A Manufacturing Process Planning Model Using Genetic Algorithms

by

Bahaa Awadh

A thesis
presented to the University of Manitoba
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Mechanical/Industrial Engineering

Winnipeg, Manitoba, Canada 1994

ISBN 0-315-92287-7

Canada

Name _____

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Industrial Engineering
SUBJECT TERM

0 5 4 6
SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

A MANUFACTURING PROCESS PLANNING MODEL USING GENETIC ALGORITHMS

BY

BAHAA AWADH

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

A process planning model is developed in conjunction with a genetic algorithm. The process plans are represented as paths in multi-stage flow networks. Two binary formulations, one based on stage matrix method and the other based on stage set method, are derived to model each stage in a flow network. Both formulations allow a proper representation of alternative routes (paths) as binary strings to be evaluated by a genetic algorithm. In the stage matrix method, a 'zero' bit represents a missing connection between two nodes in the flow network, a 'one' bit denotes an existing connection between the corresponding nodes. The stage matrix representation also facilitates the employment of a modification procedure to ensure the existence of a unique path (indication of a process plan) in each individual network. The stage set method operates in a similar fashion on the binary representation of the nodes of the network. The method is shown to be more efficient than the stage matrix method with respect to computational time.

Both models are implemented in a software package. The software, which is written in the C language, first receives a flow network description of the process plans. A modification operation is then applied in order to ensure the existence of a single path per network. A pool of these single path networks is then evolved, using a genetic algorithm, in order to find the the best path, i.e. the best process plan.

The models developed in this thesis are first illustrated using simple networks. The performance of the computer programs, based on these models, is then evaluated for different sizes of networks having single or multiple objective criteria. A comparative study of the effect of increasing the number of objectives for a multi-objective network is carried out for several objectives. The stage set model is also employed to efficiently solve the constraint shortest path problem.

Acknowledgements

I would like to thank my advisors, Dr. Nariman Sepehri and Prof. Ostap Hawaleshka for their continuous support. Dr. Sepehri for his insightful academic guidance and Prof. Hawaleshka for his useful remarks and financial assistance. I also would like to thank my dissertation committee, Dr. R. Shwedyk and Dr. A. B. Thornton-Trump, for their valuable remarks during the course of this work.

Special thanks are extended to Ms. Kathy Norman of the computer services at the Uinversity of Manitoba for her continuous assistance. I also appreciate the valuable insight offered to me by Prof. R. Kocay of computer scince deptartment, in the course of conducting the complexity analysis of the algorithms presented in this thesis. Finally, I would like to extend my appreciation to Manitoba Hydro research grant committee for their generous financial support during the early years of my work.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE **PAGE**

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Motivations and General Objectives

Process planning in manufacturing systems is a dynamic and complex activity; it involves a large number of parameters that interact with each other in the course of manufacturing a product. It is defined as the process of providing a detailed description of manufacturing requirements and capabilities in order to transform a raw stock of material into a complete product. The product and the stock of material are described by an engineering drawing by means of a Computer-Aided Design (CAD) system or a solid modeler. The specifications of machines, tools, fixtures and drawings, along with the machining requirements, process capabilities, set-up times and constraints on equipment are usually described by a Computer-Aided Manufacturing (CAM) system [1].

Referring to Figure 1.1, the process planning activity provides the link between CAD and CAM systems with the aim of producing a comprehensive description of all the steps required to manufacture the product. The output from the process planning activity is then handled by an operations planner (see Figure 1.1), where the detailed cutter location files, G-codes , set-up instructions and part programs are generated and down loaded to Computer Numerical Control (CNC) machines [1].

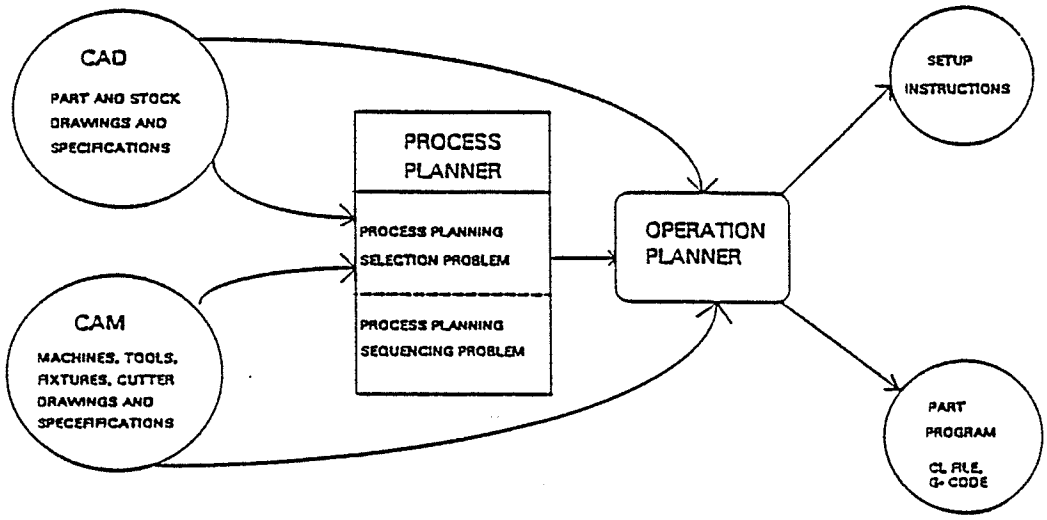


Figure 1.1: Typical manufacturing planning activity.

Computer-Aided Process Planning (CAPP) systems are being developed [2] in an attempt to overcome some of the problems that occur in manual process planning, e.g.: long turn around times, inconsistent routings and tooling, non-uniqueness in cost and labor requirements. These problems are exacerbated by a noted scarcity of skilled process planners. CAPP in modern manufacturing systems are of two types [2]: Variant Process Planning and Generative Process Planning. Variant Process Planning involves a retrieval of pre-developed process plans of part families of components derived from group technology concepts. In group technology, a family of components with similar manufacturing requirements are grouped together as a "composite" component. These composite components are coded by using some group technology coding scheme (e.g. OPITZ). When a new component is required to be manufactured, its code is mapped onto one of these component families and the

corresponding process plan is retrieved. This plan usually needs some modifications to accommodate the new component's manufacturing requirements.

