pareto optimal solution sets are often preferred to single solutions because they can be practical when considering real-life problems since the final solution of the decision-maker is always a trade-off. Pareto optimal sets can be of varied sizes, but the size of the Pareto set usually increases with the increase in the number of objectives (Konak et al., 2006).

2.2.4.3 GAs for Multi-Objective Optimization

Genetic algorithms seem particularly desirable to solve multi-objective optimization problems because they deal simultaneously with a set of possible solutions (the so-called population) which allows to find an entire set of Pareto-optimal solutions in a single run of the algorithm (Mansouri, 2005). The ability of GA to simultaneously search different regions of a solution space makes it possible to find a diverse set of solutions for difficult problems with non-convex, discontinuous, and multi-modal solutions spaces. The crossover operator of GA may exploit structures of good solutions with respect to different objectives to create new nondominated solutions in unexplored parts of the Pareto front. In addition, most of the multi-objective GAs do not require the user to prioritize, scale, or weigh objectives. Therefore, GAs have been the most popular heuristic approach to multi-objective design and optimization problems (Konak et al., 2006).

The primary questions when developing genetic algorithms for multi-objective problems are how to evaluate fitness, how to determine which potential solution points should be passed on to the next generation, and how to incorporate the idea of Pareto optimality (Marler & Arora, 2004). Konak et al. (2006) and Marler & Arora (2004) describe the approaches that address these issues and serve as potential ingredients in a genetic multi-objective optimization algorithm. These approaches are briefly summarized in the following section.

2.2.4.3.1 Fitness Functions. In this section, we discuss the alternative ways of dealing with multiple objectives in genetic algorithm approaches.

a. *Weighted Sum Approaches.* The classical and most straightforward approach to solve a multi-objective optimization problem is to assign a weight w_i to each normalized objective function $z_i'(x)$ so that the problem is converted to a single objective problem with a scalar objective function as follows:

$$\min\ z = w_1 z_1'(x) + w_2 z_2'(x) + \ldots + w_k z_k'(x)\,, \tag{12}$$

where $z_i'(x)$ is the normalized objective function $z_i(x)$ and $\sum w_i = 1$. This approach is called the a priori approach since the decision-maker must quantify the relative importance of different objective functions before actually viewing points in the solution space.

Solving a problem with such objective for a given vector $w = \{w_1, w_2, ... w_k\}$ yields a single solution, and if multiple solutions are desired, the problem must be solved multiple times with different weight combinations. The main difficulty with this approach is to select a weight vector for each run (Konak et al., 2006). Selection of weights requires assumptions on the relative worth of the cost functions prior to running the GA. The use of randomly generated weights can help to overcome this problem by stipulating multiple search directions in a single run without using additional parameters.

Implementing this approach in a GA simply requires modifying the cost function to fit the form of Eq. 12 and it does not require any modification to the GA. This approach is not computationally intensive and results in a single best solution based on the assigned weights (Haupt & Haupt, 2004).

b. *Altering Objective Functions.* One of the first treatments of multi-objective genetic algorithms is the *vector evaluated genetic algorithms* (VEGA). The general idea behind this approach involves randomly dividing the original population P_t into k equal subsets, or *sub-populations*, $P_1, P_2, ..., P_k$ where k is the number of objective functions. For each solution, fitness values are assigned according to the objective function z_i which is associated to the sub-population P_i that the solution belongs to. Then, solutions are selected from these sub-populations using the stochastic selection processes for crossover and mutation. Crossover and mutation are performed on the new population in the same way as for a single objective GA.

A similar approach to VEGA is to use only a single objective function which is randomly determined each time in the selection phase. The main advantages of the alternating objectives are the ease of implementation and being computationally as efficient as a single-objective GA. In fact, this approach is a straightforward extension of a single objective GA to solve multi-objective problems. The major drawback of objective switching is that the population tends to converge to solutions which are superior in one objective, but poor at others (Konak et al., 2006).

c. *Pareto-Ranking Approaches.* Pareto-ranking approaches explicitly utilize the concept of Pareto dominance in evaluating fitness or assigning selection probability to solutions. The population is ranked according to a dominance rule, and then each solution is assigned a fitness value based on its rank in the population, not its actual objective function value (Konak et al., 2006).

The means of determining the rank of an individual and assigning fitness values associated with this rank may vary from method to method, but the general approach is common. For a given population, the objective functions are evaluated for each individual. All non-dominated individuals receive a rank of one. Determining whether an individual is dominated or not (performing a non-dominated check) entails comparing the vector of objective function values of the individual to the vector of all other individuals. Then, the individuals with a rank of one are temporarily removed from consideration, and the others that are non-dominated relative to the remaining group are given a rank of two. This process is repeated until all individuals are ranked. Those individuals with the lowest rank have the highest fitness value. That is, fitness is determined such that it is inversely proportional to the rank (Marler & Arora, 2004). Although some of the ranking approaches can be used directly to assign fitness values to individual solutions, they are usually combined with various fitness sharing techniques to achieve the second goal in multi-objective optimization, finding a diverse and uniform Pareto front (Konak et al., 2006).

2.2.4.3.2 Population Diversity. Maintaining a diverse population is an important consideration in multi-objective GA to obtain solutions uniformly distributed over the Pareto front (Konak et al., 2006).

A *niche* in genetic algorithms is a group of points that are close to each other, typically in the criterion space. Niche techniques are methods for ensuring that a population does not converge to a niche, i.e., a limited number of Pareto points. Thus, these techniques foster an even spread of points (in the criterion space). Genetic multi-objective algorithms tend to create a limited number of niches; they converge to or cluster around a limited set of Pareto points. This phenomenon is known as *genetic drift* (or population drift), and niche techniques force the development of multiple niches while limiting the growth of any single niche (Marler & Arora, 2004). Several approaches such as fitness sharing, crowding distance and cell-based density have been devised to prevent genetic drift.

a. *Fitness* S*haring.* Fitness sharing is a common niche technique. Its basic idea is to penalize the fitness of points in crowded areas, thus reducing the probability of their survival to the next generation. The fitness of a given point is divided by a constant that is proportional to the number of other points within a specified distance in the criterion space. In this way, the fitness of all the points in a niche is shared (Marler & Arora, 2004). This approach encourages the search in unexplored sections of a Pareto front, but on the other hand, it also introduces another parameter, niche size, to be taken into consideration.

b. *Crowding Distance.* Crowding distance approaches aim at obtaining a uniform spread of solutions along the best-known Pareto front without using a fitness sharing parameter such as niche size or the k^{th} closest neighbor. As the performance of the above-mentioned fitness sharing methods in maintaining a spread of solutions depends largely on the chosen niche size value, and since these methods require the comparison of each solution with all other solutions in the population which is a computationally intensive work, crowding distance methods have been more desirable to some researchers.

One of the applications of crowding distance is the nondominated searching genetic algorithm (NSGA-II) which is proposed by Deb, Pratap, Agarwal, & Meyarivan, (2002). In NSGA-II, first, the population is sorted according to each objective function value in ascending order of magnitude. Thereafter, for each solution point, the average distance of two points on either side of this point is calculated. This quantity serves as an estimate of the perimeter of the cuboid formed by using the nearest neighbors as the vertices and called the crowding distance. This calculation is continued with other objective functions. The overall crowding distance value is calculated as the sum of individual distance values corresponding to each objective.

NSGA-II uses this crowding distance measure as a tie-breaker in a selection technique called the crowded tournament selection operator: Randomly select two solutions x and y; if the solutions are in the same non-dominated front, the solution with a higher crowding distance is the winner. Otherwise, the solution with the lowest rank is selected (Konak et al., 2006).

c. *Cell-Based Density.* In this approach, the objective space is divided into k-dimensional cells. The number of solutions in each cell is defined as the density of the cell, and the density of a solution is equal to the density of the cell in which the solution is located. This density information is used to achieve diversity similarly to the fitness sharing approach (Konak et al., 2006). Density calculation results in a global density map of the objective function space which can be used to direct the search toward sparsely inhabited regions of the objective function space. This approach is also computationally more efficient than the niching or neighborhood-based density techniques.

2.2.4.3.3 Elitism. Elitism in the context of single-objective GA means that the best solution found so far during the search always survives to the next generation. In this respect, all nondominated solutions discovered by a multi-objective GA are considered elite solutions. However, implementation of elitism in multi-objective optimization is not as straightforward as in single objective optimization mainly due to the large number of possible elitist solutions. It should be noted that unlike the earlier ones most recent multi-objective GA and their variations use elitism. Multi-objective GAs use two strategies to implement elitism: (i) maintaining elitist solutions in the population, and (ii) storing elitist solutions in an external secondary list and reintroducing them to the population (Konak et al., 2006).

a. *Elitism in the population.* Random selection does not ensure that a non-dominated solution will survive to the next generation. A straightforward implementation of elitism in a multi-objective GA is to copy all non-dominated solutions in population P_t to population P_{t+1}, then fill the rest of P_{t+1} by selecting from the remaining dominated solutions in P_t. This approach will not work when the total number of nondominated parent and offspring solutions is larger than the population size, therefore several approaches are proposed to overcome this problem. Obviously, the main advantage of maintaining non-dominated solutions in the population is straightforward implementation. In this strategy, the population size is an important GA parameter since no external archive is used to store discovered non-dominated solutions (Konak et al., 2006).

b. *Elitism with external populations.* This approach requires two sets of solutions to be stored. The second set is used for storing the nondominated solutions and it is updated each time a new solution is created by removing elitist solutions dominated by a new solution or adding the new solution if it is not dominated by any existing elitist solution.

CHAPTER THREE
LITERATURE REVIEW: APPLICATIONS OF GENETIC ALGORITHMS IN MMALS

This chapter presents a review of recent literature related to MMALS using GAs. To classify the relevant literature, we employed two schemes: one for MMALSP (Appendix 1a) and another for GAs (Appendix 1b). It should be noted that the tuple notation used for classifying the current studies on MMALSP is similar to the one proposed in Boysen et al. (2007a). Having focused on studies related to MMALS using GAs, in this study we adopted this tuple notation to the specifications of GAs.

The first scheme considers only the MMALSP and has 4 main titles: i.e. station characteristics, assembly line characteristics, sequencing objectives and the number of production levels. The second scheme is developed to classify the specifications of GA approaches developed to deal with MMALSP. It has 7 main titles; i.e. genetic representation, fitness function, initial population, selection strategies, genetic operators, GA parameters and hybridization. The reader may refer to Chapter 2 for detailed information on these specific titles.

3.1 Classification and Review of Related Literature

In order to highlight the place of this M.Sc study in the current literature, we extensively surveyed the relevant studies.

In previous survey papers, Bard et al. (1992) propose an analytical framework for sequencing MMALs and give the characteristics of sequencing problems such as launching discipline (fixed and variable rates), line movement (paced, unpacked and asynchronous), station restrictions (open- and closed-stations), operator schedules (early and late start) and design objectives (minimizing line length and throughput times). In a more recent study, Boysen et al. (2007a) present a detailed classification scheme for mixed-model sequencing problems. However, this scheme ignores the proposed solution approaches in the literature and only deals with the problem characteristics. The scheme proposed by the authors is based on three elements: (i) characteristics of stations, (ii) characteristics of the assembly line, and (iii) objectives. In this study, we not only consider the characteristics of mixed-model sequencing problems, but also the GA characteristics adopted to deal with this problem. In particular, we adapt the tuple notation and structural framework proposed by Boysen et al. (2007a) and incorporate the specifications of the proposed genetic algorithms into this framework. Table 3.1 summarizes the literature on MMALS using GAs, with the help of the proposed tuple notation (see Appendix 1 for abbreviations).

The first research on the application of GAs to the sequencing problem in MMALs has been carried out by Kim et al. (1996). The authors investigated a suitable genetic representation for the problem, and empirically found a set of genetic control parameters that yield good results. In addition, a new genetic operator, Immediate Successor Relation Crossover (ISRX), was introduced and several existing ones were modified. An extensive experiment was carried out to determine a proper choice of the genetic operators. The performance of the GA was compared with those of heuristic algorithm and of branch-and-bound method. The results of the experimental studies suggested that the proposed GA greatly reduced the computation time and it provided near optimal solution. Nevertheless, the paper aimed at solving single objective sequencing problems. In their next publication, Hyun et al. (1998), the authors developed a method to find diverse Pareto optimal solutions particularly for multiple objective sequencing problems in MMALS. This new method was based on a new evaluation and selection mechanism, called the Pareto stratum - niche cubicle. Three objective functions, i.e. minimizing total utility work, leveling part usage through production rates and minimizing total setup cost were considered simultaneously. The authors compared the proposed algorithm with three existing multiple objective GAs, i.e. vector evaluated genetic algorithm (VEGA), Pareto genetic algorithm (PGA) and niched Pareto genetic algorithm (NPGA) using various test-bed problems. The results indicating the diversity and quality of the solutions revealed that the proposed method outperformed other GAs, especially for large problems involving great variation in setup cost. The characteristics of this study are summarized using the following expressions in Table 3.1:

Problem Type:	[*setup*| | *wo, vpr, setup(c)*|]
Proposed GA Method:	[| *multi, pr, fsh*| | *rank, elit*° | *isrx, inv*| |]

The interpretation of these expressions is as follows:

- Problem type includes standard station characteristics except that set-ups are not ignored; standard assembly line characteristics; three objective functions (i.e., minimizing work overload, minimizing variation of production rates and minimizing the set-up costs) and the standard number of production stages (i.e., final production stage only).

- Proposed GA method uses the standard genetic representation (i.e., real-valued); a multi-objective approach with pareto-ranking and fitness sharing; standard initial population generation (i.e., randomly); rank selection and elitist strategies; immediate successor relation crossover and inversion genetic operators; standard GA parameters and finally a none-hybrid approach.

McMullen et al. (2000) also examined the two objectives of minimization of number of setups and leveling part usage through production rates. These two objectives were combined into a single objective using the weighted sum approach. The proposed GA was compared against the tabu search (TS) and simulated annealing (SA) approaches and was found that the performance of TS was not well in comparison to SA and GA. The authors also stated that GA and SA were equally well performing, although GA approach was computationally more expensive than SA.

Table 3.1 Overview on related literature

Paper	**Problem**	**Problem Type**	**Proposed GA Method**
Kim et al., 1996	Seq.	[*open, var, setup*| | *len*|]	[| | | *rank, elit* | *isrx, inv*| |]
Hyun et al., 1998	Seq.	[*setup*| | *wo, vpr, setup(c)*|]	[| *multi, pr, fsh*| | *rank, elit°* | *isrx, inv*| |]
McMullen et al., 2000	Seq.	[*setup*| | *vpr, setup(n)*|]	[| *multi, wsum*| | | *ox, swap*| |]
Kim et al., 2000	Bal. & Seq.	[| | *wo*|]	coevolutionary [| | | *rou*| *isrx, inv*| *ifix*|]
Rekiek et al., 2000	Bal. & Seq.	[*sto, setup, par*| | *mspan*|]	[| | *hrstc*| | *pmx, pbx, shift*| *min*|]
Miltenburg, 2002	Bal. & Seq.	[| *u*| *vpr*| *mlev°*]	[| | *hrstc*| *rank, elit°* | *two, cx, swap*| *dtprm, ifix, min*|]
Ponnambalam et al., 2003	Seq.	[*setup*| | *wo, cpu, setup(c)*| *mlev(4)*]	[| *multi, pr, fsh*| | *rank, elit°* | *ox, inv*| |]
Mansouri, 2005	Seq.	[*setup*| | *vpr, setup(n)*|]	[| *multi, pr, fsh*| | *sus, elit°* | *ox, swap, inv*| *dtprm, gfix, conv*|]
Yu et al., 2006	Seq.	[| | *vpr, mspan*|]	[| *multi, pr, crwd*| | *rou, elit°* | *ox, inv*| |]
Kim et al., 2006	Bal. & Seq.	[| *u*| *level*|]	endosymbiotic [| | | *tour*| *ox, inv*| *ifix*|]

Being a just-in-time related goal, leveling parts usage rates has been considered by several researchers. Generally it has been assumed that products require approximately same number and mix of parts, and therefore leveling parts usage rates objective was substituted by smoothing variation of production rates at the final assembly. Unlike this general trend, Ponnambalam et al. (2003) focused on four levels of production stages (i.e., product, subassembly, component, raw materials) and assumed that every product could have different part requirements. The authors solved the sequencing problem under the objective of minimizing the variation of parts usage rates. The second part of the research extends the work of Hyun et al. (1998) by incorporating this new objective function in the multi-objective GA framework. The authors reported that multiple-objective GA that uses the selection mechanism of Pareto stratum-niche cubicle performed better than that of weighted sum of multiple objective functions with variable weights.

Since it is easy to implement, most researchers used parameter tuning in their researches. Basically some initial experiments are carried out and the parameter configuration set which gives the best overall results is chosen for the rest of the experiments. Mansouri (2005) stated that using the high rates for crossover and mutation would increase diversity whereas the low rates would improve quality. Therefore it was decided to exploit both advantages by using a changing rate; using constant, high rates at the beginning followed by exponentially decreasing rates from a break point onward. Performance of the developed MOGA was compared against a total enumeration scheme in small problem sets. It was also compared against three search heuristics (i.e. GA, SA, TS) developed for the multi-objective mixed-model sequencing problem in small, medium and large problems. The results revealed that the proposed MOGA outperformed the benchmark algorithms in terms of the quality of solutions. Concerning diversity, no significant difference was observed between the MOGA and the benchmark algorithms.

While previous multi-objective approaches used fitness sharing methods (e.g. niche cubicle) for maintaining the diversity in the population, Yu et al. (2006) preferred dispersed-distance method which awards the solutions located in sparse regions. The authors also used dispersed-distance measure along with pareto-ranks to compute a new fitness value for each individual. The proposed multi-objective GA which adopts the pareto ranking and dispersed-distance calculation methods was claimed to guarantee uniformly dispersed Pareto solutions.

The implementation of the evolution concept to the aggregated problem of balancing and sequencing was originated by Kim et al. (2000). Previously, this aggregated problem has been studied using a hierarchical approach, where the problems were treated sequentially in such a manner that one of them was solved first, and then the other was considered under the constraint of the first solution. Kim et al. (2000) claimed that such a hierarchical approach to an aggregated problem combining multiple sub-problems has a limitation to exploring the solution space and demonstrated this situation by comparing the results of their proposed symbiotic co-evolutionary algorithm to those of the hierarchical GA approach. This new co-evolutionary approach, as contrasted with the hierarchical one, had the ability to simultaneously deal with both balancing and sequencing problems in MMALs. Each of the balancing and sequencing problems was considered as one species, and was characterized by genes which were adequately designed to describe the features specific to the problem. Two populations, i.e. balancing population and sequencing population, were created so that the species coevolved while interacting within and between populations. Since the only considered objective was minimization of utility work for both problems, the proposed approach was a single objective optimization.

Another GA-based approach to this aggregated problem was developed by Rekiek et al. (2000). The authors have developed an iterative hierarchical approach, called balance for ordering (BFO). Proposed algorithm employed two modified genetic algorithms to solve grouping and ordering problems. Initially, the grouping GA is employed to solve balancing problem and then the resulting configuration is used as an input by the ordering GA to solve the sequencing problem. The solutions to both problems (i.e., the effective cycle time resulting from the current line configuration and the model sequence) were compared to the desired cycle time. If the margin was beyond acceptable then the line was rebalanced. This iterative procedure continues until a configuration with an acceptable effective cycle time is obtained.

Another line configuration which was widely adopted by researchers in recent years was U-shaped MMALs. One of these researches considering U-shaped lines for the aggregated problem was presented by Miltenburg (2002). The author proposed a GA-based approach to minimize the weighted sum of the variation of work content in the stations, and the variation of production for models and parts at all facilities. In contrast to Kim et al. (2000), in this study only one population was created for both problems. Single population has been achieved by concatenating balancing and sequencing chromosomes so that a single chromosome contains solutions for both problems. Genetic operators were separately applied to the subsequences of tasks and models. The comparison of the results from the two variations of the proposed GA and randomly created solutions indicated that the quality of the solutions found by the GA was good, especially when the GA was run under different weights.

Aggregated problem for U-shaped lines has been also investigated by Kim et al. (2006). In this research, the authors proposed an extension of their previous symbiotic co-evolutionary algorithm, called the endosymbiotic evolutionary algorithm (EEA). The main difference was the introduction of a third population which includes concatenated chromosomes similar to Miltenburg (2002). Moreover, the authors have considered the U-shaped line constraints. Using absolute deviations of workload as the minimization objective, the characteristics and search capability of EEA were compared and analyzed with those of the following three evolutionary approaches: hierarchical genetic algorithm, SPA (separated and population-based symbiotic evolutionary algorithm), and SNA (separated and neighborhood-based symbiotic evolutionary algorithm). The experimental results showed that for every test-bed problem, all the symbiotic evolutionary algorithms provided better outcomes than the hierarchical approach. They also reported that for every problem instance, EEA showed the best results.

It should be noted that the implementation of adaptive genetic algorithms to MMALSP have been adopted by a few researchers. Miltenburg (2002) has proposed a deterministic parameter control approach in his single-objective genetic algorithm approach to solve the joint problem of balancing and sequencing in MMALs. The author assumed that diversity was more important in early populations than in later ones, hence he set the probability of mutation to 0.5 for the first ten populations and zero thereafter. However, this study lacks a comparative study evaluating the performance of this deterministic approach against existing approaches in the literature.

Experimenting with different parameter settings, Mansouri (2005) observed the effectiveness of high rates on more diversification of solutions along the frontier and hence, he stated that fixed rates did not result in quality as well as diverse frontiers at the same time. Moreover, he noted that the rate of improvement became constantly slower and slower until it ceased to improve amid generations and the low rates for crossover and mutation were found to be effective on improving quality of solutions in a less-diverse frontier. To exploit advantages of both scenarios, Mansouri proposed a new parameter control approach which changed the rates of the operators in such a way that high rates were allowed at the beginning followed by lower rates thereafter. Conducting several experiments on various schemes for changing rates of these two operators, the following formula was found to be promising to determine R_t, i.e. the rate at generation t based on an initial IR rate:

$$R_t = \begin{cases} IR, & t < (K \times Max\_Gen), \\ IR \times \dfrac{1}{t + 1 - (K \times Max\_Gen)}, & (K \times Max\_Gen) \leqslant t < Max\_Gen, \end{cases} \tag{13}$$

where K was a constant that was set to 0.33 after some trials. Initial rates for crossover and inversion operators were set to 0.6 and 0.8, respectively. Since the rates vary according to time (t), this approach is also categorized as a deterministic parameter control method.

3.2 Findings of the Literature Survey

Referring to the Classification Table (Table 3.1), we could list our findings based on problem specifications as follows:

- More than half of the articles surveyed dealt with only the MMAL sequencing problem assuming that the line balancing problem was already solved. The others dealt with both balancing and sequencing problems using various approaches.
- In general, researchers have a general tendency to assume that station borders are closed so that workers cannot move out of their stations. However, this may be too restrictive, since open-stations or hybrid lines composed of both closed and open stations along the line are very common in real world applications.
- Most of the researches have considered deterministic processing times. However, in manual assembly lines, processing times are subject to significant variation.
- Issues which may considerably increase the complexity of the problem such as parallelization and concurrent work have been largely ignored.
- Nearly half of the articles have considered the issue of set-up cost minimization.
- Except two articles (Kim et al., 2006; Miltenburg, 2002) that considered U-shaped line layouts, majority of the surveyed articles considered straight layouts.
- Fixed-rate model launching is a common assumption, whereas workers' movement times or velocities are ignored.
- More than half of the articles surveyed (i.e., 6 out of 10) have considered the JIT objective of uniform parts usage rates.
- All of the researchers dealing with both balancing and sequencing MMALs, adopted single objective optimization approaches ignoring multi-objective nature of this aggregated problem.

Likewise, referring to Table 3.1, we could list our findings based on GA specifications as follows:

- Simultaneous consideration of balancing and sequencing problems have been achieved by three methods, i.e. single population (concatenated balancing and sequencing chromosomes; Miltenburg, 2002), two interacting populations (one for balancing and one for sequencing; Kim et al., 2000), and three interacting populations (one for balancing, one for sequencing and another for concatenated balancing and sequencing chromosomes; Kim et al., 2006).
- For sequencing problems, usually multiple objectives have been taken into consideration, whereas aggregated problem has been solved for single objectives.
- Pareto ranking approaches have been preferred to weighted sum approaches (scalar fitness functions) for multi-objective GAs.
- Population diversity has been generally enhanced by fitness sharing approaches, e.g. niche cubicles. Yu et al. (2006) have used a crowding distance approach as an alternative to fitness sharing approaches.
- The general tendency to generate the initial population is randomization. Only Rekiek et al. (2000) and Miltenburg (2002) have used heuristics to generate initial populations.
- Majority of the articles (i.e., 7 out of 10) have used elitist strategies to preserve the best individuals.
- As crossover and mutation operators, the majority of the researchers have preferred order crossover and inversion, respectively. When used together, these two operators have been shown to work successfully for MMALSP.
- Parameter tuning has been widely adopted by the researchers (i.e. 8 out of 10). Best configurations of GA parameters (e.g. crossover and mutation probabilities, population size, etc.) have been determined through several experiments prior to the run of the algorithm. These configurations are then used throughout their experiments.

In the light of above listed findings, the following research directions can be suggested:

- Single objective optimization can be preferred when the objective unconditionally rules over all the others. However, in balancing and sequencing problems, there are often several conflicting objectives that need to be simultaneously considered. Although there are several research papers that dealt with either multi-objective balancing or multi-objective sequencing problems, the aggregated problem of balancing and sequencing is only dealt with single-objective optimization approaches. Therefore, studying mixed-model assembly line balancing and sequencing problem using multi-objective approaches seems to be an attractive research area.
- The values of GA parameters greatly determine the performance of GAs. Choosing the right parameter values, however, is a time-consuming task. Furthermore, this task may need to be repeated for different instances of the problem. Parameter control has been proposed as an alternative to this tuning of parameters before the run of the algorithm. Miltenburg (2002) and Mansouri (2005) have used deterministic parameter control where crossover and mutation probabilities have been changed according to some deterministic rules. However, adaptive and self-adaptive parameter control mechanisms still remain untested for MMALSP.
- During the literature survey, we noted quite number of studies employing multi-objective GA approaches including Vector Evaluated Genetic Algorithm (VEGA) (Schaffer, 1985), Multi-Objective Genetic Algorithm (MOGA) (Fonseca & Fleming, 1993), Niched Pareto Genetic Algorithm (NPGA) (Horn, Nafpliotis, & Goldberg, 1994), Non-dominated Sorting Genetic Algorithm (NSGA-II) (Deb et al., 2002), Pareto Stratum - Niche Cubicle Genetic Algorithm (PS-NC GA) (Hyun et al., 1998), Multiple Objective Genetic Local Search (MOGLS) (Jaszkiewicz, 1998), Strength Pareto Evolutionary Algorithm (SPEA2) (Zitzler, Laumanns, & Thiele, 2001), Pareto Archive Evolution Strategy (PAES) (Knowles & Corne, 1999) and Multi-Objective Scatter Search (MOSS) (Beausoleil, 2006). However,

MMALS literature lacks applications of most of the above mentioned approaches. Hence, a comparative study evaluating the performance of these multi-objective GA approaches in solving MMALSP may be a valuable contribution to this research area.

- GAs coupled with other heuristics such as local optimizers may perform better than the traditional GAs. These hybridization efforts aim to improve the convergence speed and/or solution diversity, either by building the initial population using solutions obtained by a heuristic or by letting a local optimizer to take over the search process when the GA progress slows down. Although, Rekiek et al. (2000) and Miltenburg (2002) have utilized some initial population heuristics to seed the initial population, current MMALS literature doesn’t have any hybrid applications combining GAs with local optimizers.

3.3 Motivation For This Study

A recent trend in GA based research studies is to employ adaptive or self-adaptive parameter control mechanisms so that a better performance can be gained in dealing with various optimization problems (e.g., Bingul et al., 2000; Chang et al., 2007; Eiben et al., 1999; Herrera & Lozano, 2003; Huang et al., 2006; Liu et al., 2003; Shi et al., 1999; Smith & Fogarty, 1997; Srinivas & Patnaik, 1994; Zhao et al., 2005). During the survey of current literature, as mentioned above, however we have not noted any study employing adaptive or self-adaptive parameter control mechanisms to solve MMAL sequencing problem. Considering the intense competition in global markets to provide a rich product variety at low cost, we believe that developing new methodologies to solve MMAL sequencing problem efficiently is very important. Another aspect which will increase the efficiency of the proposed GA based methodologies is to consider this problem as a multi-objective mixed model sequencing problem. Having motivated by these research gaps this study aims at solving single- and multi-objective MMAL sequencing problems using adaptive genetic algorithm.

CHAPTER FOUR
ADAPTIVE GA BASED APPROACH FOR SOLVING SINGLE OBJECTIVE MIXED MODEL ASSEMBLY LINE SEQUENCING PROBLEM

In this chapter, we focus on solving single objective MMALSP, which concerns with the minimization of part consumption rates. An adaptive GA based approach is proposed for the solution of this problem. The proposed approach tries to maintain a good balance between exploration and exploitation of the solution space by evaluating the progress results periodically.

The MMALSP is solved using two different GA-based approaches. While the first approach is simply implementation of the GA specifications described in Ponnambalam et al. (2003) which ignores any of the adaptive techniques, the second approach is novel to this M.Sc study which modifies the number of elite individuals and probability of mutation during the course of the run. Various sets of computational experiments are carried out to compare performance of the proposed adaptive GA and this commonly used approach.

The rest of this chapter is organized as follows. In section 4.1, the specifications of the proposed adaptive GA based approach are explained. In section 4.2, both the results of comparative experiments carried out on a set of benchmark problems and also the insights gained through these experimental studies are presented.

4.1 Specifications of the Adaptive GA Based Approach

In this section, the components of the proposed adaptive GA based approach are introduced. The general structure of this proposed approach is given in Figure 4.1. As it can be seen from the figure, the genetic search process begins with the random creation of the initial population. Once the initial population is generated, the fitness value of each individual is evaluated using the VPC function presented in Eq. 2. This is followed by the calculation of survival probabilities which are used to select individuals for possible crossover and mutation. The selection process employs a fitness proportionate selection method, the roulette wheel sampling. The selected individual pairs produce the offspring under various crossover and mutation operations.

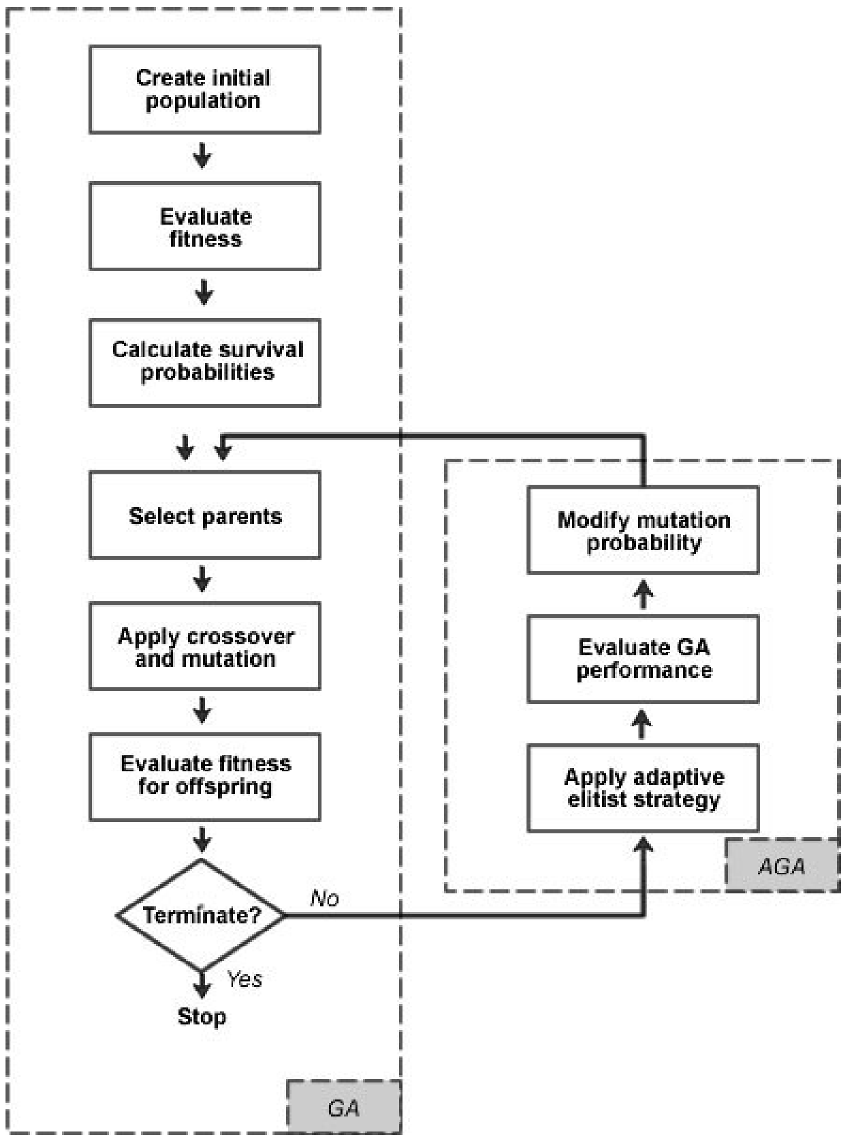

Create initial population
Evaluate fitness
Calculate survival probabilities
Select parents
Apply crossover and mutation
Evaluate fitness for offspring
Terminate?
No
Yes
Stop
GA
Modify mutation probability
Evaluate GA performance
Apply adaptive elitist strategy
AGA

Figure 4.1 General structure of the proposed GA

Following, the quality of these newly created offspring is evaluated. If the termination criteria are met, then this new generation becomes the final generation of the genetic search process and the best individual of the generation is presented. If the termination criteria are not satisfied, then before proceeding to the next generation, an adaptive elitist strategy is applied and individuals of the parent generation that are worthy to survive are transferred to the next generation. Also, once in every 5 generations, the convergence performance is evaluated and the probability of mutation is changed accordingly. After the application of these adaptive techniques, the algorithm continues with the selection process in order to create the next generation.

The details regarding the specifications of the proposed adaptive GA based approach, such as chromosome representation, genetic operators, selection schemes and ranking are explained in the following subsections.

4.1.1 Genetic Representation

Representation of feasible solutions is largely determined by problem-specific characteristics. A string representation, which is a real-valued encoding scheme, is most widely used because of its compatibility with natural gene and genetic operation. In this research, a solution is a sequence of all the models simply listed in their launching order. Obviously, there exists a one-to-one correspondence between the representation space and the solution space. Suppose that during a cycle three types of products, A, B, and C are required to be produced in quantities of 1, 2 and 3, respectively. For instance, a solution alternative to this sequencing problem will be represented as a string of (C B C A C B).

4.1.2 Initial Population

An initial population is necessary to run the algorithm. Either heuristic procedures or random creations can be used to generate strings that form the initial population. Since the scope of this chapter is limited to analysis of the effects of adaptive approaches, it is preferred to create all initial populations randomly.

The initialization process can be summarized as follows:

```
Create an alphabetically ordered sequence according to the given demand
for each model
        Eg. (3A, 2B, 2C)→ AAABBCC
For n=1 to Population Size
        Create a random number associated with every bit in the chromosome
                Eg. AAABBCC→5724369
        Obtain a shuffled sequence by sorting the random numbers in
        ascending order
                Eg. 2345679→ABBACAC
        Inject the shuffled sequence
Next n
```

4.1.3 Fitness Evaluation

The aim of the optimization efforts is the minimization of variations in part consumption rates. Therefore, the fitness of each individual is evaluated using the VPC objective function given by Eq. 2. The pseudocode representation of the fitness evaluation process is given below:

```
VPC=0, w ={1,1,1,1}.
For K=1 to Total Demand
        For G=1 to Production Levels
                For I=1 to Output(I)
                        Calculate $x_{igk}$ : number of units of output i at level g
                        produced during stages 1, 2,…k.
                        Calculate $XT_{gk}$ : total production at level g during
                        stages 1, 2,…k.
                        Calculate $d_{ig}$ : demand for output i at level g; where i
                        = 1, 2,…$n_g$, and g = 2, 3, 4.
                        Calculate $DT_g$ : total demand for production at level g;
                        where g = 1, 2, 3, 4.
                        Calculate $r_{ig} = d_{ig} / DT_g$.
                Variability = $w_g[x_{igk} - (XT_{gk} \times r_{ig})]^2$.
                VPC = VPC + Variability.
            Next I.
        Next G.
    Next K.
    Return VPC.
```

4.1.4 Fitness Conversion and Selection

Before proceeding to the actual selection phase, we need to convert current fitness values, VPCs, to a new fitness value which will be a suitable parameter for minimization objective. This step is required as the selection is performed using a fitness proportionate selection scheme. If the current fitness values were used instead, then individuals with better (lower) VPC values will have lower survival probabilities. In order to circumvent this undesired situation, individuals are assigned new fitness values which are inversely proportional to their original fitness values. These new fitness values will then be used to calculate survival probabilities. The pseudocode representation of the fitness evaluation process is given below:

```
Calculate totalFitness:  the sum of all fitness values in the population.
For every individual i calculate new fitness,
    $newfit_i = 1 - (fitness_i / totalFitness)$;
Calculate totalNewFitness: the sum of all newfit in the population.
For every individual i calculate survival probability,
    $frequency_i = newfit_i / totalNewFitness$;
For every individual i calculate cumulative survival probability,
    $cumulative_i = \sum_{j=1}^{i} frenquency_j$.
```

Once the new fitness values and cumulative survival probabilities are calculated, the selection process can begin. This process employs the roulette wheel selection mechanism that creates a random number, *r*, between 0 and 1, and selects the chromosome, *i*, which satisfies the following condition:

$$cumulative_{i-1} < r \leq cumulative_i \tag{14}$$

This selection process is repeated as many times as the size of the population.