A complete integration of CAD/CAM systems, which is essential in Flexible Manufacturing Systems (FMS), requires an automatic and generalized generation of process plans. The variant process planning approach demands a post processing plan modification operation and usually requires a huge data base of plans to be developed and maintained beforehand [2]. The Generative Process Planning (GPP) approach offers an alternative for overcoming these limitations, since no pre-stored process plans are usually retrieved. In GPP a new process plan is generated "automatically" according to some decision logic. The decision logic interacts with some automatic manufacturing feature recognition mechanism and the corresponding machining requirements are derived by using optimization techniques.

The modeling and analysis of process planning activities in manufacturing systems require a generalized model since:

- Alternative process plans exist in industrial cases. These plans represent the number of different ways in which a part can be produced, ie. a measure of the scheduling flexibility which, in turn, is an important element of manufacturing flexibility.

- In FMS, it is required to find the optimal process plan as fast as possible, e.g.: to allow for a recovery procedure from tool/machine failures [3].

- Handling machine loading, scheduling flexibility, and layout problems efficiently for a single part or multiple parts with a single or multiple objectives cases is needed.

The Generative Process Planning (GPP) approach seems to be the most promising approach as it can accommodate general and automated process planning mod-

els. This is particularly important in FMS, since flexible manufacturing systems exhibit a high degree of complexity where many alternative process plans exist, as opposed to a single process plan per component assumption in conventional manufacturing systems. Moreover, due to the requirement of a high degree of automation in flexible manufacturing systems, a generalized and automated process planning approach would seem to be most profitably applicable to these systems [3].

One of CAPP's many challenges is the numerous decisions that must be made in a logical manner regarding the selection of the best process plan from several alternatives.

The problem of finding the best alternative process plan is called the Process Planning Selection Problem (PPSP). This problem has also been called the Optimum Routing Analysis Problem [4]. Solving this problem along with the process selection problem, constitutes the crux of solving generalized process planning problems. The main component of PPSP is finding the best process plan amongst numerous alternatives given a certain criterion such as minimum cost, minimum time, maximum quality or under a collection of some or all these criteria [3]. Numerous machines, in the order of several hundreds machines and/or tens of machining centers with hundreds of thousands alternative process plans, are not uncommon features of large manufacturing systems [4]. This size of manufacturing systems demands a fast response with respect to finding the best alternative process plan under a single objective criterion; or a pool of compromise solutions (ie. process plans) for several objective criteria [4]. The implicit enumeration of all these alternatives can be formulated using flow networks. Flow network formulation has been widely used [22, 24, 56] to find the best process plan under single objective criterion where the problem is equivalent to solving the Shortest Path Problem (SPP). Algorithms such as Dijkstra's [22], Floyed's or other network techniques such as Out-Of-Kilter

[22, 24] are amongst the algorithms that handle SPP. These techniques, however, cannot be applied to solve the SPP under multi-objective criteria [26]. For such combinatorial problems, techniques based on goal programming and linear programming [8, 28] have been employed. Formulations based on dynamic programming [18, 20] have also been developed to handled multi-objective cases for small networks. No report related to large sizes of networks of multi-objective criteria was found in the literature search.

In this thesis a development and a study of a model based on Genetic Algorithms (GA's) for process planning activities is conducted. The model is able to find the optimal plan according to prescribed objective criteria. Such a model should be efficient, robust and able to readily handle single or multi-objective functions.

In the last few years research devoted to GA's and their application have significantly increased as attested by the existence of several conferences on the topic. In particular the use of GA's has gained some popularity in optimization [31, 32] and has been identified as a potential technique for use in heuristic search and combinatorial problems [33]. A genetic algorithm is a global search technique. It simultaneously evolves many points in the parameter space. By working with a population of solutions, the algorithm can in effect search many local minima and thereby increases the likelihood of finding the global minima. Genetic algorithms have recently been applied to problems such as job shop scheduling [34], quadratic assignment problems [44], transportation [40], floorplan design [51] and the traveling salesman problems [46]. In these studies, appropriate encoding schemes to represent the solution points were developed and GA operators were modified to meet the requirements of these schemes.

This thesis focuses on a process planning model that first handles the PPSP as a multi-stage decision problem of multi-objective that are mostly of conflicting and

non-commensurable nature. A model based on Genetic Algorithms (GA's) in order to find the best solution(s) to these objectives is then presented. A unique aspect of this model is the generation of a substantial set of Pareto (compromise) solutions. The set is generated as a side product of the process of finding the best compromise solution of the multi-objective process planning selection problem. Finding the best process route, in a pool of numerous alternative process plans is combinatorial in nature. A GA feasibility study to find such a process route is explored in this thesis.

The objective of this study is to introduce a bit string representation of the solution points for the class of problems under investigation. The coding that has been shown to be the optimal one is the binary coding [38]. Bit strings are simple to create and manipulate; many performance theorems have been proved for bit strings, and the set of parameters for a robust GA has been investigated for binary coding [57, 50, 45] . Consequently, many researchers have used this technique in carrying out real-world applications of GA's [39, 35]. Since the algorithm only needs to evaluate the objective function to guide its search, the model should be easily applied to multi- objective cases. One motivation of selecting GA's is that they are inherently parallel algorithms. Their computational speed can be enhanced using parallel processing techniques [52, 55, 53, 54]. This property is particularly attractive for the purpose of the investigation that is carried out in this thesis, since once the confidence in the model is established in terms of its convergence to an optimal solution, one should be able to increase the speed of computation.

1.2 Scope of the Thesis

This thesis formulates the PPSP as a multi-objective routing network by applying two novel approaches. The first is a stage matrix method of the zero/one type. The second is a set theoretic method. Both methods allow a proper represen-

tation of possible routes (process plans) in a multi-stage network. These routes are then represented as binary strings to be used by a genetic algorithm to identify the optimal path. This path is usually called the "compromise path" or the" efficient solution", for the multi-objective case [5, 6, 7, 8]. Also, a study of the constraint shortest path problem, where one objective of the two objective network of paths is considered as a bottleneck criterion, is carried out.

The binary strings, of all the above classes of problems, are initially formed in a randomly generated population of solution points which are then evaluated for their performances by a fitness function that describe the problem. These bit strings (solution points) are then manipulated by simple GA operators towards the best solution. The bit string representation of the process plans requires only one modification procedure to be incorporated with a traditional genetic algorithm to ensure the existence of a single path within an individual. This modification procedure involves the elimination or the addition of arcs (or nodes) at each stage of the network. The objective is to obtain a complete path that links the initial node and the final node through a single node per stage of the network.

Due to the stochastic nature of genetic algorithms, a rather detailed study of various parameters that are related to GA is carried out in this thesis. For example, the effects of selecting the population size, ie. the size of the solution pool samples, the number of generations, the number of iterations needed to converge to an optimal solution, the probability of crossover of two individuals etc. are discussed in this thesis.

The rest of this thesis is organized as follows: a relevant background of GA's and some solution techniques for PPSP is briefly described in Chapter 2. Chapter 3 describes the relevant formulation, a description of the zero/one stage matrix and computational complexity of the method; finally, some demonstrative results

of that method is presented and some concluding remarks regarding this method are drawn. The stage matrix investigation lays the ground for a better formulation, namely: the stage set method, that can handle PPSP more efficiently than the stage matrix method. The set theoretic method is similarly investigated in Chapter 4 with addition of the analysis of the effect of increasing the complexity of flow networks with various number of objectives. Finally, a treatment of the multi-parts and multi-processing production systems is illustrated in that chapter with an example taken from the literature; several larger examples are randomly generated and treated in that chapter. These demonstrations show a fast rate of convergence to an optimal solution. The robustness of genetic algorithms is demonstrated for various degrees of complexity, ie. various number of objectives and network sizes. This robustness should ensure a similar performance for a wide variety of complex network cases. The constraint shortest path problem is also studied in this Chapter. Finally, the thesis concludes with Chapter 5 where an outline of benefits of the approach is stated. This chapter also suggests some extensions to the models.

CHAPTER 2

Literature Review and Background

The formulation of process plans, described as flow networks, are presented in this chapter. Flow networks delineate the various alternatives that exist in most multi-stage manufacturing systems. Optimization techniques that either handle the process planning selection problem or other related problems are discussed. Elements of genetic algorithms are then described along with their applications to relevant optimization problems in manufacturing systems.

2.1 Previous Relevant Work

The generalized process planning problem is usually decomposed to its constituent problems, namely the PPSP and the sequencing problem. The PPSP is the focus of this thesis since it is considered as the preparatory stage to process planning activities. PPSP is also called "optimum process planning" or "optimum routing analysis" [4]. The overall flexibility of manufacturing systems is a function of the specific routes that the jobs take through the machines [9]. Typically, there are several alternative process plans (routes or work flow) for converting a raw stock of material into a final product. Solving PPSP efficiently demands certain properties from an optimization technique:

- It must be generalized (robust).
- It must be global in nature.

- It should be able to operate in real time to recover from machine or tool failures.
- It should handle multi-objective cases and be easily transformable to handle various decision making activities.
- It should be able to handle precise and imprecise information.

Most of the early work on process planning in manufacturing systems involved solving the machine loading problem, where the production capacity of machines is balanced to increase their utilization, see for example [10, 11, 12, 13, 14]. These studies have considered a single process plan per product and the objectives to be optimized are of the single type. Modern manufacturing systems exhibit numerous process plan alternatives for a single part or multiple parts, i.e. a family of parts.

Recent work on process planning comprise, basically, two types of process planning techniques. The first type is based on artificial intelligence techniques that are extensively surveyed in [2, 3]. The second type is based on mathematical programming techniques [3, 4]. Formulating alternative process plans in flow network fashion leads to the equivalence of finding the best path that links a chain of nodes in that network, starting at the source node and ending at the terminal node. This is also called the Shortest Path Problem (SPP) [21, 23]. A network formulation of SPP adapted from [23] is shown here:

Let (N,A) be a directed network with N nodes and A arcs. Each arc of the network has one attribute, e.g., cost. Let $c_{ij} >= 0$ be the cost of arc (i, j) where i and j are node numbers up to N nodes. The objective is to find the minimum cost path in the network. This is formulated as a minimization function [23],

$$Minimize \quad Z_{i,j} = \sum_{i} \sum_{j} c_{ij} X_{ij} \tag{2.1}$$

such that,

$$\sum_j X_{sj} - \sum_j X_{js} = -1, \tag{2.2}$$

$$\sum_j X_{rj} - \sum_j X_{jr} = 1, \tag{2.3}$$

$$\sum_j X_{ij} - \sum_j X_{ji} = 0, \quad for \ \ all \ \ i, \ i = s, \ i = r, \tag{2.4}$$

where, $X_{ij} \ \varepsilon \ (0, 1)$, and it is the flow passing through the network arcs (i, j); s and r correspond to source and sink nodes respectively.

Z_{ij} is the objective function for each path in the entire network, and

c_{ij} is the cost associated with each link in the path of each stage, j=1,2,..,N

Equation (2.1), corresponds to the SPP. The constraints, depicted by Equations (2.2) to (2.4), insure that a single arc input to a node and output from a node is evaluated at each node. Solution techniques employed to solve the single objective type of this problem efficiently are in abundance. Leading these techniques is the greedy algorithm called Dijkstra algorithm [21]. Other flow network techniques include Floyed algorithm, Out-of-Kilter [22] and many more techniques listed in [21, 22, 23, 24, 25].

The SPP takes on a different shape if it accompanied by a second objective that can be treated as a bottleneck criterion. The new problem is termed the " Constraint Shortest Path Problem" (CSPP) [23, 26].