4.1.5 Crossover and Mutation

In this research, a slightly modified order crossover (OX) and inversion (INV) operators have been jointly used as crossover and mutation operators, respectively. The probability of crossover has been fixed at 0.8 while mutation probability starts with an initial value of 0.2 and varies according to the feedback coming from the algorithm.

4.1.6 Adaptive Techniques

There are two adaptive techniques proposed in this research. The first technique is used to determine the number of elites to be transferred to the next generation. This procedure is used at the end of every generation and varying numbers of elites may be transferred depending on the quality of the offspring generation. The second technique is used to modify the probability of mutation depending on the progress results.

4.1.6.1 Adaptive Elitist Strategy

The sequences with better fitness values can be regarded as elite individuals. Such sequences are to be preserved for the next generation. In general, the number of elite individuals that will be transferred to the next generation is fixed before the run of the algorithm. However, this may result in inefficient elite transfers depending on the quality of the newly created generation. Therefore, we propose an adaptive strategy to determine the number of elites at every generation. According to this strategy, assuming a population size of 50, all the parent individuals which have better fitness values than the 5th best child individual are transferred to the next generation. Moreover, to circumvent some extreme situations such as mass transfers in a case of very poor offspring generation, a higher bound is set for the number of elites. This higher bound is half of the population size. This strategy results in varying numbers of elite transfers at every generation and it is directly dependent on the quality of the offspring. It ensures that good solutions do not disappear by chance during the genetic evolution.

4.1.6.2 Adaptive Mutation Probability

In the proposed adaptive approach, evaluation and adaptation are controlled by an AGA controller, which takes recent performance measures and GA control parameters as input, evaluates the progress of evolution, determines the next set of GA control parameters and returns the modified parameters as an output. These modified parameters are then used for the evolution of the next generation by various GA components. This process is illustrated in Figure 4.2.

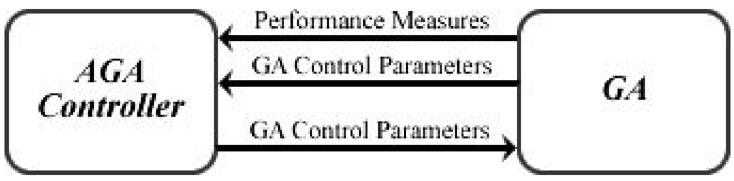


Figure 4.2 Performance evaluation and adaptation

The AGA controller has been developed using a similar approach described in Herrera & Lozano (2003) and it aims to adapt the mutation probability (p_m) during the run of the algorithm. Controller evaluates the algorithm performance in every *G* generations, and p_m is fixed over the generations in these time intervals. It takes into account the p_m value used during the last *G* generations and the improvement achieved on f_b (fitness value for the best element found so far). Then, it computes a new value for p_m, which shall be used during the next *G* generations. To accomplish this task, a set of rules are defined and used by the controller. In Herrera & Lozano

(2003), the authors have employed a set of fuzzy rules; in other words, they fired multiple rules simultaneously for the same input. In this study, unlike theirs, we left out the fuzzy logic and fired one rule at a time.

In short, the goal of the AGA controller is to observe the effects of the value of p_m on performance of the GA during G generations, and produce a new value for p_m that properly replies against a possible poor rate of convergence, or allows the current performance to be improved even more in the case of past suitable progress.

4.1.6.2.1 Required Input. There are two categories of controller input as seen in Figure 4.2. These are performance measures (e.g., convergence of the objective function values) which are evaluated after every G generations and GA control parameters (e.g., crossover and mutation operators, probabilities, etc.) which have been modified G generations before.

Convergence (*CM*) is the measure of the progress made in best objective function values during a number of generations. For the single-objective problem, convergence measure can be calculated using the following equation:

$$CM = f^b \big/ f^0 \tag{15}$$

where f^b is the objective function value of the current best element found so far and f^0 is the objective function value of the best element found before the last G generations. Since an elitist strategy is used, *CM* belongs to [0, 1]. If *CM* is high, then convergence is high, i.e. no progress is made during the last G generations, whereas if it is low, the GA finds a new best element, which outperforms the previous one.

The proposed algorithm uses a fuzzy concept in order to determine if a given *CM* value should be treated as low or high. This requires the use of a set of linguistic labels associated with membership functions. The set of linguistic labels for *CM* is

{*Low*, *High*}. The *Low* label corresponds to a *CM* with a value of 0.8 or lower, whereas the *High* label corresponds to a *CM* with a value of 0.95 or greater. If the *CM* value is in the (0.8, 0.95) range, then both labels are possible. In such a case, label is determined using a random number (*r*) as follows:

$$CM \text{ label} = \begin{Bmatrix} low, & r \leq \dfrac{0.95 - CM}{0.95 - 0.8} \\ high, & otherwise \end{Bmatrix} \quad (16)$$

Numerical representations of the linguistic *CM* labels are depicted in Figure 4.3a.

The mutation probability is used as the GA control parameter. Mutation helps to maintain the diversity in the population and it also helps to prevent the algorithm getting stuck in local optima. Thus, by adapting the mutation probability during the run, we hope to improve the algorithm performance. The set of linguistic labels associated with mutation probability is {*Low*, *Medium*, *High*}. Each of these labels corresponds to a triangular distribution as depicted in Figure 4.3b. For example, the *low* label corresponds to a triangular distribution with a lower limit of 0.10, a mode of 0.15 and an upper limit of 0.20.

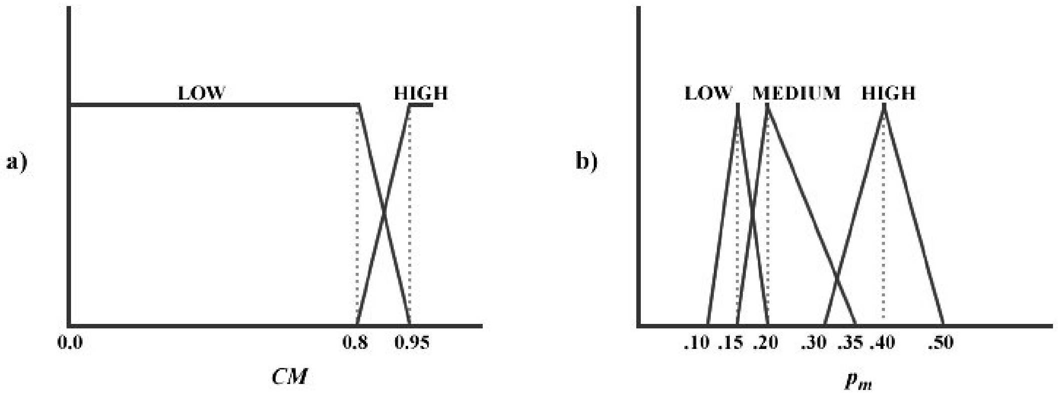


Figure 4.3 Meanings of the linguistic terms

When combined with the above-mentioned *CM* performance measure, the current mutation probability (which is used for the last *G* generations) forms the necessary input for the next step – progress evaluation.

4.1.6.2.1 Progress Evaluation and Output. The goal of this step can be stated as observing the effects of several input types on the GA performance during the last *G* generations, and producing a new set of parameter values that properly replies against a poor performance in a specific measure, or allows performance to be improved even more. The proposed AGA controller attempts to accomplish this task using a set of rules, which describes the relation between the inputs and outputs. Table 4.1 shows the set of rules used by the proposed controller.

Table 4.1 Rule base for the control of p_m

Rule	*CM*	p_m	p_m'
1	High	Low	Medium
2	High	Medium	High
3	High	High	Low
4	Low	Low	Low
5	Low	Medium	Low
6	Low	High	Medium

The rules presented in Table 4.1 are developed by using the following heuristics:

- **Heuristic 1**. This heuristic involves decreasing the value of p_m when progress is made and increasing it when there are no improvements. If a stationary state is detected meaning that the *CM* is high, this could be due to very low value of p_m inducing a premature convergence, with the search process being trapped in a local optimum. Heuristic 1 deals with this problem by increasing the value of p_m so that more diversity is introduced with the possibility of escaping from the local optimum. It should be noted that the rules 1, 2, 4, 5 and 6 listed in Table 4.1 employ this heuristic.
- **Heuristic 2.** This heuristic involves decreasing the value of p_m if there is still no progress while it is high. Another possible cause of a poor performance

may be the use of a too high value for p_m, which does not allow the convergence towards better individuals. Heuristic 2 deals with this problem by decreasing the value of p_m value when both *CM* and p_m are high. It should be noted that the rule 3 listed in Table 4.1 employs this heuristic.

The progress evaluation can be summarized as follows: first the *CM* value is calculated, and then its linguistic label is determined as *low* or *high*. This information is matched with the current p_m label in Table 4.1 and the resulting p_m' label is found. Finally, the new mutation probability is calculated according to the label of p_m'.

4.2 Computational Experiments and Analysis

To evaluate the performance of the proposed adaptive algorithm, we carried out experiments on both small and large sized problems. Moreover, we compared the performance of the adaptive algorithm with the pure (non-adaptive) GA described in Ponnambalam et al. (2003). These two algorithms along with their main differences are listed in Table 4.2.

Table 4.2 The algorithms compared

	Standard	**Adaptive**
Algorithm definition	[l l l *elit* l *ox, inv*l l]	[l l l *elit* l *ox, inv*l *adprm* l]
Mutation rate	Fixed to 0.2	Varies in the range of [0.10, 0.50]
Number of elites	10	Varies in the range of [0, 25]

4.2.1 Benchmark Problems

The experiments have been carried out using a set of 60 problems. The problems were generated so as to provide different sets of conditions in mixed-model assembly line sequencing problems. The first factor considered is the number of models produced on the assembly line; three cases, i.e. 3-5 models, 6-8 models and 9-11 models, are considered. The second factor considered is the MPS (minimum part set), which is to be repeated to satisfy the demand for a specified planning horizon. This factor is considered at three levels: (1) a random number between 10 and 20; (2) a random number between 20 and 30, and (3) a random number between 30 and 40, all chosen from a uniform distribution. The number of subassemblies, components and raw materials are chosen in the range of [3, 6]. Using these conditions, we have divided 60 problems into 6 problem sets, each containing 10 problems. These sets, accompanied by their average solution space, can be seen in Table 4.3.

Table 4.3. Problem sets

Total Demand	**Number of Models**		
	[3, 5]	[6, 8]	[9, 11]
[10, 20]	**I**	**III**	
[20, 30]	**II**	**IV**	**V***
[30, 40]			**VI***

**additional problem sets*

P. Set	**Average number of solutions**
I	$1.6 * 10^{7}$
II	$3.4 * 10^{9}$
III	$8.2 * 10^{12}$
IV	$2.1 * 10^{17}$
V	$6.2 * 10^{19}$
VI	$1.0 * 10^{30}$

The data for the first four problem sets are taken from Ponnambalam et al. (2003) in which the generated problems were solved under the objective of minimizing the variation of part consumption rates. Moreover, to evaluate the performance of the proposed algorithm on large size problems, we generated the problem sets V and VI (see Table 4.3).

Using the classification schemes proposed in Chapter 3, we can define all these problems as follows:

$$[\ setup \mid \mid cpu \mid mlev^{4}].$$

4.2.2 Parameter Setting

The algorithm is programmed in C, and implemented on a PC with a 2GHz CPU. The values of genetic parameters have been determined through preliminary experiments. For all problems, the population size has been set to 50. The initial rates for crossover and mutation are set to 0.8 and 0.2 respectively. While the crossover rate is fixed throughout the generations, the mutation rate varies in the range of $[0.10,\ 0.50]$. For the non-adaptive GA algorithm, the number of elites is set to 10. Adaptive controller is called once in every 5 generations and finally, for the termination of the algorithm, the maximum number of generations is set to 100.

4.2.3 Analysis and Discussion of the Results

All of the instances in the problem sets were solved using both standard GA and adaptive GA algorithms. In order to get more reliable results, each algorithm has been run 10 times for each of the problem instances and the best and average performances for the solutions were recorded. The comparison of the performance of two approaches is based on both the best and also average values.

Tables 4.4-4.9 present the results of the computational experiments. For each of the 60 problems, the best and average VPC values are given for both standard and adaptive approaches. The percent of improvement achieved by using the adaptive approach is also given in each column. These improvement rates are calculated as follows:

$$\frac{VPC_{std} - VPC_{adp}}{VPC_{std}} \times 100 \tag{17}$$

where VPC_{std} is the VPC value obtained by the standard approach and VPC_{adp} is the VPC value obtained by the adaptive approach. The last column of each table gives the average improvement rates achieved in that problem set.

Table 4.4 Results for probems 1-10

	1	2	3	4	5	6	7	8	9	10	Average
Standart											
Best	18789,9	33274,2	58144,6	24696,8	2487,2	101843,2	5109,2	37836,2	8373,3	34792,5	
Average	19001,2	33973,3	58144,6	24696,8	2487,2	101892,3	5144,9	37866,5	8968,1	36465,6	
Adaptive											
Best	18789,9	33274,2	58144,6	24696,8	2487,2	101843,2	5109,2	37836,2	8205,4	34636,0	
Average	18789,9	33828,8	58144,6	24696,8	2488,4	101843,2	5180,6	37926,9	8443,8	34853,6	
Improvement											
Best	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	2,01%	0,45%	0,25%
Average	1,11%	0,43%	0,00%	0,00%	-0,05%	0,05%	-0,69%	-0,16%	5,85%	4,42%	1,10%

Table 4.5 Results for probems 11-20

	11	12	13	14	15	16	17	18	19	20	Average
Standart											
Best	32499,5	91294,0	42618,9	39713,6	62746,9	56634,2	29467,1	31176,8	29764,9	19002,3	
Average	34282,8	99723,4	45341,0	43213,0	65052,4	63943,2	31072,2	34765,8	32242,9	20653,1	
Adaptive											
Best	32499,5	82684,4	39390,4	39713,6	57690,0	56537,6	28000,2	28194,8	26422,5	17191,5	
Average	33151,0	86484,8	40663,5	41185,1	59516,4	57890,0	29185,7	30535,1	28823,3	18237,0	
Improvement											
Best	0,00%	9,43%	7,58%	0,00%	8,06%	0,17%	4,98%	9,56%	11,23%	9,53%	6,05%
Average	3,30%	13,28%	10,32%	4,69%	8,51%	9,47%	6,07%	12,17%	10,61%	11,70%	9,01%

Table 4.6 Results for probems 21-30

	21	22	23	24	25	26	27	28	29	30	Average
Standart											
Best	25532,5	82285,2	35495,6	29338,0	53684,7	53684,7	15525,9	16552,1	16974,8	26640,8	
Average	28938,1	86320,1	36911,9	30268,4	55557,7	55557,7	15744,2	16853,8	18299,2	30997,1	
Adaptive											
Best	25532,5	79305,3	35281,3	29338,0	53684,7	53684,7	15525,9	16552,1	15974,6	27845,2	
Average	26349,0	82268,3	36216,3	29510,3	54590,5	54590,5	15825,7	16884,1	17107,9	29183,2	
Improvement											
Best	0,00%	3,62%	0,60%	0,00%	0,00%	0,00%	0,00%	0,00%	5,89%	**-4,52%**	0,56%
Average	8,95%	4,69%	1,88%	2,50%	1,74%	1,74%	-0,52%	-0,18%	6,51%	**5,85%**	3,32%

Table 4.7 Results for probems 31-40

	31	32	33	34	35	36	37	38	39	40	Average
Standart											
Best	33799,1	6922,8	166179,1	62113,0	37264,8	70552,3	214156,3	59891,9	129720,7	56268,8	
Average	38276,9	7352,1	180529,7	69197,2	40789,7	79327,1	239456,3	65672,9	139835,6	63269,9	
Adaptive											
Best	32701,2	6898,4	141574,9	55564,6	34099,3	66140,6	196741,7	55955,4	117859,1	52395,0	
Average	34843,3	7117,0	155697,6	60329,8	35722,9	71000,2	209887,2	60209,4	124338,8	56277,3	
Improvement											
Best	3,25%	0,35%	14,81%	10,54%	8,49%	6,25%	8,13%	6,57%	9,14%	6,88%	7,44%
Average	8,97%	3,20%	13,76%	12,81%	12,42%	10,50%	12,35%	8,32%	11,08%	11,05%	10,45%

Table 4.8 Results for probems 41-50

	41	42	43	44	45	46	47	48	49	50	Average
Standart											
Best	396689,5	233520,0	174574,5	579839,1	656388,7	662774,9	421073,7	351627,2	20699,0	188876,5	
Average	427642,1	254572,8	187735,9	627423,4	713458,9	749974,8	431897,9	412425,9	22257,7	202115,8	
Adaptive											
Best	374312,9	210208,0	165702,0	548879,9	544539,5	635944,6	391315,6	307616,4	17817,5	168351,6	
Average	398617,3	222725,3	175620,9	606382,8	591872,2	682996,6	407504,7	351429,4	20000,8	181134,9	
Improvement											
Best	5,64%	9,98%	5,08%	5,34%	17,04%	4,05%	7,07%	12,52%	13,92%	10,87%	9,15%
Average	6,79%	12,51%	6,45%	3,35%	17,04%	8,93%	5,65%	14,79%	10,14%	10,38%	9,60%

Table 4.9 Results for probems 51-60

	51	52	53	54	55	56	57	58	59	60	Average
Standart											
Best	348494,0	407110,9	129361,0	2837580,1	145626,2	879397,2	445576,1	93924,6	108777,5	387611,8	
Average	415278,3	417454,6	149703,9	3088061,7	172189,6	988347,2	515615,7	105789,1	122648,4	428665,8	
Adaptive											
Best	317629,2	344151,6	123938,3	2250754,1	135832,0	782415,1	365076,0	87983,5	100112,9	322383,4	
Average	343039,8	373339,1	134109,2	2557965,8	145739,6	855853,2	411040,0	94587,9	105830,8	357715,7	
Improvement											
Best	8,86%	15,46%	4,19%	20,68%	6,73%	11,03%	18,07%	6,33%	7,97%	16,83%	11,61%
Average	17,40%	10,57%	10,42%	17,17%	15,36%	13,41%	20,28%	10,59%	13,71%	16,55%	14,54%

It can be seen from these tables that except for one problem instance (i.e., problem 30th), the adaptive approach provided the best VPC values. However, considering the general improvement of adaptive algorithm over standard algorithm for this problem instance (see Table 4.6, average improvement of 5.85%), this result can be interpreted as coincidental.