The CSPP can be simply solved by enumerating every combination of the decision variable (Equation 3.1), ie. by generating all zero-one vectors $(x_1, x_2, ..., x_n)$, by setting each variable x_j to a value of zero or one. From among all those vectors, one then chooses the combination with the smallest value of the objective function Cx (or Equation 3.1). Obviously, this complete enumeration method suffers from a combinatorial explosion problem, i.e. it is limited to handle small problems

[23]. For example, to enumerate all solutions for 100 decision variables, x_j, requires 2^{100} computations; e.g. a computer of a microsecond per calculation capability will take 10^{12} years to enumerate all these solutions [23]. Many researchers [23, 26, 27] have employed some relaxation techniques based on Lagrangian, gradient and sub-gradient solutions techniques to reduce the amount of computations needed to find approximate solutions for CSPP. A network formulation of CSPP has been carried out by these researchers. CSPP comprises the network formulation of SPP, Equations (2.1) and (2.2) to (2.4), as well as a new constraint that represents a second objective in addition to the cost objective of the SPP, e.g. time. Therefore, each arc of a network that represents the CSPP has two attributes, cost and time. The new objective is described below [23, 26]:

$$\sum_i \sum_j t_{ij} X_{ij} <= T \tag{2.5}$$

where, X_{ij} is as described earlier, $t_{ij} >= 0$ is the time to traverse an arc, and T is the highest allowable time (adjustable time) value per path.

Equation (2.5) transforms SPP to a non-polynomial problem [23, 26, 27]. It is equivalent to the knapsack problem which is known to be NP-complete, i.e. no polynomial algorithm exists to solve it, [23, 26, 27]. Efficient (sub-optimal) solutions for the constraint shortest path problem have been found by applying a network reduction technique developed by Hassan [26].

The more demanding case for the SPP is the multi-objective case. Several researchers have dealt with this problem by using various mathematical programming techniques [3, 5, 6]. The bulk of multi-objective work in the last few decades is quite impressive [5, 6, 7, 8]. Classically, multi-objective problems comprise a vector-valued objective function as follows [5, 6, 30]:

$$y_{(x)} = (y^1_{(x)}, \ldots, y^\ell_{(x)}) \tag{2.6}$$

where $y_{(x)}$ is the ℓ-objectives function and $x \in E^n$ is a decision variable. Let $X \subseteq E^n$ be a set of all feasible solutions. The objective is to find for all points $x^\circ \in X$ which are non-dominated, i.e. there exists no other $x \in X$ such that, $y_{(x)} \leq y_{x^\circ}$, . Thus the minimization objective function is described as:

$$Minimize\ y_{(x)} \tag{2.7}$$

As far as the previous work is concerned, a multi-objective dynamic programming approach was employed by Szadkowaski [11] where alternative process plans to machine a single part were modeled by a multi-stage flow network of processes and proceeded to find the best compromise solution. Sancho [18] described a formalism that was based on thee stage flow network concept. In his method, the search space of feasible routes in the network is explored using a dynamic programming technique. A reduction of the set of efficient solutions of a multi-objective routing network (of an overall objective values) was carried out and all but one of the multidimensional objectives were incorporated in the state space [18]. Every stage, starting at the final stage, was checked recursively for an optimality condition until the initial stage was reached. The collection of the optimal decisions at each stage yield a solution to the network problem. Sniedovich [20] extended Sancho's model to include all costs in a multidimensional objective function. These methods tend to give progressively weaker performance as the network of process plans grow larger and more complex due to the "dimensionality curse" [19].

Kusiak et al [3] formulated multi-objective optimization problems using two approaches. The first approach was based on "Graph Theoretic Formulation". A graph $G = (N, A)$ consisting of N nodes (process plans) linked by undirected arcs A where the objective was to find a set of connected nodes (plans) of a minimum total cost within such m-partite graph [3]. The second approach was basically an "Integer Programming Formulation" which required two more zero/one decision variables to

signify whether a process plan is selected (one), or is not selected (zero). The solution was to choose, for each set of process plans, only one representative plan so that the sum of their costs is minimized. Kusiak et al [3] developed a composite objective function for a single or multiple parts. Both of these formalisms utilized approximate heuristics to arrive at sub-optimal solutions.

Singh et al. [15], extended the previous work by considering a multi-objective formulation. They applied a min/max approach, developed by Osyczka [16] for that purpose. They considered a solution for a bi-criteria process plan with a third criterion as the bottleneck one. The number of objectives considered in their paper was small in order for decision making to be as clear and manageable as possible. The same authors have extended their technique to handle imprecise information regarding the various objective criteria [17].

Generally, the multi-objective problem should be formulated so a set of Pareto solutions can be generated [5, 20]. The Pareto optimal is a feasible solution to a multi-objective optimization problem wherein there exist no other feasible solution that will yield an improvement in the performance of one criterion without causing a decrease in performance of at least one other criterion [5]. A complete non-dominated set (Pareto set) is not of primary interest from the practical viewpoint [30]. A concept of representative non-dominated solutions is technically, as well as practically, more attractive. This falls into a class of techniques called the generating techniques.

A second set of techniques, namely "weight/utility" techniques, require the decision maker to provide some preference and weight values of relative importance to the various objectives beforehand. Also, a measure of deviation from these preferences is needed as a metric to find the best compromise solution(s) [5, 6, 13, 28]. Goal programming (both linear and nonlinear) has been applied to find the efficient

set of solutions [29]. In goal programming, an aspiration level is attached to each constraint, transforming it into a set of goals to be fulfilled during the optimization of the multi-objective function.

A third set of techniques, namely parameter space investigation technique, was introduced by Sobol [5]. This technique is basically an interactive procedure (also called Linear Parameter Table method LPt-method) and it is based on a deterministic sampling technique. It generates a very uniformly distributed sequence of multi-dimensional points that are used to generate trial points. These trial points are then incrementally ordered. The next step is an interactive step where bounds on each objective are set. After that, the sampled points are compared with the bounds to determine an admissible points set. If this set turns out to be too restrictive (an empty set exist) then some of the constraints of the problem are relaxed, i.e. the bounds are changed, and the process is repeated until a good compromise solution is found or the suggested constrained values are said to be incompatible [5]. The interactive nature and the large table look-up requirements of the parameter space investigation method makes it unwieldy to handle complex multi-objective networks.

The above techniques and many more including linear programming techniques that were developed to handle the multi-objective problem [6, 7, 23, 30], were reviewed. The survey revealed that these techniques suffer from the fact that the number of iterations, needed to converge to a compromise solution, is exponentially related to the number of objectives and the constraints of the multi-objective problem. Additionally, the techniques do not appear to be extendible to parallel implementation versions. This trait may pose an obstacle in applying them for real time operation as required by automated manufacturing process planning systems.

Next, Genetic Algorithms (GA's) are reviewed. A general background is presented first. Some GA–based techniques that handle related manufacturing problems are then discussed.

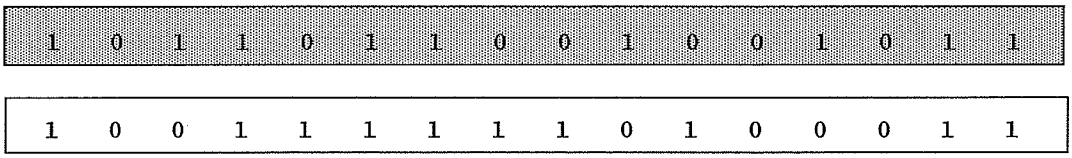
2.2 Elements of Genetic Algorithms

Genetic algorithms are directed randomized search procedures. They derive their power from the mechanics of natural selection and the survival of the fittest principles. Their application to optimization problems has been very successful due to the fact that they are well suited as global search techniques [31, 32, 33, 34]. GA have been applied to parameter identification problems of artificial systems [35]. They have also been developed as classifier systems [36].

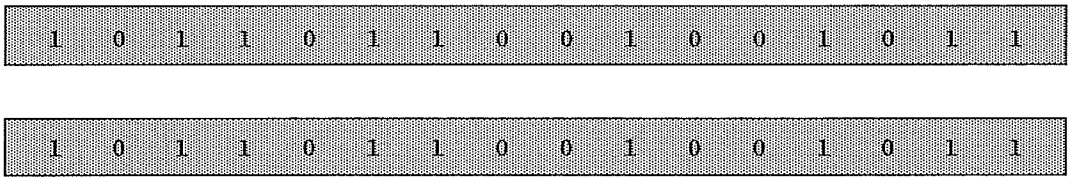
Genetic algorithms require the following innovations [31, 37]:

- introducing a finite string length of codings (of the binary or alphabetical type) of the search space parameters.
- applying a fitness function based on defined objectives of a particular problem in order to prune feasible solutions; no derivatives of the evaluation function are required.
- scanning solution points of the search space in a population–based approach to allow for parallel and global search.
- using a probabilistic transition rule in order to progress from one search space to the next.
- taking advantage of stage flow network formulation (natural to all multi-stage decision making problems).
- performing in robust and real time for complex dynamic problems.

Genetic algorithms consist of a randomly generated initial population (generation), within which each individual is evaluated by means of a fitness function in order to determine its performance index. Further generations are created by applying GA operators. Individuals in each of these generations are duplicated or eliminated according to their performance indices. There are usually three operators in a typical genetic algorithm [31]. The first is the reproduction operator which makes one or more copies of any given solution (i.e., individual), if the fitness value of such an individual is above the average fitness of individuals in a population the individual stays; otherwise, the individual is eliminated from the solution pool. For example, consider the following two individuals presented as binary strings:

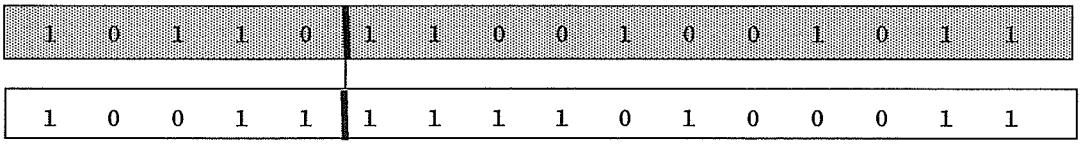


The first individual is considered to perform better than the second one. After the reproduction operator is applied, the first individual is duplicated; the second individual is eliminated from the population, due to its low performance, as shown below:

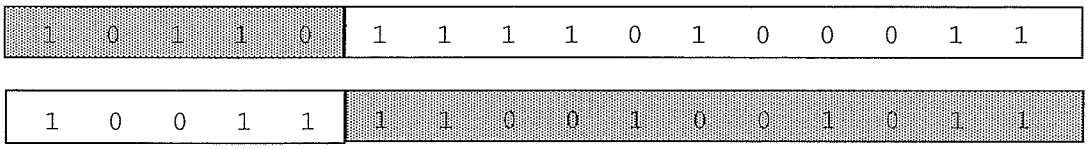


The second operator is the recombination (also known as the 'crossover') operator. This operator selects two individuals within the generation and a crossover site and performs a swapping operation of the string bits to the right hand side of the crossover site of both individuals.

For example, consider the following two strings selected for mating:

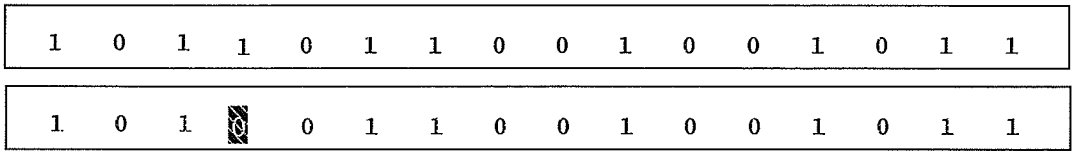


The outcome of the crossover operation is two individuals that possess some traits inherited from both parents as shown below:



Cross-over operations synthesize bits of knowledge gained from both parents exhibiting better than average performance. Therefore, the probability of a better performing offspring is greatly enhanced. The new individuals will then form a new generation which will be subjected to reproduction and crossover operations again until an optimal solution (fittest individual) is found.