In the terms of average values, out of 60 problems, adaptive approach provided the best results in 55 problems. Although the standard approach was the one with better results in the other five problem instances (i.e., 5, 7, 8, 27 and 28), the difference was always under 1%. This can be attributed to the fact that all of these 5 problems belong to the first two problem sets whose possible solution spaces are rather small compared to the other four sets. It should also be noted that for these 5 problems, both the standard and adaptive approaches were able to reach the same best VPC values in several replications.

In order to derive a general perspective, all results in Tables 4.4-4.9 are summarized in Figure 4.4. This figure provides the average improvement values achieved by the implementation of adaptive techniques for each problem set. The improvements on best and average values are gathered from the last column of each table.

The figure shows that the contribution of the adaptive techniques grows as the problem size gets larger. In the first two problem sets that contain rather small-sized problems, usually both approaches reach to the same VPC values. However, the gap between the standard and adaptive approach widens as the problems become complex. Especially in the sixth problem set, the improvement rates may reach up to 20%.

Regarding to these findings, it could be stated that the adaptive parameter control approaches that alter the values of GA parameters depending on the feedback taken from the algorithm during the course of the run, has the potential to improve the performance of the pure GA approaches. Particularly, for large-sized problems, adaptive methods have an unneglectable impact on the performance of the solution method.

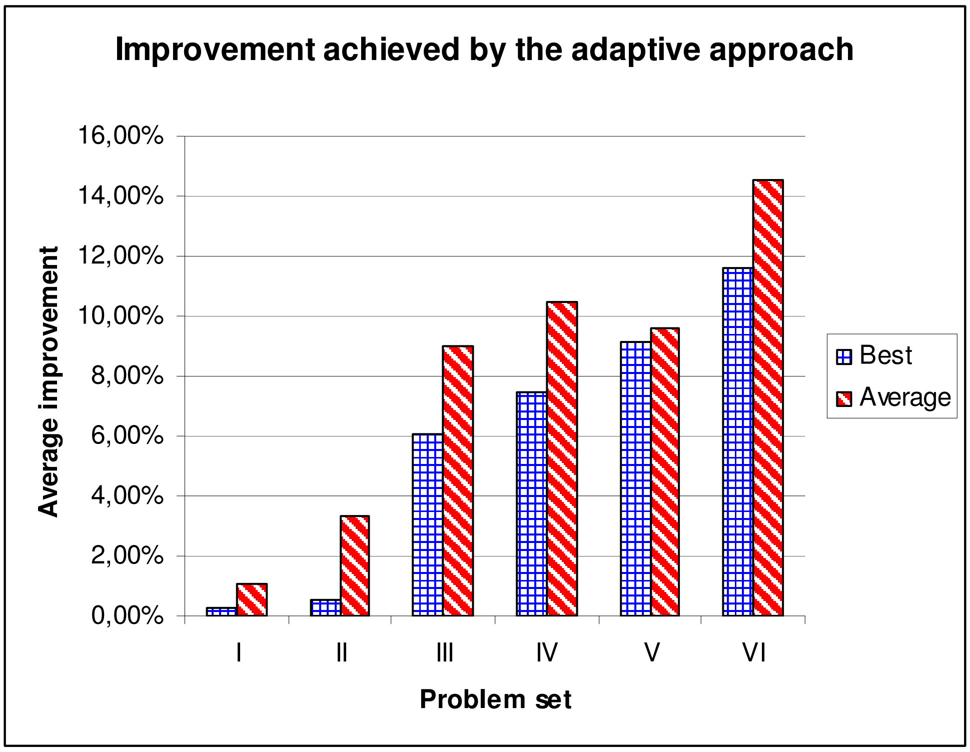


Figure 4.4 Improvement by problem set

CHAPTER FIVE
ADAPTIVE GA BASED APPROACH FOR SOLVING MULTI-OBJECTIVE MIXED MODEL ASSEMBLY LINE SEQUENCING PROBLEM

In this chapter, an adaptive GA based approach is introduced to solve multi-objective MMALSP. The proposed approach combines the features of the multi-objective GAs (MOGAs) and adaptive GAs (AGAs) in order to provide alternative solutions for the decision makers. An adaptive elitist strategy and a scheme for changing the probability of mutation have been developed for this purpose.

In order to evaluate the performance of the proposed adaptive GA algorithm, several experiments are carried out. Moreover, the results of the AGA are compared to the results of a standard (non-adaptive) MOGA that was proposed in Ponnambalam et al. (2003). The experiments also incorporate two initial population heuristics to observe the effects of employing different initialization schemes on the performance of the proposed algorithm.

The rest of this chapter is organized as follows: In section 5.1, the proposed adaptive based approach is discussed in detail and in section 5.2, the results of comparative experiments carried out on a set of benchmark problems are presented.

5.1 Specifications of the Adaptive MOGA Based Approach

This section discusses in detail how the two algorithms, adaptive GA and multi-objective GA were combined to solve MMALSP. The general structure of the proposed approach is given in Figure 5.1.

As it is seen in Figure 5.1, there are three major components in the proposed algorithm. The main component, the GA, incorporates the multi-objective GA (MOGA) which ranks the individuals and adaptive GA (AGA) which modifies the values of the GA parameters.

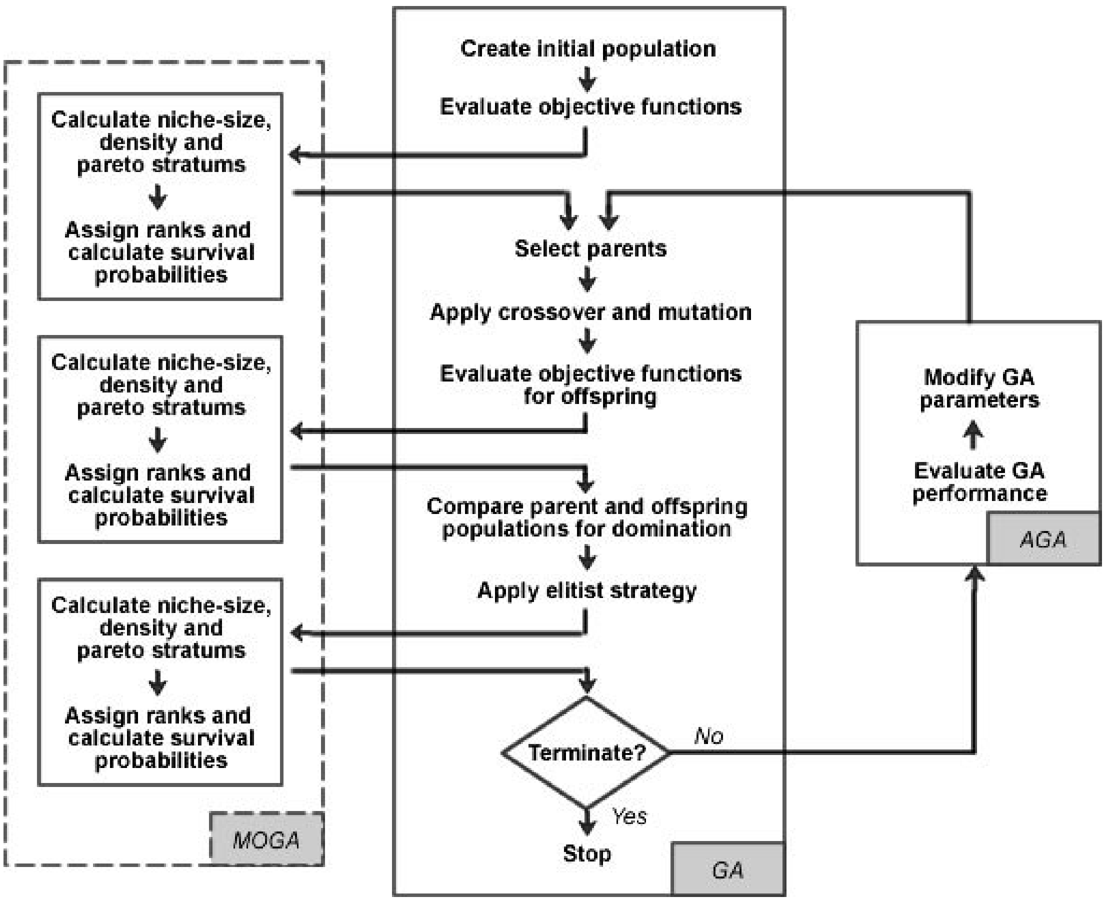

Create initial population
Evaluate objective functions
Select parents
Apply crossover and mutation
Evaluate objective functions for offspring
Compare parent and offspring populations for domination
Apply elitist strategy
Terminate?
No
Yes
Stop
GA
Calculate niche-size, density and pareto stratums
Assign ranks and calculate survival probabilities
Calculate niche-size, density and pareto stratums
Assign ranks and calculate survival probabilities
Calculate niche-size, density and pareto stratums
Assign ranks and calculate survival probabilities
MOGA
Modify GA parameters
Evaluate GA performance
AGA

Figure 5.1 General structure of the proposed multi-objective adaptive GA

A pseudocode representation of the proposed algorithm is given below:

```
(main) Read GA parameters and options.
    (mainloop) Read problem data.
        (initialize) Create and return an initial population.
        (evaluate) Calculate and return objective function values for all individuals
        (constructCubicle) Calculate niche-size, density and pareto stratums for the current population, assign ranks and survival probabilities according to these values (those with higher pareto stratum and lower density values are assigned higher ranks).
    (mainloop) Start generations loop (loop for a pre-determined number of generations)
```

```
        (mainloop) Modify GA parameters if necessary (crossover and/or
        mutation operators, crossover and/or mutation probabilities, number
        of elites to be transfered, etc.)
            (evolve) Choose parents, choose appropriate GA operators,
            apply crossover and mutation, return a candidate population.
            (evaluate) Calculate and return objective function values for all
            individuals of the candidate population.
            (constructCubicle) Calculate niche-size, density and pareto
            stratums for the candidate population, assign ranks and
            survival probabilities according to these values.
            (domCompare) Compare the individuals of previous and
            candidate populations, mark those which are dominated by an
            individual from the other population.
            (eliteMulti) If adaptive elite system is enabled, transfer all the
            unmarked (by the domCompare function in the previous step)
            individuals of the previous generation into newly created
            generation. If it is disabled, transfer a pre-specified number of
            individuals from the previous generation into the new
            generation (here, the selection depends on ranks of the
            individuals, which were assigned by the previous
            constructCubicle function call).
            (eliteMulti) After elite transfers are completed, populate the
            remaining quota with the individuals of the candidate
            population, giving priority to individuals with higher ranks.
            (constructCubicle) Calculate niche-size, density and pareto
            stratums for the new generation, assign ranks and survival
            probabilities according to these values.
        (mainloop) End generations loop if pre-determined number of
        generations is reached.
        (mainloop) Display and print the best sequences found.
(main) End program.
```

The expression before each step in the pseudocode refers to the program module which is responsible from that operation. A chart presenting the code structure (program modules) of the proposed algorithm is given in Appendix 2.

The details regarding the specifications of the proposed multi-objective adaptive GA based approach, such as chromosome representation, genetic operators, selection schemes and ranking are explained in the following sections.

5.1.1 Genetic Representation and Initial Population

Genetic representation of feasible solutions is the same as in the previous chapter. However, the creation of initial population introduces two new heuristics in addition to the totally random creation. As a result, some of the individuals in the initial population are generated using two different heuristics, and the rest are randomly generated.

The overall initialization process can be summarized as follows:

```
Generate and inject palindromic sequences (IPH-1)
        Eg. (3A, 2B, 2C)→ ABCACBA and ABCABCA
Create an alphabetically ordered sequence according to the given demand
for each model
        Eg. (3A, 2B, 2C)→ AAABBCC
Inject the alphabetically ordered and reverse-ordered sequences (IPH-2)
        Eg. (3A, 2B, 2C)→ AAABBCC and CCBBAAA
For n=1 to (Population Size – (2 + 2))
        Create a random number associated to every bit in the chromosome
                Eg. AAABBCC→5724369
        Obtain a shuffled sequence by sorting the random numbers in
ascending order
                Eg. 2345679→ABBACAC
        Inject the shuffled sequence
```

Next n

The heuristics used in the initialization process are explained in the following sections.

5.1.1.1 Initial Population Heuristic – 1 (IPH-1)

During the pilot studies, it has been noticed that the optimal sequences for some of the smaller sized problems (when variation in part consumption rates were to be minimized) were palindromic sequences. In the context of this research, a palindrome is a sequence of models that has the property of resulting in the same sequence regardless of starting the reading either in forward or backward direction; for instance ABCACACBA is a palindromic sequence. Based on this finding, in this study we used the heuristic IPH-1 to inject some palindromic sequences into the initial population hoping that this will help to start the search from a better point.

The procedure to generate the palindromic sequences is given below:

i. For each product i of type m , compute the following position value, pv_{mi},

$$pv_{mi} = \frac{i\sum_{m=1}^{M} d_m}{d_m + 1}, \quad i = 1,2,\ldots, d_m \,;\; m = 1,2,\ldots,M. \qquad (18)$$

where d_m is the demand of model m and M is the number of different models.

ii. Sort the products according to their position value (pv_{mi}) in ascending order.

Eq. 18 allows calculating pv_{mi} values that are used to *evenly* distribute the products of a specific model, in the production sequence. Later, when the products are sorted in the second step, a palindromic sequence is obtained. During the sorting process, there may be situations where two or more products share the same position value. In such cases, ties are broken using two alternative methods each resulting in different production sequences: (i) the tied products are always sorted in alphabetical order, (ii) the tied products in the first half of the production sequence are sorted alphabetically whereas the ones in the second half are sorted in reverse-alphabetical order. Although the symmetry in the second method provides the palindromic sequences that are searched for, in some cases, the first method provides sequences with better VPC results. Therefore, both sequences are injected into the initial population to ensure the better individual gets included. The following example demonstrates this procedure.

Example. Suppose that during a cycle four types of products, A, B, C and D are required to be produced in quantities of 3, 4, 3 and 2, respectively. Using Eq. 18, the position values are calculated as follows:

$$\left\{\begin{array}{ll} pv_{A1} = 3 & pv_{C1} = 3 \\ pv_{A2} = 6 & pv_{C2} = 6 \\ pv_{A3} = 9 & pv_{C3} = 9 \\ pv_{B1} = 2.4 & pv_{D1} = 4 \\ pv_{B2} = 4.8 & pv_{D2} = 8 \\ pv_{B3} = 7.2 & \\ pv_{B4} = 9.6 & \end{array}\right\}$$

When the position values are sorted in ascending order, the following graphic can be obtained.

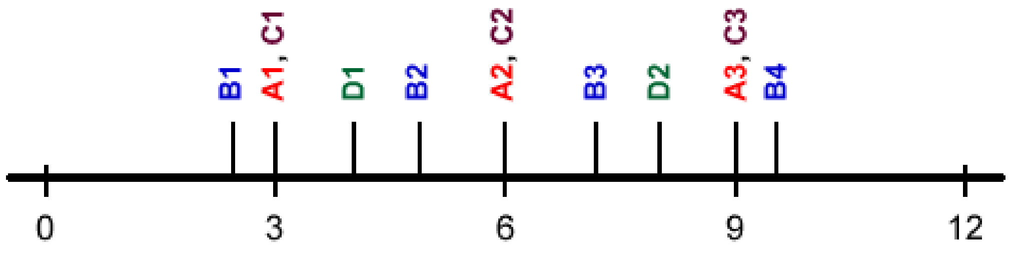


Figure 5.2. Products sorted according to pv_{mi}

As it can be seen from Figure 5.2, the distribution of models is symmetric with respect to 6. However, products of type A and C share the same position values, {3,6,9}. When the tie-breaker methods are applied, the following two sequences are obtained.

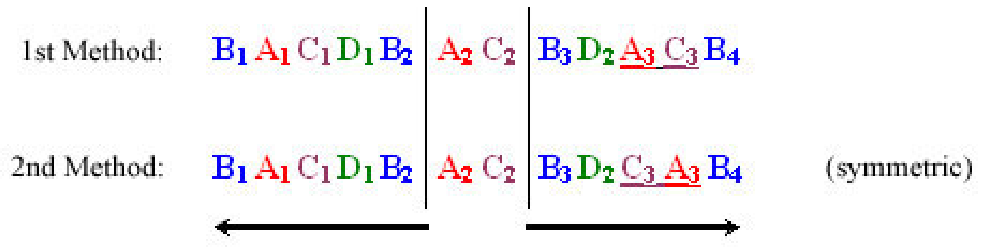

1st Method:
B_1 A_1 C_1 D_1 B_2 | A_2 C_2 | B_3 D_2 A_3 C_3 B_4
2nd Method:
B_1 A_1 C_1 D_1 B_2 | A_2 C_2 | B_3 D_2 C_3 A_3 B_4
(symmetric)

Figure 5.3 Sequences obtained by using IPH-1

Using Eq.2, the VPC values of these two sequences are calculated as 3188.56 and 2509.75, respectively. The implementation of GA to this problem results in a solution with VPC value of 2487.15 which is very close to the one obtained from the second sequence of IPH-1 (2509.75).

5.1.1.2 Initial Population Heuristic – II (IPH-2)

The second heuristic is used to inject two sequences into the initial population. These sequences are alphabetically ordered and alphabetically reverse-ordered sequences of the models to be produced. In doing so, it was hoped that the setup cost would be reduced as the injected sequences contained the minimum number of model changes.