The third operator is the 'mutation' operator. This operator acts as a background operator and is used to explore some of the unvisited points in the search space by randomly flipping a 'bit' in a population of strings. In the following example the mutation operation is carried out on the fourth bit of the string (individual):



Frequent application of this operator would lead to a completely random search. Therefore a very low probability is usually assigned to the activation of this operator. In the approach presented in this thesis, this concept has been applied differently compared to conventional genetic algorithm approaches and will be discussed later.

The coding that has been shown to be the optimal one is the binary coding [38]. The binary representation of search space points that is employed by the GA reveals another important property, namely the "building block" concept. The building blocks in GA consist of a third bit, in addition to the zero/one bits (alleles), that is the "do not care" bit which is shown below as an asterisk:

$$1\ 0\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ *\ 0\ 1$$

The "do not care" alleles (bits) allow for multi-regions points to be explored simultaneously since these bits can have either "one" or "zero" value, i.e. points (individuals in a population) that are described by the above building block (schemata) can represent a variety of points in multiple regions of the search space depending on the values of the "$*$" alleles. Therefore, one building block can have many points in separate regions allowing for a population of individuals to be explored in parallel; this, in fact, is called "implicit parallelism" [31]. The "implicit parallelism" concept is one of the most important concepts that gives GA a clear advantage over other optimization techniques. This is because a genetic algorithm that samples a certain amount of points (individuals) in the search space is actually sampling a vastly larger samples of regions in that search space [38, 39].

2.3 GA–Based Techniques Survey

The techniques reviewed in Section 2.1 are mathematical programming based, hence deterministic in nature. GA's are stochastic evolution techniques. GA's have proven their versatility in handling various optimization problems[31, 38, 39]. The previous applications of GA to solving some related optimization problems is now discussed.

Vigmanx [40] have proposed a GA for solving the linear transportation problem. Their technique involved a GA matrix representation structures of transportation networks, where linear and nonlinear cost matrices are manipulated by GA to reach an optimal cost network. New routines for GA operators were needed in order to accommodate their formulation; consequently, available GA parameters setting can not be employed by this technique.

Biegel [34] has proposed an application of a partially mapped crossover (PMX) operator GA to solve the job shop scheduling problem. The PMX operator was first introduced by Goldberg [41] in order to tackle various instances of the traveling salesman problem (TSP). The job shop scheduling problem was modeled as a TSP, thus GA could be applied to solve it. Applying a stochastic technique to solve this known non-polynomial complete problem (TSP) seems to be the prudent approach, specially in the large size instances of these problems. This technique has not been applied to PPSP in multi-stage manufacturing systems.

Thangaia et al [42] have proposed a GA vehicles routing of limited capacity and travel time by using "GIDEON" package. Their package consists of a global clustering module that assign customers to vehicles and a local routing optimization module. This technique is an interactive one and is demonstrated for a couple of objectives and small networks [42]. Other GA–based optimization techniques that are applied to various manufacturing and control problems have been described in [31, 32, 39, 41, 43, 44, 45, 46].

In order to handle PPSP using a genetic algorithm, a novel treatment and methodology which is different from the above surveyed techniques is presented in this thesis. Though it may be possible to apply the reviewed techniques to solve PP-SP, it does not appear to be advantageous compared to the model presented in this thesis since these techniques require more modification operations that constitute a

departure from utilizing a simple GA. This departure requires further comprehensive studies in order to evaluate the performance of GA operations in the context of PPSP.

The models presented in the remainder of this thesis, are readily incorporated within a simple GA. Therefore a wealth of parameters setting of GA operators, previously obtained by classical GA research, can be directly utilized. In addition to that, these models offer a new solution technique to handle multi-objective cases.

CHAPTER 3

Stage Matrix Formulation

The formulation of process plans as flow networks is presented here. The stage matrix method that facilitates the employment of simple genetic algorithms to find the best process plan among a large set of alternative process plans is discussed. The formulations include both single and multiple objective criteria for process planning selection problems.

In the zero/one stage matrix method, flow networks are represented as binary strings. Every entry in these strings corresponds to a single link in the network. This binary representation approach is essential when genetic algorithms are to be considered in most optimization problems [31, 38, 39, 41].

3.1 Flow Networks Binary Formalism

Flow networks are graphical representation tools that comprise nodes and arcs. They have been employed to model a variety of physical and conceptual cases [22]. One such application involves the modeling of the Process Planning Selection Problem (PPSP) as flow routing problem [20, 56]. As an example, Figure 3.1 shows a typical network of possible alternative routes for machining operations (turning, planing, drilling and finishing) that must be performed to transform a part into its final shape. In this network, each node represents a machining tool or a machining center. Each arc represents an operation that can be performed on a part type by the machine tool the arc is aiming at. The processing cost for such an operation is

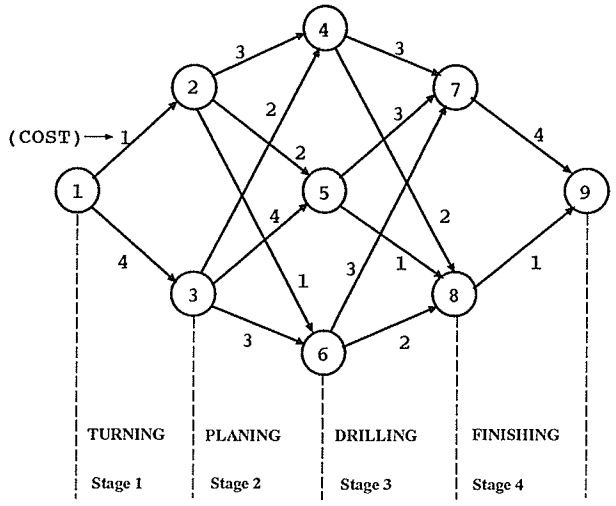


Figure 3.1: Flow network model of a multi-stage manufacturing system.

shown on the arc. For example arc $1 \rightarrow 2$ describes a turning operation to be performed by machine tool 2 at a cost of 1. Each set of similar (or same) manufacturing operations (arcs) is considered as a single stage.

In this work, a $(\hat{n} \times \hat{m})$ zero/one matrix is introduced to represent each stage in a flow network, where $\hat{n}$ stands for the input nodes and $\hat{m}$ stands for the output nodes. Referring to Figure 3.1, the stage 1 matrix has a dimension of (1×2) signifying that two alternative operations can be carried out in two nodes. The stage 2 matrix has a dimension of (2×3) indicating three alternative operations originated from the previous two nodes. The stage 3 matrix has a dimension of (3×2) which means that two alternative operations are available from each of the three previous nodes. Finally, the stage 4 matrix has a dimension of (2×1). A 'one' entry in each matrix signifies an existing connection between two nodes in the network which indicates that an operation is chosen along that particular arc. A 'zero' entry indicates that an arc in the network is not chosen; therefore, any path containing that link is not available.

Referring to the flow network shown in Figure 3.2, this network is similar to the

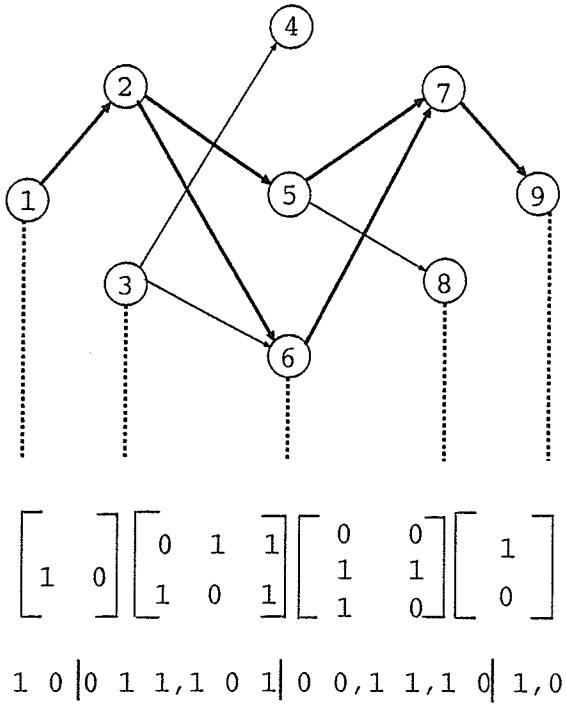


Figure 3.2: Multiple paths flow network.

one shown in Figure 3.1 except for some missing arcs . The network represents a randomly generated individual to be evaluated by a GA. The matrix representation of this network is also shown in Figure 3.2. The four matrices represent the four machining stages. The stage three matrix has three rows (representing nodes 4, 5 and 6) and two columns (representing nodes 7 and 8). No connection is allowed from node (machine) 4 to nodes 7 and 8; thus, the first row of the matrix consists of two zeros. The travel of parts along arcs $5 \rightarrow 7$ and $5 \rightarrow 8$ is allowed; this is shown by two ones in the second row of the matrix. Finally, machining of parts located at node 6 can only be done by machine 7; therefore, the first entry in the third row is one and the second entry is zero.

The zero/one matrix entries are then mapped onto a zero/one string format by concatenating the rows of each matrix, side by side, to form a finite length string (see Figure 3.2). This type of mapping is necessary to apply conventional GA operators.

Since GA is only applied to evaluate a single path of a network at a time, several instances (population) of the original network must be created. Each instance of these networks must consist of a single path. The initially generated random population of networks of the type shown in Figure 3.2 attempt to represent single path networks, i.e. instances of the general network of Figure 3.1. These network instances possess some arcs that have been randomly chosen and disabled in order to create a single path network. However, these instances may comprise more than one path or they may consist of no complete path. In order to handle these scenarios, a modification procedure must be applied to insure a single path existence per an instance network. For example consider the network, shown in Figure 3.2, which displays the availability of two alternative paths, as shown in the solid lines: $1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9$ and $1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 9$. The matrix representation allows for using a simple procedure to ensure the fulfillment of this condition. This is done by multiplying the stage matrices together. The final product of such multiplications will indicate the status of the network's individuals, i.e. the number of distinct paths of an individual. For the case shown in Figure 3.2 one can identify two paths:

$$\begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \end{pmatrix}$$

The product matrix that yielded this outcome is:

$$\begin{pmatrix} 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \end{pmatrix}$$

The first column of the first product matrix is 2; thus, the first column of the third stage matrix, $\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \end{pmatrix}$ needs to be modified. One then randomly selects a 'one' value entry in the first column (e.g., the 'one' in the second row) and replaces

it with a 'zero' value entry. The multiplications of the remaining stage matrices are then continued using the new matrix, $\begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$, in order to either confirm the existence of a single path network or identify another multiplicity. The final outcome of the multiple path modifier is shown in Figure 3.3.

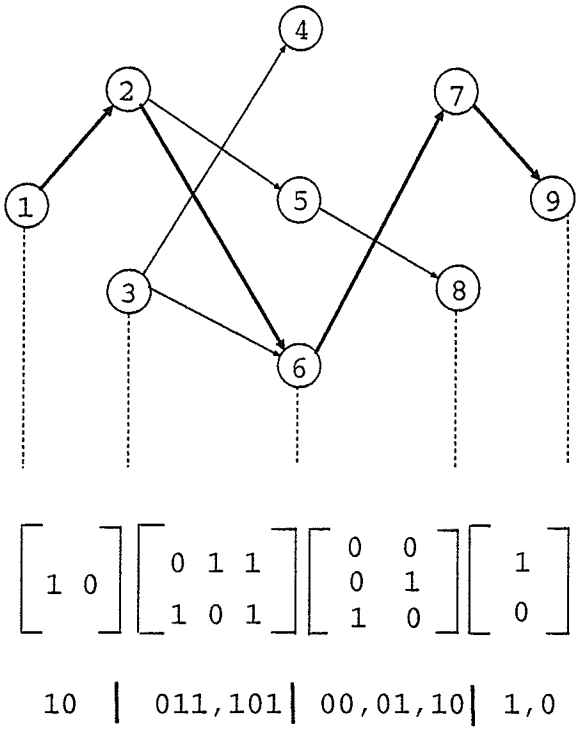


Figure 3.3: A single path flow network.