5.1.2 Objective Functions

To evaluate the fitness of each individual, the following three objective functions are employed: (i) total utility work (Eq. 1), (ii) variation of parts consumption rates (Eq. 2) and (iii) total setup costs (Eq. 3). Given a feasible sequence, the fitness of individual is evaluated using these three functions.

5.1.3 Ranking and Selection

Before proceeding to selection, each individual must be given a selection probability. For this purpose, there are several approaches proposed in the literature, which are mentioned in Section 2.2.2.4. In this study, the pareto stratum – niche cubicle (PS-NC) approach, which is proposed by Hyun et. al (1998), is used to determine the selection probability for each individual. This method is a combination of pareto ranking and fitness sharing schemes. The pareto ranking scheme focuses on improving the solution quality, hence it emphasizes the exploitation issue. The fitness of each string is determined by a non-dominated sorting procedure. This, however, has no real control over the solution diversity. Unlike Pareto ranking scheme, the fitness sharing scheme – niche-cubicle – emphasizes the exploration issue. This strategy introduces a niching concept which can improve the diversity of solutions in a population, however, it has limitations in finding good solutions. Therefore, these two methods are combined to round off the weakness of each other, and to attempt to simultaneously meet the two issues of exploration and exploitation.

This approach integrating two schemes, associates every individual with a rank which is determined by the sparseness of individuals and Pareto optimality. In the following paragraphs, the concepts of niche cubicle and Pareto strata are explained and then the overall selection procedure is given.

Niche-Cubicles. Niche cubicles are first constructed for every individual of a generation. A niche cubicle is a rectangular region whose center is the individual. The size of the niche cubicle is computed using Eq.19. Suppose a problem is solved under n objectives. Let MAX_{lt} and MIN_{lt} be the maximum and the minimum of the lth objective function at generation t, respectively. Then, σ_{lt}, the niche size for the lth objective is computed as follows:

$$\sigma_{lt} = \frac{MAX_{lt} - MIN_{lt}}{\sqrt[n]{pop.size}}, \quad l = 1,2,...,n \qquad (19)$$

where *pop.size* is the size of the population. The niche size is calculated at every generation. Figure 5.4 illustrates the construction of niche cubicles when $n = 2$ and $pop.size = 25$. Two niche cubicles are shown for arbitrarily chosen individuals, p_1 and p_2. Since the size of every niche cubicle is same, the solution density of a niche cubicle can be simply measured by the number of individuals included in the cubicle. A solution located in a less dense cubicle is allowed to have a higher probability to survive in the next generation. For example, since the niche cubicle created for p_1 is less dense than that for p_2, p_1 will have higher survival probability than p_2 if both of these individuals are included in the same Pareto stratum, which is described below.

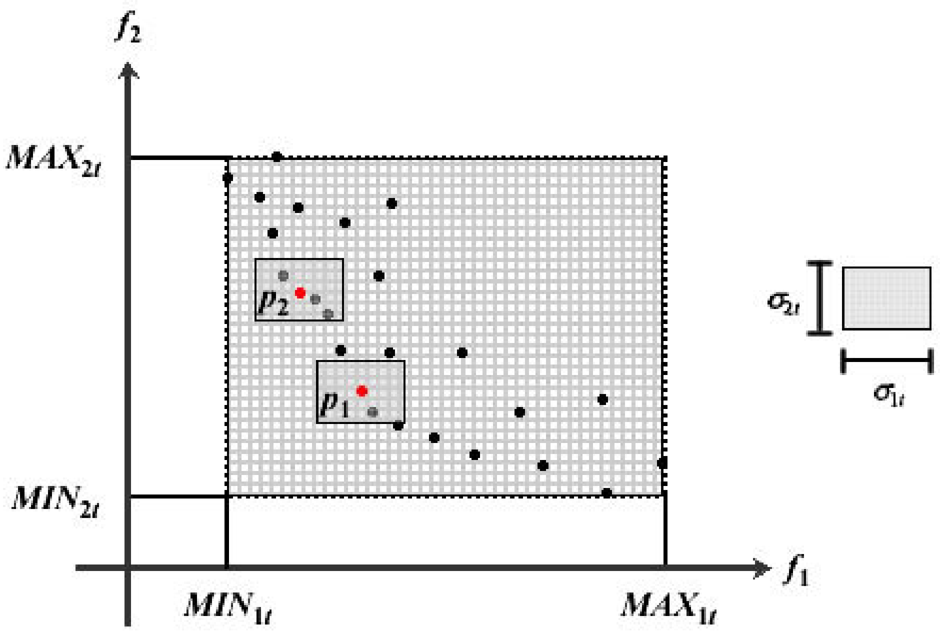


Figure 5.4. Niche cubicles

Pareto Strata. Next, Pareto strata are identified. Given a population of solutions, some of them are non-dominated solutions. Such solutions form a Pareto stratum as illustrated in Figure 5.5, where two objectives, f_1 and f_2, are minimized simultaneously. Removing the stratum from the population uncovers the next Pareto stratum. This can be repeated until all the solutions are used up. A solution which is contained in a stratum found earlier, is allowed to have a higher probability to survive in the next generation.

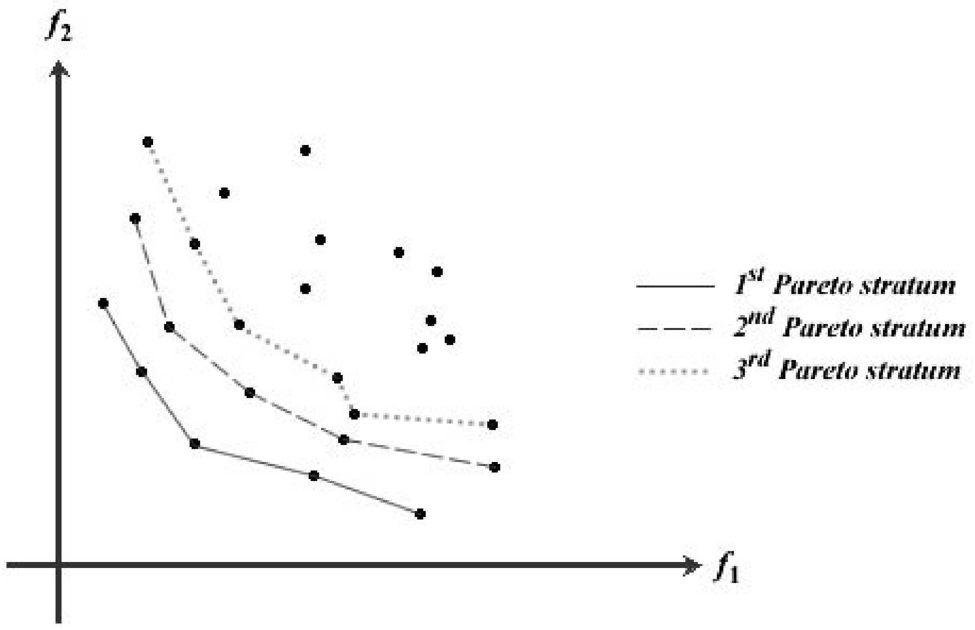


Figure 5.5. Pareto strata

The overall selection procedure combining the Pareto stratum and the niche cubicle is provided below. $P(t) = \{p_1, p_2,\ldots, p_{pop.size}\}$ denotes the current population and p_v is the individual whose rank is v.

Step (0) $v = 0$; $u = 1$ and $P_u = P(t)$

Step (1) For each individual in $P(t)$; construct its niche cubicle and calculate the solution density of this cubicle.

Step (2) Find the uth Pareto stratum, $\mathrm{PS}_u = \{(p_i, b_i) \mid p_i$ is a non-dominated solution to P_u, b_i is the solution density of the niche cubicle associated with p_i $\}$.

Step (3) Sort the individuals of PS_u in the increasing order of b_i's, and assign the rank $(v + s)$ to the sth individual, for s = 1, 2,…, $\mid \mathrm{PS}_u \mid$.

Step (4) $v = v + \mid \mathrm{PS}_u \mid$; $u = u + 1$ and $P_u = P_{u-1} - \mathrm{PS}_{u-1}$. If $P_u = \varnothing$, go to step 3.2; otherwise, go to step 3.5.

Step (5) For each individual, determine the probability of survival using a rank-based selection scheme wherein the following geometric distribution is used.

$$\text{Prob}[p_v] = q(1-q)^{v-1}, v = 1,2,3,..., pop.size \tag{20}$$

where q is the selection parameter ($0 < q < 1$).

Step (6) For each individual p_v, calculate the cumulative probability of survival, $\text{CProb}[p_v]$, using Eq. 21.

$$\text{CProb}[p_v] = \sum_{i=1}^{v} \text{Prob}[p_i] \tag{21}$$

For each parent selection process, using the roulette wheel selection mechanism a random number between 0 and 1, *r,* is generated and the individual p_v, which satisfies the following condition, is selected as a parent:

$$\text{CProb } [p_{v-1}] < r \leq \text{CProb } [p_v] \tag{22}$$

This selection process is repeated as many times as the size of the population. In this selection, the primary concern is given to Pareto optimality. Individuals dominated by an individual *p* can never be more highly ranked than *p*. Solution density is considered secondarily for individuals in the same Pareto stratum. This contributes to maintaining a distribution of diverse solutions in a population.

5.1.4 Crossover and Mutation

The crossover and mutation operators that are used in the solution of multi-objective MMALSP are the same as the ones used for the single-objective GA presented in Chapter 4, namely order crossover (OX) and inversion (INV) operators. While the probability of crossover has been fixed at 0.8 during the whole run, mutation probability starts with an initial value of 0.2 and varies periodically according to the feedback coming from the algorithm. This variation process is handled by the AGA controller which will be explained later.

5.1.5 Comparison of Domination and Elitist Strategy

To determine which individuals from the previous generation will survive to the next generation, we propose an adaptive elitist strategy. The number of elites to be selected is determined after a domination check between the two populations. Both the parent and offspring populations are subject to a cross-population check, where non-dominated individuals of the parent population are identified. These elite individuals are then transferred directly to the new generation. The remaining population quota is completed with the offspring individuals, starting from the one with the highest rank.

5.1.6 Performance Evaluation and Adaptation

This is where the adaptive GA features come into play. Evaluation and adaptation are controlled by a multi-objective AGA controller, which is a slightly modified version of the single-objective one presented previously in Chapter 4.

As it was the case in the single objective AGA controller, the multi-objective AGA controller uses the convergence measure to evaluate the progress made in best objective function values during a number of generations. Since there are three separate objective functions in this multi-objective environment, an overall convergence measure is calculated using the following equation:

$$CM = \prod_{i=1}^{3} (f_i^b / f_i^0) \tag{23}$$

where f_i^b is the *i*-th objective function value of the current best element (for the *i*-th objective function) found so far and f_i^0 is the *i*-th objective function value of the best element found before the last *G* generations. Once again, the use of an elitist strategy limits *CM* to the range of [0, 1].

The set of linguistic labels for *CM* is {*Low*, *High*}. However, as the overall *CM* is calculated by multiplication of three *CM* values, the same level of progress may result in a lower CM value in a multi-objective environment than a single objective one. In order to circumvent this effect, the values of linguistic labels are toned down. The *Low* label now corresponds to a *CM* with a value of 0.7 or lower, whereas the *High* label still corresponds to a *CM* with a value of 0.95 or greater. If the *CM* value is in the (0.7, 0.95) range, then both labels are possible. In such a case, label is determined using a random number (*r*) as follows:

$$CM \text{ label} = \begin{Bmatrix} low, & r \le \dfrac{0.95 - CM}{0.95 - 0.7} \\ high, & otherwise \end{Bmatrix} \tag{24}$$

Numerical representations of the linguistic *CM* labels are depicted in Figure 5.7a.

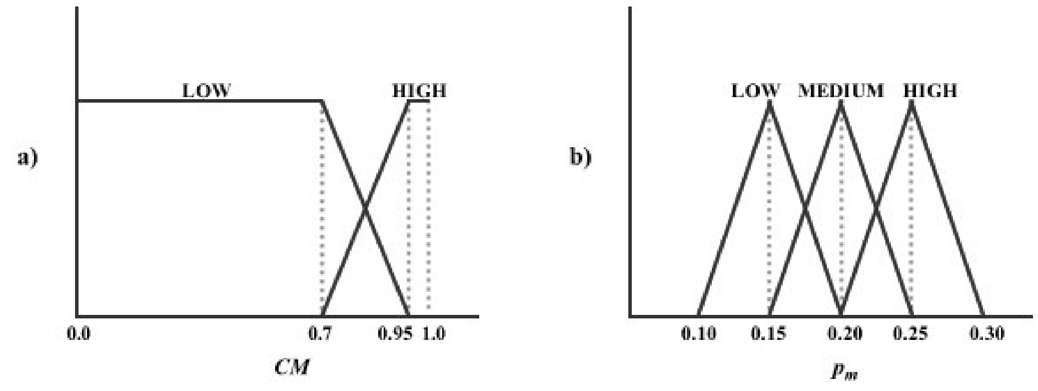


Figure 5.7 Meanings of the linguistic terms

The mutation probability is still used as the GA control parameter. The set of linguistic labels associated with mutation probability is {*Low*, *Medium*, *High*}. Each of these labels corresponds to a triangular distribution as depicted in Figure 5.7b.

The proposed multi-objective AGA controller uses the same set of rules to evaluate the progress and determine the new value for the probability of mutation (see Table 4.1).

5.1.7 Termination Check

Once the maximum number of generations is reached, the algorithm terminates and displays the individuals that are in the first Pareto stratum of the latest population. These solutions are the decision alternatives for the problem at hand. If check fails, the algorithm proceeds to the next generation.

5.2 Computational Experiments and Analysis

In this section, we describe the method for evaluating the performance of the GA based solution approach proposed in section 5.1. First, the test problems and the algorithms compared are presented in Section 5.2.1, then the GA parameters are given in Section 5.2.2 and performance measures are introduced in Section 5.2.3. Finally, the test results and the insights gained through these experimental studies are given in Section 5.2.4.

5.2.1 Problem Sets and the Algorithms Compared

In order to investigate the effectiveness of the proposed GA approach in solving the multi-objective MMAL sequencing problem, various experimental studies have been carried out using the same set of 60 problems given in the previous chapter (see Table 4.3).

The data for the first four problem sets are taken from Ponnambalam et al. (2003) in which the generated problems were solved under the objective of minimizing the variation of part consumption rates. In this study, in addition to minimizing the variation of part consumption rates, we considered two other objective functions, which are namely minimizing the total setup cost and minimizing the total utility work. These objective functions take into consideration sequence dependent setup costs, model assembly times and workstation lengths. It should be noted that in order to create a comparable problem set, we added all these features to all six problem sets given in Table 4.3.

Using the classification schemes proposed in Chapter 3, we can define all these problems as follows:

$$[\, setup \mid \mid wo, cpu, setup^c \mid mlev^4].$$

The characteristics and the search capability of the proposed adaptive algorithm are compared to those of the standard PS-NC genetic algorithm, which is similar to the one proposed in Ponnambalam et al. (2003). Moreover, to distinguish between the effects of using heuristics to create initial population and using adaptive algorithm, several variations of the standard and adaptive algorithms have been created. These algorithms along with their main differences are listed in Table 5.1. Each algorithm has been run 10 times for each problem to get reliable results.

5.2.2 Parameter Setting

The values of genetic parameters have been determined through preliminary experiments. For all problems, the population size has been set to 100. The initial rates for crossover and mutation are set to 0.8 and 0.2 respectively. While the crossover rate is fixed throughout the generations, the mutation rate varies in the range of $[0.10,\ 0.30]$. For the non-adaptive GA algorithm, the number of elites is set to 20. The selection parameter, q, is fixed to $3/popsize$ (see Eq. 20), adaptive controller is called once in every 10 generations and finally, for the termination of the algorithm, the maximum number of generations is set to 250.

5.2.3 Performance Measures

This section explains the performance metrics employed to compare the strengths and weaknesses of standard and adaptive genetic algorithm approaches. These performance metrics examine the fraction of first pareto stratum individuals in a population, the non-dominated individuals in the Pareto front, the ratio of non-dominated individuals as well as the convergence performance for each objective function. In the data collection process, all these metrics are averaged on the number of replications made.

Table 5.1 Algorithms Compared

Algorithm	Definition	Initial Population Heuristics	Mutation Rate	Number of Elites
Std	[| *multi, pr, fsh*| | *rank, elit* | *ox, inv*| |]	Random	Fixed to 0.2	20
Std + Both*	[| *multi, pr, fsh*| *hrstc* | *rank, elit* | *ox, inv*| |]	Random + IPH1 + IPH2		
Std + IPH1		Random + IPH1		
Std + IPH2		Random + IPH2		
Adp	[| *multi, pr, fsh*| | *rank, elit* | *ox, inv*| *adprm*|]	Random	Varies in the range of [0.10, 0.30]	Varies
Adp + Both*	[| *multi, pr, fsh*| *hrstc* | *rank, elit* | *ox, inv*| *adprm*|]	Random + IPH1 + IPH2		
Adp + IPH1		Random + IPH1		
Adp + IPH2		Random + IPH2		

**Includes both of the two initial population heuristics, IPH-1 and IPH-2.*

5.2.3.1 Number of 1st Pareto Stratum Individuals (NPS1)

This performance metric simply shows the number of individuals in the first Pareto stratum of the population. A higher number means more alternatives are available for decision makers.

5.2.3.2 Number of Non-dominated Individuals (NNI)

When a population is compared to another, some of its first Pareto stratum individuals might be dominated by those of the other population. The number of the remaining non-dominated individuals is denoted by *NNI*. The higher the value of *NNI*, the better the solution quality, and hence the better the algorithm performance is.

5.2.3.3 Ratio of Non-dominated Individuals (RNI)

The first two measures, *NPS1* and *NNI*, are absolute values which may possibly lead to misjudgments about the performance of the algorithm. Hence we define a relative measure, *RNI*, which is the ratio of number of non-dominated individuals (*NNI*) to number of first Pareto stratum individuals (*NPS1*):

$$RNI = \frac{NNI}{NPS1} \tag{25}$$

Once again, the higher the value of *RNI*, the better the solution quality, and hence the better the performance of the algorithm is.

5.2.3.4 Convergence

Convergence pertains to the speed at which the algorithm approaches to the optimal or near-optimal solution. Since we use a multi-objective approach, convergence values are calculated for each objective function separately. For the sake of simplification, actual values are normalized using the following equation:

$$b'_g = \frac{b_g - B_{\min}}{B_{\max} - B_{\min}}, \quad g = 1,2,...250 \tag{26}$$

where b'_g is the normalized best value obtained in generation *g*, b_g is the actual best value obtained in generation *g*, $B_{\min}$ is the minimum best value and $B_{\max}$ is the maximum best value obtained during the whole experiments. It follows from Eq. 26 that b'_g is in the range of [0, 1], i.e. $0 \le b'_g \le 1$. For each objective function, b_g values are calculated for the last generation to compare the convergence performance of algorithms.

5.2.4 Experimental Results

In this section, the results of the experiments are presented and analyzed in terms of previously declared performance measures. The performance of the algorithms is compared for each of the six problem sets. The results are presented as averages of 10 problems in each problem set.

Figures 5.1 to 5.3 depict the number of first pareto stratum individuals (*NPS1*), non-dominated individuals (*NNI*) and the ratio of non-dominated individuals (*RNI*), respectively. In each figure, it is possible to see the performance of each algorithm for every problem set.

Based on Figure 5.1, we can state that all of the algorithms tend to provide more alternative solutions as the problem size expands. Moreover, all of the adaptive algorithms provide larger number of solutions than the standard algorithms. Another finding is that, the heuristics do not seem to improve the performance of the algorithms except for the standard ones with IPH2 heuristics (Std+Both and Std+IPH2) in problem set 6.

Figure 5.2 shows the average number of non-dominated individuals for each problem set. It is obvious from this figure that the adaptive algorithms outperform all the standard algorithms. The number of non-dominated individuals provided by the adaptive algorithms starts from 55 and tends to increase as the problem size expands, whereas the performance of the standard algorithms seems to be stabilized around 20. The heuristics do not seem to affect the performance of none of the algorithms.

In Figure 5.3, the ratio of non-dominated individuals can be seen for each problem set. For the standard algorithms, the number of non-dominated individuals does not change while the total number of alternative solutions increases (see Figure 5.1 and Figure 5.2). Thus, the ratio of non-dominated solutions decreases as the problem size expands. Unlike standard algorithms, for the adaptive algorithms, both the number of non-dominated individuals and the total number of alternative solutions increase (see Figure 5.1 and Figure 5.2). This results in a relatively stagnant ratio of non-dominated solutions throughout all the problem sets.

Figure 5.3 also shows that employing IPH2 heuristic (in Std+Both, Std+IPH2, Adp+Both and Adp+IPH2) usually causes the algorithms to have lower RNI values. This effect is more significant for standard algorithms.

In summary, from the inspection of Figures 5.1 to 5.3, it can be stated that the initial population heuristics, IPH1 and IPH2, do not have any significant effect on the domination performance of algorithms. It can also be stated that adaptive algorithms provide higher number of alternatives in all cases and unlike the standard ones, their *RNI* performance do not decrease as the problem size expands.

Figures 5.4 to 5.6 show the relations between the three objective functions. The graphic data has been taken from the first pareto stratum of a randomly selected replication for problem 60th. These individuals are then plotted in three diagrams showing the relations between VPC and SC, VPC and UW, and finally SC and UW values. From inspection of these figures, it's clear that while VPC and SC objectives conflict with each other, UW objective function does not interfere with the other two.

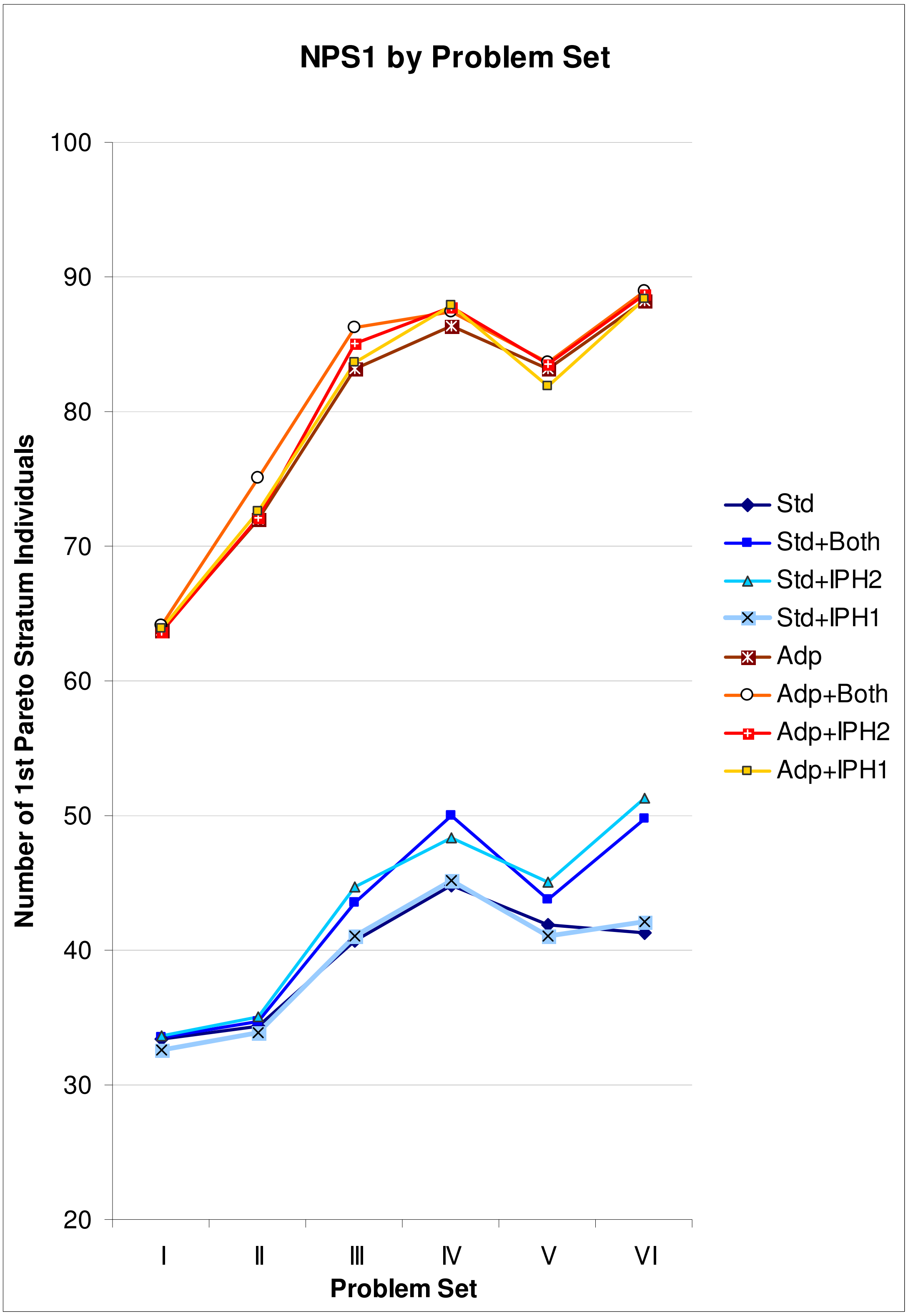


Figure 5.1 Number of alternative solutions

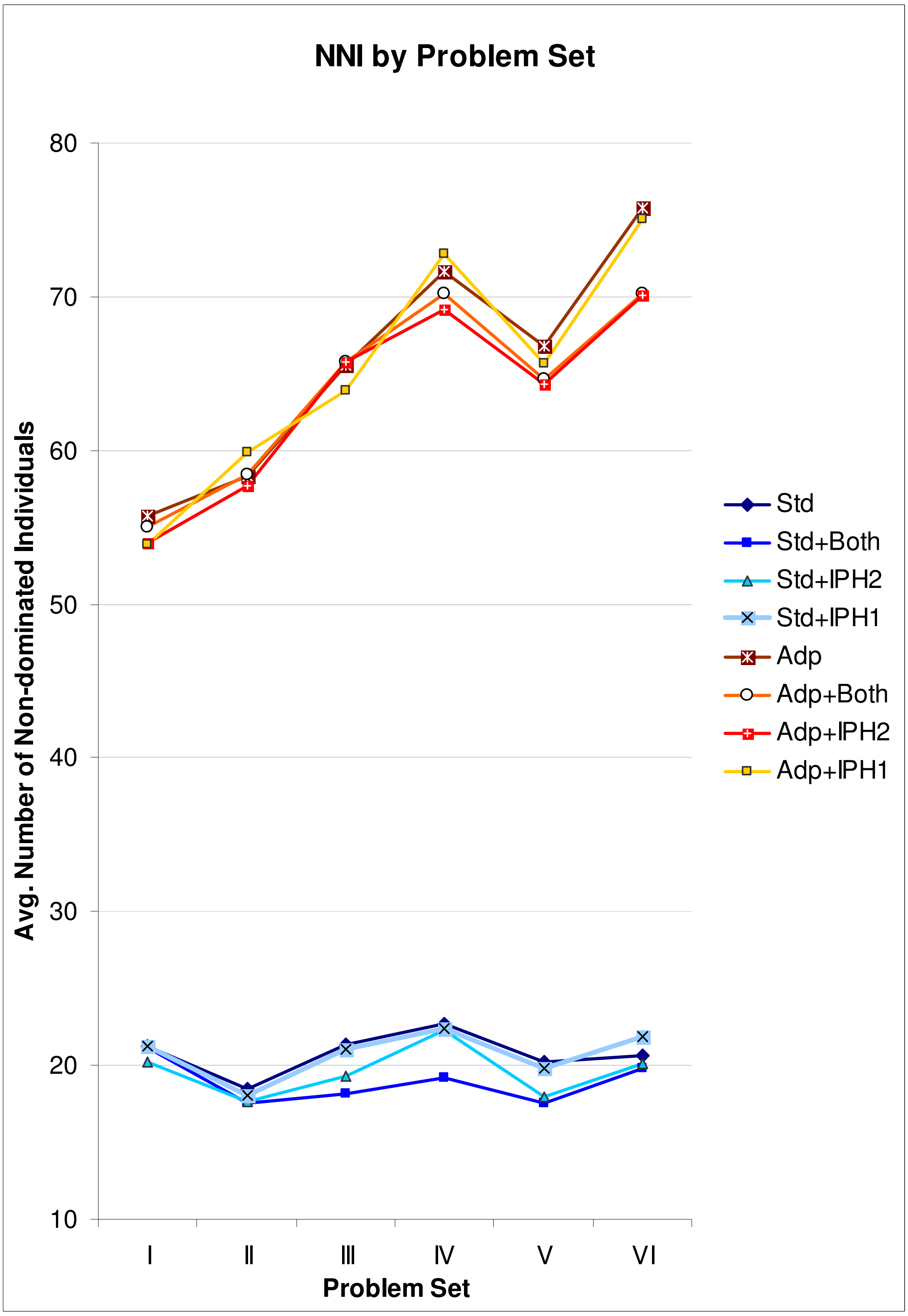


Figure 5.2 Number of non-dominated solutions

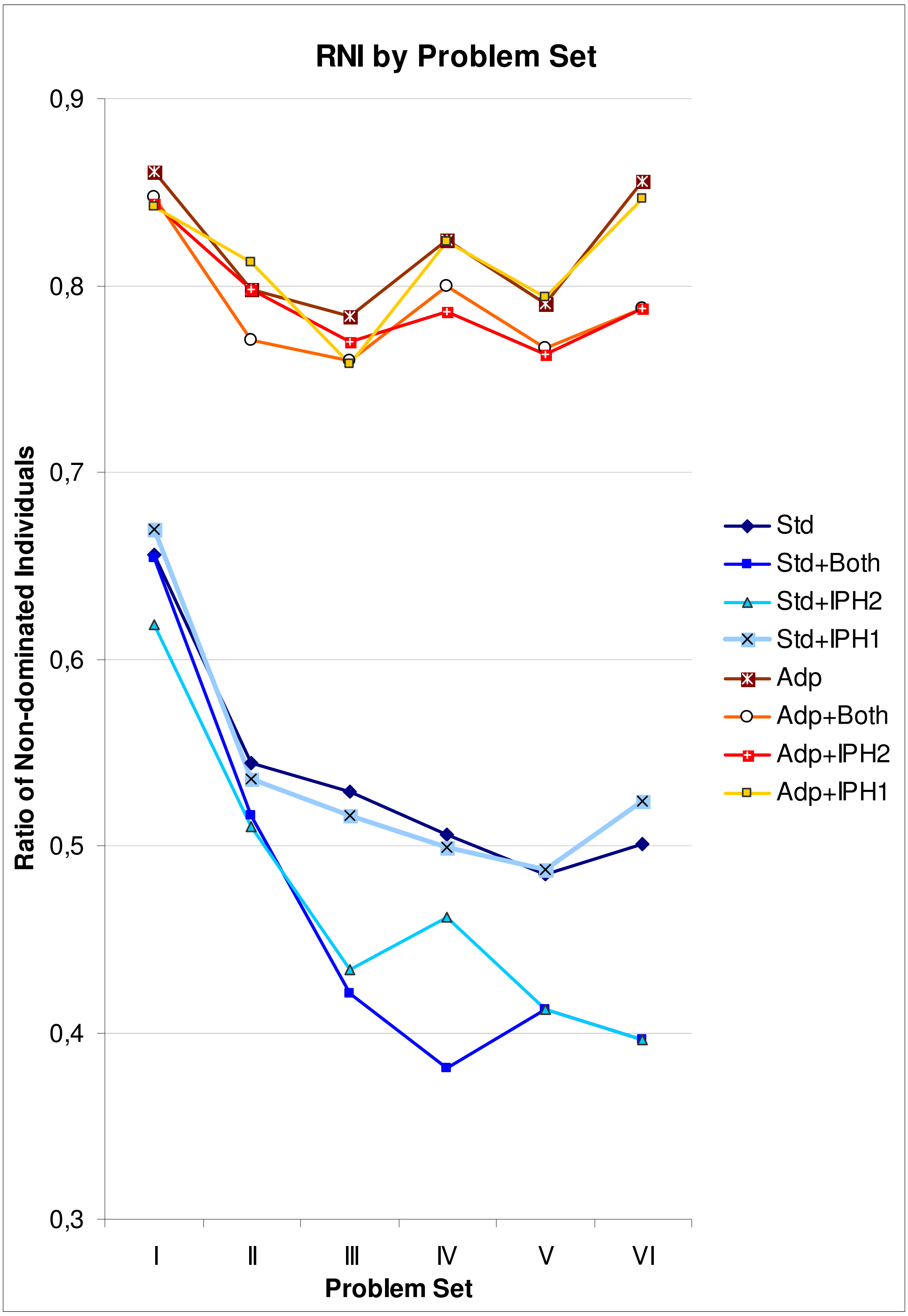


Figure 5.3 Ratio of non-dominated solutions

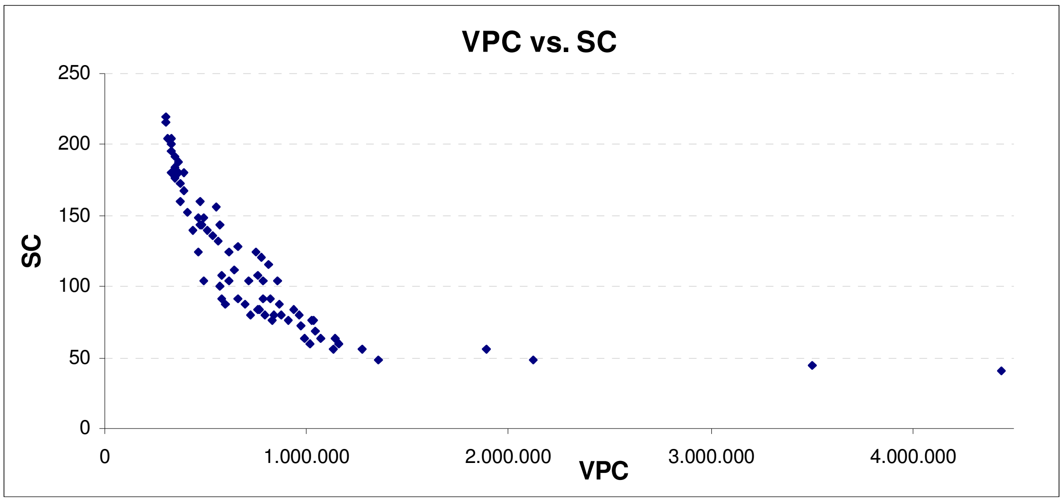


Figure 5.4 VPC – SC relation

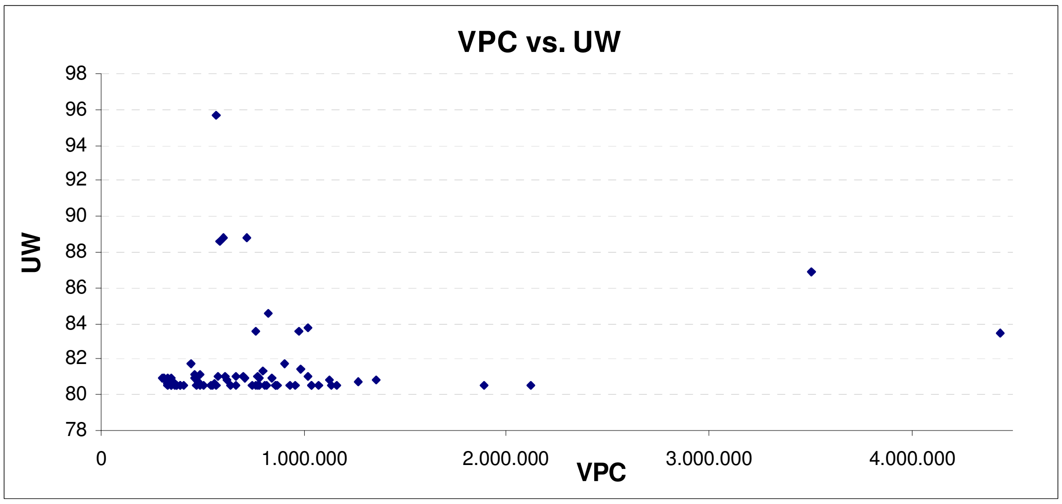


Figure 5.5 VPC – UW relation

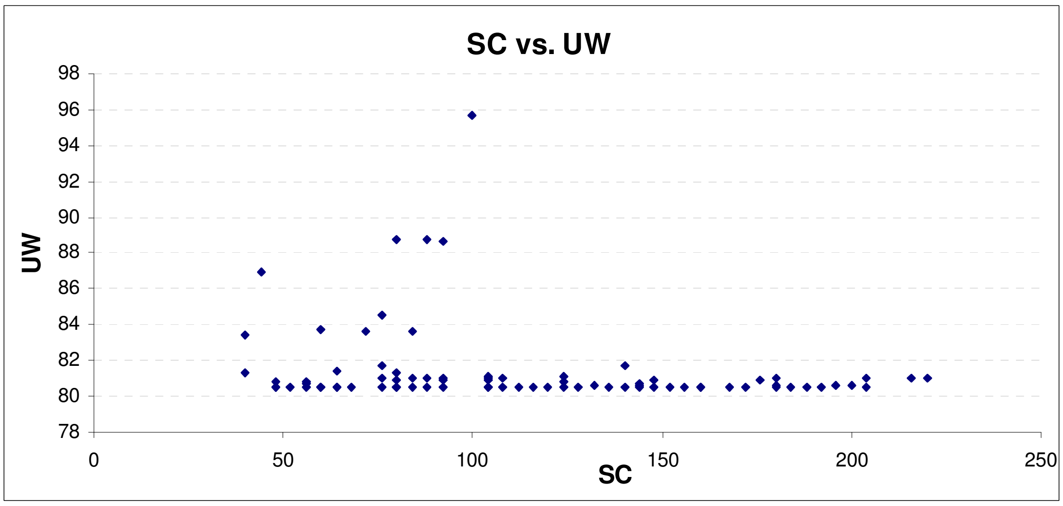


Figure 5.6 SC – UW relation

Moreover, we compared the performance of standard and adaptive algorithms with respect to their convergence rate (see Figures 5.7 and 5.8). In both of the figures, the deviation from the best value given in y-axis is obtained by using Eq. 26 based on the last generation of each run. These values are then plotted for all of the six problem sets. It should be noted that in these sets of comparative studies, the third objective function minimizing utility work is ignored since it does not interfere with the other two and generally it converges to a best value very quickly (see Figures 5.4 to 5.6).

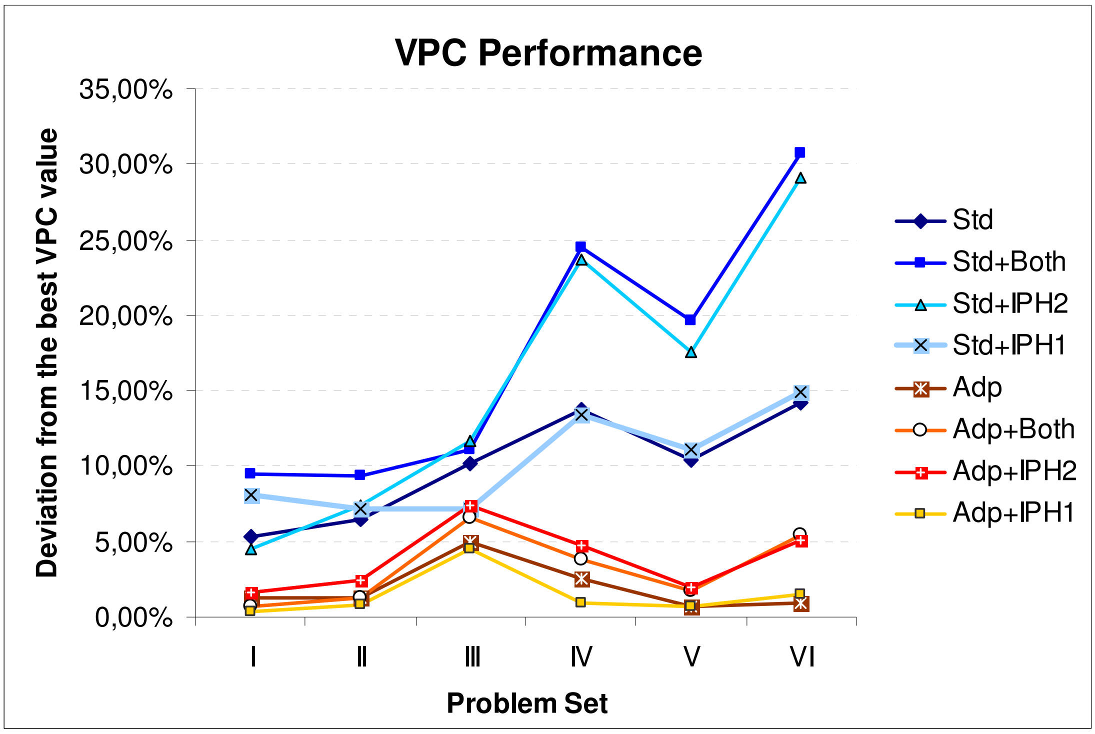


Figure 5.7 VPC convergence performance

Figure 5.7 shows that, the results of the adaptive algorithms usually stay in the 5% vicinity of the best VPC values. However, standard algorithms start with a 5% vicinity and their performance worsens as the problem size expands. It can be also seen that employing the IPH2 heuristic affects the performance of VPC negatively for both the standard and adaptive algorithms. Unlike IPH2 heuristic, the IPH1 heuristic, which is employed to have better initial populations for VPC optimization, does not seem to make any improvement on final VPC values. Another finding during these experimental studies was that for some small sized problems, the IPH1 heuristic provided the best values during the generation of initial population, actually before the start of the genetic search.

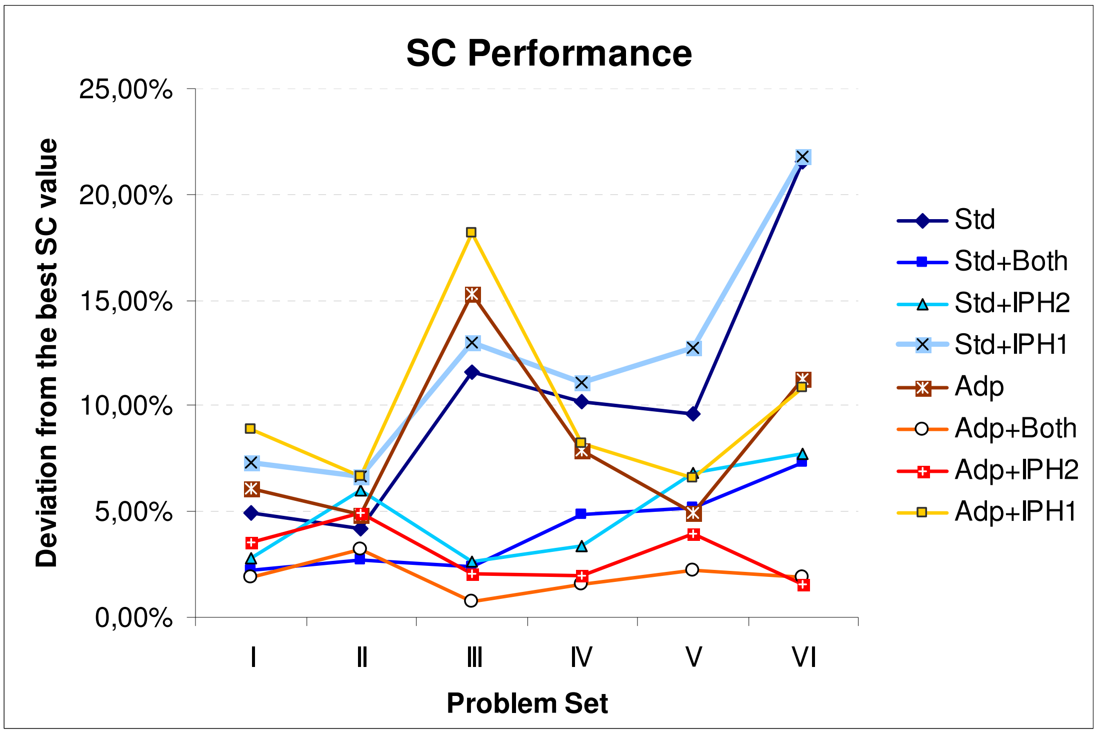


Figure 5.8 SC convergence performance

In Figure 5.8, it can be seen that the adaptive algorithms outperform the standard algorithms. Also, the algorithms which employ the IPH2 heuristics provide better SC values. Although the standard algorithms which employ IPH2 heuristic (Std+Both, Std+IPH2) provide better SC values than the adaptive algorithms that do not employ it (Adp and Adp+IPH1), they are outperformed by their adaptive counterparts, Adp+Both and Adp+IPH2. An interesting outcome of this analysis is that the algorithms which employ only IPH1 heuristic usually provide worse results than the algorithms without any heuristics, whereas IPH1 heuristic improves the results when employed together with the IPH2 heuristic. Figure 5.8 also shows that the performance of the standard algorithms tends to decrease as the problem size expands.

To sum up, the experimental results show that for every problem set, adaptive algorithm provides larger number of alternative solutions than the standard PS-NC algorithm. This makes it clear that the standard approach is less effective in exploration of the solution space. Besides the quantity of the solutions, it is also shown that the solution quality of the adaptive approach is much better when compared to the standard algorithm. This fact is reflected in both the number and ratio of non-dominated individuals (see Figure 5.2 and Figure 5.3). Lastly, the adaptive algorithm converges to better results which means it is much more effective in exploitation of the solution space.

CHAPTER SIX
CONCLUSIONS

A mixed-model assembly line is a type of a production line which is capable of producing a variety of different product models simultaneously and continuously. Such an assembly line is widely used in the manufacturing industry to achieve increased flexibility for product diversification.

The design and planning of mixed-model assembly lines involve several long- and short-term problems. Among these problems, determining the sequence of products to be produced has received considerable attention from the researchers. This problem is known as the Mixed-Model Assembly Line Sequencing Problem.

MMALSP appears when variations of the same basic product are produced on the same production line. These variations imply that the processing times on the individual stations and part requirements may differ, depending on the model to be processed. Therefore the production sequences need to be determined such that they do not cause work overloads or idle times and they allow maintaining a smooth rate of production and/or part consumption.

In MMALS literature, several objectives have been proposed to judge the efficiency of different production sequences. Particularly, for JIT systems, continual and stable part supply is an important requirement. Since this requirement can be realized when the demand rate of parts is constant over time, minimizing the variations in part consumption rates is an important objective to be considered. Among other objectives, the minimization of work overload and sequence-dependent setups have also attracted researchers' attention in the past years.

In the case of an objective that is more important than the others, MMALSP can be solved as a single-objective problem. However, as far as real world applications are concerned, there are usually more than one objective that need to be considered simultaneously. In such a case, MMALSP becomes a multi-objective problem.

In this study, we aimed at addressing the mixed-model assembly line sequencing problem, where there are multiple conflicting objectives. It is known that these type of sequencing problems fall into NP-hard class of combinatorial optimization problems. Among the proposed solution approaches in the literature, genetic algorithms have been shown to be quite successful in dealing with many manufacturing optimization problems. A genetic algorithm is a highly simplified computational model of biological evolution that can provide desirable solutions to challenging problems of a combinatorial nature, within a reasonable amount of CPU time. Hence we proposed a genetic algorithm approach to solve the multi-objective mixed-model assembly line sequencing problem.

In this context, after an extensive survey of literature on GAs developed to solve MMALS problem, we presented a classification scheme for MMALS problem and GA approaches adopted to deal with this problem. Later, the experiments have been started with the simplest form of MMALSP, i.e., the single objective problem. This part of the study aimed at developing adaptive techniques to improve the performance of the standard GA in solving single objective MMALSP. To evaluate the performance of the adaptive approach, a number of comparative experiments have been carried out. As a result it was observed that, the adaptive GA based approach outperformed the standard GA which did not implement any of the parameter control techniques. It was also noted that, the improvements achieved by the implementation of adaptive techniques were more remarkable as the size of problems increased.

Moreover, to solve the multi-objective MMALSP, we proposed an adaptive approach involving a new elitist strategy and initial population heuristics and using over 60 test problems we compared its performance to that of the standard PS-NC algorithm presented in Ponnambalam et al. (2003). The experiments showed that the adaptive GA based approach outperformed the standard PS-NC algorithm in both the solution quantity and solution quality. As a result of this comparative study, we could state that both the exploration and exploitation performances of PS-NC genetic algorithm can be improved noticeably by using adaptive techniques and a selective elitist strategy.

As a future work, several topics call for further attention. Future research may extend into three directions: (i) extending the adaptive approach by using variable population size, crossover rate, and several genetic operators; (ii) implementing self-adaptive parameter control techniques, in which the GA parameters are encoded into the chromosomes and are optimized simultaneously with the objective functions; and (iii) tackling the aggregated problem of assembly line balancing and sequencing using multi-objective approaches.

REFERENCES

Ashlock, D. (2005). *Evolutionary computation for modeling and optimization*, NY: Springer.

Augusto, O. B., Rabeau, S., Dépincé, Ph., & Bennis, F. (2006). Multi-objective genetic algorithms: A way to improve the convergence rate. *Engineering Applications of Artificial Intelligence*, 19(5), 501-510.

Bard, J. F., Dar-El, E., & Shtub, A. (1992). An analytic framework for sequencing mixed model assembly lines. *International Journal of Production Research*, 30, 35-48.

Beasley, D., Bull, D. R. & Martin, R. R. (1993). An overview of genetic algorithms – Part 1, Fundamentals, *University Computing*, 15(2), 58-69.

Beausoleil R. P. (2006). "MOSS" multiobjective scatter search applied to non-linear multiple criteria optimization *European Journal of Operational Research*, 169 (2), 426-449.

Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168, 694-715.

Bingul, Z., Sekmen, A., & Zein-Sabatto, S. (2000). Evolutionary approach to multi-objective problems using adaptive genetic algorithms. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, October 8–11, Nashville, TN, 2000.

Boysen, N., Fliedner, M., & Scholl, A. (2007a). Sequencing mixed-model assembly lines: Survey, classification and model critique. *Jena Research Papers in Business and Economics*, (JBE) 2/2007.

Boysen, N., Fliedner, M., & Scholl, A. (2007b). Sequencing mixed-model assembly lines to minimize part inventory cost, *OR Spectrum*, 2007.

Celano, G., Costa, A., Fichera, S., & Perrone, G. (2004). Human factor policy testing in sequencing of manual mixed model assembly lines. *Computers & Operations Research*, 31, 39–59.

Chang P.-C., Hsieh J.-C., & Wang C.-Y. (2007). Adaptive multi-objective genetic algorithms for scheduling of drilling operation in printed circuit board industry, *Applied Soft Computing Journal*, 7 (3), 800-806.

Cho, H., Paik, C., Yoon, H., & Kim, H. (2005). A robust design of simulated annealing approach for mixed-model sequencing. *Computers & Industrial Engineering*. 48(4), 753-764

Dar-El, E. M., & Cother, R. F. (1975). Assembly line sequencing for model mix. *International Journal of Production Research*, 13 (5), 463-477.

Deb, K. (1999). Evolutionary algorithms for multi-criterion optimization in engineering design. In Miettinen, K. et al. (Eds.) *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming, and Industrial Applications*, NY: Wiley.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2), 182-197.

Ding F.Y., Zhu J., & Sun H. (2006). Comparing two weighted approaches for sequencing mixed-model assembly lines with multiple objectives. *International Journal of Production Economics*, 102 (1), 108-131.

Drexl, A., & Kimms, A. (2001). Sequencing JIT mixed-model assembly lines under station-load and part-usage constraints. *Management Science*, 47, 489-491.

Drexl, A., Kimms, A., & Matthießen, L. (2006). Algorithms for the car sequencing and the level scheduling problem. *Journal of Scheduling*, 9, 153–176.

Eiben, A.E., Hinterding, R., & Michalewicz, Z. (1999). Parameter control in evolutionary algorithms, *IEEE Transactions on Evolutionary Computing*, 3(2), 124–141.

Färber, G., & Coves, A. (2005). Overview on sequencing in mixed model flowshop production line with static and dynamic context. *Reports de recerca de l'Institut d'Organització i Control de Sistemes Industrials*, Nº. 7.

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: formulation, discussion, and generalization. In: *The Proceedings of the 5th International Conference on Genetic Algorithms*, 416-423.

Gagné, C., Gravel, M., & Price, W. L. (2006). Solving real car sequencing problems with ant colony optimization. *European Journal of Operational Research*, 174, 1427–1448.

Haupt, R. L., & Haupt, S. E. (2004). *Practical genetic algorithms* (2nd ed.), NJ: John Wiley. &. Sons.

Herrera, F., & Lozano, M. (2003). Fuzzy adaptive genetic algorithms: design, taxonomy, and future directions, *Soft Computing*, 7, 545–562.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Horn, J., Nafpliotis, N., & Goldberg, D. E. (1994). A niched Pareto genetic algorithm for multiobjective optimization. In: *The First IEEE Conference on Evolutionary Computation*, 82–87.

Huang, Y. P., Chang, Y. T., & Sandnes, F.E. (2006). Using fuzzy adaptive genetic algorithm for function optimization, *Fuzzy Information Processing Society*, 484-489.

Hyun, C. J., Kim, Y. K., & Kim, Y. (1998). A genetic algorithm for multiple objective sequencing problems in mixed model assembly lines. *Computers and Operations Research*, 25, 675–690.

Jaszkiewicz, A. (1998). Genetic local search for multiple objective combinatorial optimization. *Technical report RA-014/98*, Institute of Computing Science, Poznan University of Technology.

Kara, Y., Özcan, U., & Peker, A. (2007a). Balancing and sequencing mixed-model just-in-time U-lines with multiple objectives. *Applied Mathematics and Computation*, 184, 566-588.

Kara, Y., Özcan, U., & Peker, A. (2007b). An approach for balancing and sequencing mixed-model JIT U-lines, *The International Journal of Advanced Manufacturing Technology*, 32 (11-12), 1218-1231.

Kim, Y. K., Hyun, C. J., & Kim, Y. (1996). Sequencing in mixed model assembly lines: A genetic algorithm approach. *Computers & Operations Research*, 23, 1131–1145.

Kim, Y. K., Kim, J. Y., & Kim, Y. (2000). A coevolutionary algorithm for balancing and sequencing in mixed model assembly lines. *Applied Intelligence*, 13, 247-258.

Kim, Y. K., Kim, J. Y., & Kim, Y. (2006). An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-lines. *European Journal of Operational Research*, 168, 838–852.

Knowles, J., & Corne, D. (1999). The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimisation. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, 98-105.

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety*, 91 (9), 992-1007.

Liu, Z., Zhou, J., & Lai, S. (2003) New adaptive genetic algorithm based on ranking. In: *Proceedings of the Second Internationd Conference on Machine Learning and Cybernetics*, 1841- 1844.

Mane, A., Nahavandi, S., & Zhang, J. (2002). Sequencing production on an assembly line using goal chasing and user defined algorithm, In Yücesan, E., Chen, C.-H., Snowdom, J.L., & Charnes, J.M. (Eds.) *Proceedings of the 2002 Winter Simulation Conference*, 1269 – 1273.

Mansouri, S.A. (2005). A multi-objective genetic algorithm for mixed-model sequencing on JIT assembly lines. *European Journal of Operational Research*, 167, 696–716.

Marler, R.T., & Arora, J.S. (2004). Survey of multi-objective optimization methods for engineering, Structural and Multidisciplinary Optimization, 26, 369–395.

McMullen, P.R. (1998). JIT sequencing for mixed-model assembly lines with setups using Tabu search, *Production Planning & Control*, 9, 504–510.

McMullen, P.R., & Frazier, G.V. (2000). A simulated annealing approach to mixed-model sequencing with multiple objectives on a just-in-time line, *IIE Transactions*, 32, 679-686.

McMullen, P.R., Tarasewich, P., & Frazier, G.V. (2000). Using genetic algorithms to solve the multi-product JIT sequencing problem with set-ups. *International Journal of Production Research*, 38, 2653-2670.

Merengo, C., Nava, F., & Pozetti, A. (1999). Balancing and sequencing manual mixed-model assembly lines. *International Journal of Production Research*, 37, 2835–2860.

Miltenburg, J. (1989). Level schedules for mixed-model assembly lines in just-in-time production systems, *Management Science*, 35, 192–207.

Miltenburg, J. (2002). Balancing and scheduling mixed-model U-shaped production lines. *The International Journal of Flexible Manufacturing Systems*, 14, 119–151.

Miltenburg, J., & Sinnamon, G. (1989). Scheduling mixed-model multi-level just-in-time production systems. *International Journal of Production Research*, 27, 1487–1509.

Mitchell, M. (1999). *An introduction to genetic algorithms* (5th ed.), Cambridge, MA: Bradford Book, The MIT Press.

Monden, Y. (1993). *Toyota production system*, (2nd ed.), Norcross, GA: Industrial Engineering Press.

Obitko, M. (1998). *Encoding*. Retrieved October 6, 2007, from http://www.cs.unibo.it/~babaoglu/courses/cas/resources/tutorials/ga/encoding.html.

Ponnambalam, S. G., Aravindan, P., & Rao, M. S. (2003). Genetic algorithms for sequencing problems in mixed model assembly lines. *Computers & Industrial Engineering*, 45, 669–690.

Rekiek, B., De Lit, P., & Delchambre, A. (2000). Designing mixed-product assembly lines. *IEEE Transactions on Robotics and Automation*, 16 (3), 268–280.

Sarker, B. R., & Pan, H. (2001). Designing a mixed-model, open-station assembly line using mixed-integer programing. *Journal of Operational Research Society*, 52, 545–558.

Schaffer, J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In: *The First International Conference on Genetic Algorithms and their Applications*, 93–100.

Scholl, A. (1999). Balancing and sequencing of assembly lines, (2nd ed.), Heidelberg.

Scholl, A., Klein, R., & Domschke, W. (1998). Pattern based vocabulary building for effectively sequencing mixed-model assembly lines, *Journal of Heuristics*, 4, 359–381.

Shi, Y., Eberhart, R., & Chen, Y. (1999). Implementation of evolutionary fuzzy systems, *IEEE Transactions on Fuzzy Systems*, 7(2), 109-119.

Smith, J. E., & Fogarty, T. C. (1997). Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2), 81–87.

Srinivas, M., & Patnaik, L. M. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4), 17–26.

Starkweather, T., McDaniel, S., Mathias, K., Whitley, D., & Whitley, C., A (1991). Comparison of genetic sequencing operators. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, 69-76.

Tan, K.C., Lee, T.H., & Khor, E.F. (2002). Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons. *Artificial Intelligence Review*, 17, 253–290.

Tavakkoli-Moghaddam, R., & Rahimi-Vahed, A.R. (2006). Multi-criteria sequencing problem for a mixed-model assembly line in a JIT production system. *Applied Mathematics and Computation*, 181, 1471–1481.

Thomopoulos, N.T. (1967). Line balancing-sequencing for mixedmodel assembly, *Management Science*, 14 (2), 59–75.

Ventura, J.A., & Radhakrishnan, S. (2002). Sequencing mixed model assembly lines for a just-in-time production system. *Production Planning & Control*, 13, 199-210.

Wester, L., & Kilbridge, M. (1964). The assembly line model-mix sequencing problem. In: *Third international conference on Operation Research*, 247–260.

Xiaobo, Z., Zhou, Z., & Asres, A. (1999). A note on Toyota's goal of sequencing mixed models on an assembly line. *Computers & Industrial Engineering*, 36, 57-65.

Yano, C. A., & Rachamadugu, R. (1991). Sequencing to minimize work overload in assembly lines with product options. *Management Science*, 37, 572–586.

Yano, C.A., & Bolat, A. (1989). Survey, development, and application of algorithms for sequencing paced assembly lines. *Journal of Manufacturing and Operations Management*, 2, 172–198.

Yu, J., Yin, Y., & Chen, Z. (2006). Scheduling of an assembly line with a multi-objective genetic algorithm. *The international Journal of Advanced Manufacturing Technology*, 28, 551–555.

Zhao, S., Zhao, J., & Jiao, L. (2005). Adaptive genetic algorithm based approach for evolutionary design and multi-objective optimization of logic circuits. In: *Proceedings of the 2005 NASA/DoD Conference of Evolution Hardware*, 67-72.

Zitzler, E., Laumanns, M. & Thiele, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. *TIK-Report 103*, Eidgenoessisch-Technische Hochschule (ETH), Zuerich.

APPENDIX – I. Classification Schemes

a) Classification Scheme for Mixed-model Assembly Line Sequencing Problem

$\alpha \mid \beta \mid \gamma \mid \delta$

δ : Number of Production Levels $\in \{ \circ, \, mlev^{\lambda} \}$

$\circ$	: final stage only
$mlev^{\lambda}$	: multiple production levels [$\lambda = \circ$: an arbitrary number of levels, $\lambda \in \{2,3,...\}$: exactly λ production levels]

γ : Objectives $\in \{ \circ, \, cpu, \, vpr, \, setup^{\lambda}, \, len, \, tput, \, mspan, \, level, \, stop, \, idle \}$

$\circ$	: minimizing workoverload/ total utility work (wo),
cpu	: keeping a constant rate of part usage,
vpr	: minimizing variation of production rates,
$setup^{\lambda}$	: minimizing the set-up cost/time [$\lambda = t$: min. set-up times, $\lambda = c$: min. set-up costs, $\lambda = n$: min. number of setups],
len	: minimizing line length,
$tput$	: minimizing throughput time,
$mspan$	: minimizing makespan,
$level$	: leveling workloads for stations on the line,
$stop$	: minimizing the duration of line stoppages,
$idle$	: minimizing total idle time.

Remark: In the case of multi-objective optimization more than a single objective can be selected out of the set, separating by a semicolon.

β : Assembly Line Characteristics ($\beta_1, \beta_2, ..., \beta_5$)

β_1	: Number of stations $\in \{ \circ, \, n \}$ [$\beta_1 = \circ$: arbitrary number of stations, $\beta_1 = n$: number of stations is n]
β_2	: Homogeneity of stations $\in \{ \circ, div \}$ [$\beta_2 = \circ$: stations characteristics are same, $\beta_2 = div$: diverging characteristics]

$\alpha \mid \beta \mid \gamma \mid \delta$

β_3 : Line layout $\in \{ \circ,\ u \}$ [$\beta_3 = \circ$: straight line, $\beta_3 = u$: u-shaped line]

β_4 : Launching discipline $\in \{ \circ,\ vrl \}$ [$\beta_4 = \circ$: fixed rate launching, $\beta_4 = vrl$: variable rate launching]

β_5 : Return velocity $\in \{ \circ,\ fin \}$ [$\beta_5 = \circ$: infinite return speed, $\beta_5 = fin$: finite return speed]

α : Station Characteristics ($\alpha_1, \alpha_2, \ldots, \alpha_6$)

α_1 : Station boundaries $\in \{ \circ,\ open \}$ [$\alpha_1 = \circ$: closed stations, $\alpha_1 = open$: station boundaries are open]

α_2 : Reaction on work overload $\in \{ \circ,\ off,\ stop,\ var^{\lambda} \}$

$\alpha_2 = \circ$: line continues processing,

$\alpha_2 = off$: workpiece is taken off the transportation system,

$\alpha_2 = stop$: line is stopped,

$\alpha_2 = var^{\lambda}$: overloads are compensated by variable station borders [$\lambda = \circ$: early start, $\lambda = late$: late start model]

α_3 : Processing times $\in \{ \circ,\ sto \}$ [$\alpha_3 = \circ$: static and deterministic, $\alpha_3 = sto$: stochastic processing times]

α_4 : Concurrent work $\in \{ \circ,\ cc \}$ [$\alpha_4 = \circ$: concurrent work is not allowed, $\alpha_4 = cc$: concurrent work is allowed]

α_5 : Set-ups $\in \{ \circ,\ setup \}$ [$\alpha_5 = \circ$: set-ups are ignored, $\alpha_4 = setup$: set-ups are considered.]

α_6 : Parallel station $\in \{ \circ,\ par^{\lambda} \}$

$\alpha_6 = \circ$: no parallelization,

$\alpha_6 = par^{\lambda}$: parallel stations are used. [$\lambda = \circ$: parallel stations are side by side, $\lambda = chr$: chronological parellelization]

b) Classification Scheme for Genetic Algorithm Approaches to MMALSP

$\alpha \mid \beta \mid \gamma \mid \delta \mid \varepsilon \mid \zeta \mid \eta$

$\boldsymbol{\eta}$: Hybridization $\in$ { $\circ$, *hybrid*) [$\circ$: none, *hybrid* : GA and another heuristic is used together].

$\boldsymbol{\zeta}$: GA parameters ($\zeta_1, \zeta_2, \zeta_3$)

ζ_1 : Crossover and mutation probabilities $\in$ { $\circ$, *dtprm* , *adprm* , *sfprm*}

$\zeta_1 = \circ$	: parameter tuning,
$\zeta_1 =$ *dtprm*	: deterministic parameter control,
$\zeta_1 =$ *adprm*	: adaptive parameter control,
$\zeta_1 =$ *sfprm*	: self-adaptive control

ζ_2 : Population size $\in$ { $\circ$, *one* , *vpop* } [$\circ$: fixed size, *one* : only one individual, *vpop*: varying size]

ζ_3 : Termination criteria $\in$ { $\circ$, *min*, *tfix* , *conv* , *mnl*, *ifix* }

$\zeta_3 = \circ$	: fixed number of generations (*gfix*),
$\zeta_3 =$ *min*	: satisfaction of minimum criteria,
$\zeta_3 =$ *tfix*	: fixed time,
$\zeta_3 =$ *conv*	: successive iterations no longer produce better results,
$\zeta_3 =$ *mnl*	: manuel inspection,
$\zeta_3 =$ *ifix*	: fixed number of reproduced individuals.

$\boldsymbol{\varepsilon}$: Genetic operators ($\varepsilon_1, \varepsilon_2, \varepsilon_3$)

ε_1 : Crossover Operator $\in$ { $\circ$, *two* , *ucx* , *adpt* , *boltz* , *rank* , *tour* , *stdy* }

$\varepsilon_1 = \circ$	: single-point crossover,
$\varepsilon_1 =$ *two*	: two-point crossover,
$\varepsilon_1 =$ *ucx*	: uniform crossover,

$\alpha \mid \beta \mid \gamma \mid \delta \mid \varepsilon \mid \zeta \mid \eta$

$\varepsilon_1 = adpt$	: adaptive crossover,
$\varepsilon_1 = edge$	: edge-recombination,
$\varepsilon_1 = ox$	: order crossover,
$\varepsilon_1 = pmx$	: partially mapped crossover,
$\varepsilon_1 = cx$	: cycle crossover,
$\varepsilon_1 = pbx$	: position-based crossover,
$\varepsilon_1 = isrx$	: immediate successor relation crossover.

ε_2 : Mutation operator $\in \{ \circ, mmut, um, lam, swap, shift \}$

$\varepsilon_2 = \circ$	: single-point mutation,
$\varepsilon_2 = mmut$	: multiple point mutation,
$\varepsilon_2 = um$	: uniform mutation,
$\varepsilon_2 = lam$	: lamarckian mutation,
$\varepsilon_2 = swap$	: swap mutation,
$\varepsilon_2 = shift$	: shift mutation.

ε_3 : Inversion operator $\in \{ \circ, inv \}$ [$\circ$: not used, inv : inversion operator is used]

$\boldsymbol{\delta}$: Selection Strategies (δ_1, δ_2)

δ_1 : Selection method $\in \{ \circ, rou, sus, sigma, boltz, rank, tour, stdy \}$

$\delta_1 = \circ$	: roulette wheel (rou),
$\delta_1 = sus$	: stochastic universal sampling,
$\delta_1 = sigma$	: sigma scaling,
$\delta_1 = boltz$	: boltzman selection,
$\delta_1 = rank$	: rank selection,
$\delta_1 = tour$	: tournament selection,

$\alpha \mid \beta \mid \gamma \mid \delta \mid \varepsilon \mid \zeta \mid \eta$

	$\delta_1 =$ *stdy*	: steady-state selection.
δ_2	: Elitist strategy $\in \{ \circ, elit^{\lambda} \}$	
	$\delta_2 = \circ$	: elitism not used
	$\delta_2 = elit^{\lambda}$	: elitism is used [$\lambda = \circ$: using the same population (internal), $\lambda = ext$: external population]

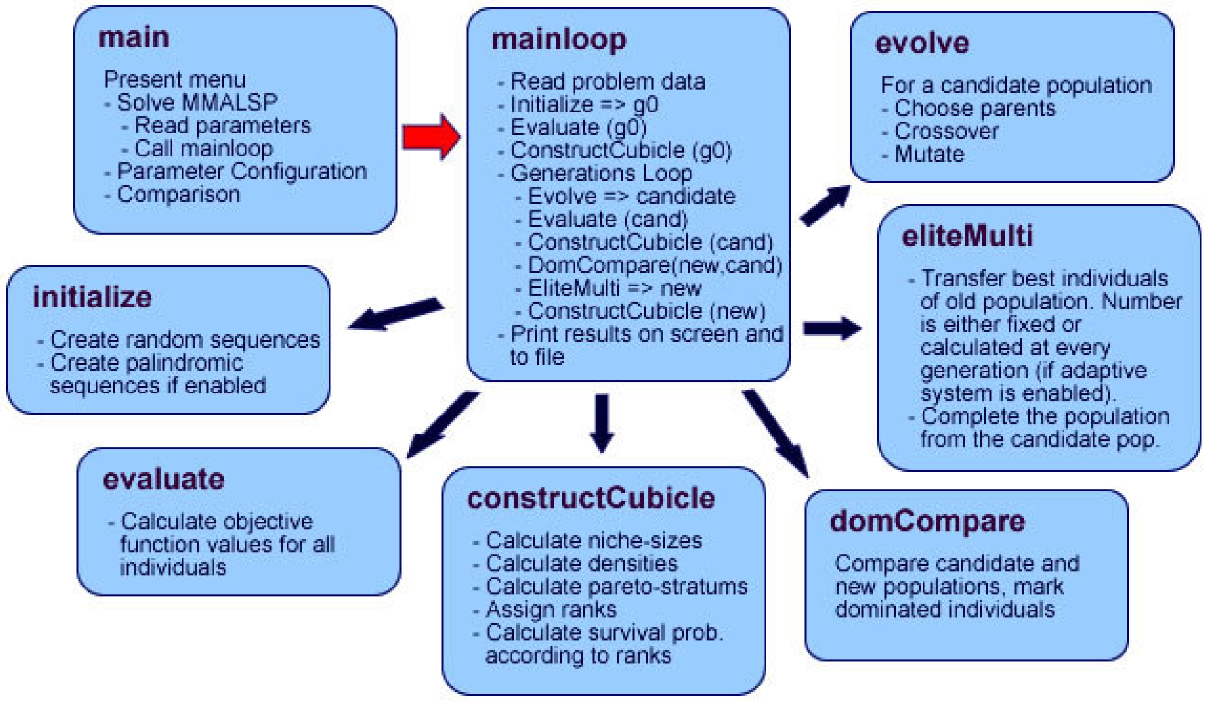
γ : Initial population $\in \{ \circ, hrstc, mem \}$ [$\circ$: randomly created, *hrstc* : heuristics, *mem* : memory-based reasoning]

β : Fitness function ($\beta_1, \beta_2, \beta_3$)

β_1	: Objective function $\in \{ \circ, multi \}$ [$\circ$: single objective, *multi* : multi-objective]	
β_2	: Fitness assignment $\in \{ \circ, wsum, alt, pr \}$	
	$\beta_2 = \circ$	: single objective function value,
	$\beta_2 =$ *wsum*	: weighted sum,
	$\beta_2 =$ *alt*	: altering objectives,
	$\beta_2 =$ *pr*	: pareto ranking.
β_3	: Diversity mechanism $\in \{ \circ, fsh, crwd, cell \}$	
	$\beta_3 = \circ$	: none,
	$\beta_3 =$ *fsh*	: fitness sharing,
	$\beta_3 =$ *crwd*	: crowding distance,
	$\beta_3 =$ *cell*	: cell-based density.

α : Genetic representation $\in \{ \circ, bin, per, tree \}$ [$\circ$: real-valued, *bin* : binary, *per* : permutation, *tree* : tree]

APPENDIX – II. General Code Structure of the Proposed Adaptive MOGA


main
Present menu
- Solve MMALSP
- Read parameters
- Call mainloop
- Parameter Configuration
- Comparison
mainloop
- Read problem data
- Initialize => g0
- Evaluate (g0)
- ConstructCubicle (g0)
- Generations Loop
- Evolve => candidate
- Evaluate (cand)
- ConstructCubicle (cand)
- DomCompare(new,cand)
- EliteMulti => new
- ConstructCubicle (new)
- Print results on screen and to file
evolve
For a candidate population
- Choose parents
- Crossover
- Mutate
eliteMulti
- Transfer best individuals of old population. Number is either fixed or calculated at every generation (if adaptive system is enabled).
- Complete the population from the candidate pop.
initialize
- Create random sequences
- Create palindromic sequences if enabled
evaluate
- Calculate objective function values for all individuals
constructCubicle
- Calculate niche-sizes
- Calculate densities
- Calculate pareto-stratums
- Assign ranks
- Calculate survival prob. according to ranks
domCompare
Compare candidate and new populations, mark dominated individuals