The binary representation of the arcs, in flow networks, allows for a proper mapping of the objective function of the particular optimization problem, PPSP in this work, to the corresponding fitness function as described in Sections 3.2.1 and 3.2.2).

3.2 GA–Based Evaluation Functions

3.2.1 The Single-Objective Case

The main objective function that describes the shortest path problem (Equation 2.1) is mapped onto a fitness function, $Y_{j(j=1,2,..,p)}$ of an individual network in a population of p networks having a single objective criterion (e.g., minimizing cost). This fitness function is formulated as shown below:

$$Y_{j(j=1,2,..,p)} = U - \sum_{i=1}^{m} C_i X_i \tag{3.1}$$

where $C_{i(i=1,2,..,m)}$ is a cost associated with each arc, m is the total number of arcs in a network, $X_{i(i=1,2,..,m)}$ is a zero/one variable; X_i is one if a link is part of the path and is zero otherwise. U is a bias (upper bound) to ensure a positive fitness and to set a limit for the GA reproduction operator (note: only individuals with positive fitness can reproduce).

In order to satisfy the constraints shown in Equation (2.2), the formulation presented in this work assumes a single flow item at each node in the network. Also, an item is supplied at the source node and is received at the sink node of the network. These constraints shown in Equations (2.2) to (2.4) are handled in the modification procedures which will be disussed in Section 3.3.

The fitness function is a direct mapping of the objective function that describes the shortest path problem, i.e. finding the least cost path (process plan) in a flow network. The upper bound does not necessarily entail a feasible solution point. It can be selected by summing the largest cost value of each stage together or by selecting some large hypothetical cost value. This value has been employed here as a mapping factor of the objective function towards a metric of performance indices (fitness values) . For example, a path in a network that possesses high cost value

will have a low fitness value and thus may be eliminated from the next generation of paths that are explored by the GA. Similarly, paths (individuals) with high fitness values will be reproduced by the reproduction operator.

The single objective evaluator explained here will be demonstrated with a network in Section 3.3.

3.2.2 The Multi-Objective Case

Most real life process planners deal with multiple objectives, usually of the conflicting type. A generalized process planning model must handle this type of objectives. The metric introduced here, to find a compromise solution for multi-objective criteria, is based on the notion of a direct contribution of each objective criterion towards a fitness value of the total links (or nodes) that constitute a single individual network. Various objectives affect increasingly or decreasingly (according to their maximisation or minimization nature) the fitness value of each individual.

In this section, a technique is described that is of the generating multi-objective type. Generating techniques are based on the idea of generating all non-inferior solutions that satisfy all objectives and constraints [5, 6, 23, 29, 30]. These solutions are then scanned for the best compromise result(s), also called Pareto solutions. These latter solutions have the particular characteristic where an improvement in one or more objective(s) will inadvertently lead to a degradation with respect to the rest of the objectives. In this GA–based technique, a direct calculation of the contribution of optimizing the separate objectives (e.g. cost and quality) to a fitness function $Y_{j(j=1,2,\ldots,p)}$ can be carried out for the jth individual, by the following formulation:

$$Y_{j(j=1,2,\ldots,p)} = \sum_{k=1}^{l_1} \omega_k (U_k - \sum_{i=1}^{m} C_{ik} X_i)/U_k + \sum_{q=l_1+1}^{l} \omega_q \sum_{i=1}^{m} Q_{iq} X_i / U_q \qquad (3.2)$$

$$\sum_{k=1}^{l} \omega_k = 1 \tag{3.3}$$

where, C_{ik} is the kth minimization objective value associated with link i, Q_{iq} is the qth maximisation objective value associated with link i, U_k and U_q are the upper bound on the maximum value of all minimization and maximization objectives respectively, l_1 is the number of minimization objectives and $l - l_1$ is the number of maximization objectives.

ω_k and ω_q are the weighting factors; each is determined based on its degree of importance in relation to the other objective criteria.

The fitness function described above is directly mapped from the objective function that describes the problem domain. This mapping was carried out by subtracting from an upper bound values "U_k"and "U_q" which are calculated separately for each objective, by the summation of the maximum values of the objective criteria of each link for all stages. An example of this formulation is presented in Section (3.4.2).

The above formulation is not the only way to assess the fitness function of an individual network. For example, one can adopt a method that is based on Min/Max approach as shown in Appendix A. Both of these methods converge to a best compromise solution. However, the method presented in this section requires less user (or decision maker) intervention; this is elaborated in Appendix A.

Network problems of pairs of conflicting objectives are called the "bi-criteria" problems [15]. For such cases, the formulation described by Equations (3.2) and (3.3) can be modified so as the first term of Equation (3.2) is mapped onto a new term that accommodates for the conflicting criteria of the problem. This mapping is demonstrated in Appendix B (Section B.1) where the relevant conflicting objective criteria, i.e. the first term of Equation (3.2), are formulated in a manner that reflects their conflicting nature.

In the case where only the 'pure' bi-criteria, i.e. only two conflicting criteria network, is being considered, this new mapping will yield the resultant contribution of these criteria to the fitness values, which in turns represents the compromise solution of the bi-criteria network. A detailed description of the bi-criteria formulation is first illustrated by using a simple example in Appendix B. A large size network example is then handled in that Appendix.

Alternatively a second solution technique for 'pure' bi-criteria cases is presented in Appendix B (Section B.2). This technique is based on calculating a utility value (analogous to the fitness value) associated with a path. The performance of this bi-criteria evaluator is indicated with a simple network in Appendix B.

3.3 Algorithms and Implementations

Based on the formulations presented earlier, a computer package that can handle single and multiple objectives problems which can be modeled as flow networks have been developed. Figure 3.4 shows the four modules of the software flow-chart. The first module, called here the "Supervisory Procedure Module", involves building the flow network model where all GA parameters are initialized and the parameters of these networks are input to the software. The second module, the "Modification Procedure Module" contains an algorithm which insures that a single path per network is produced at a time.

The third module, "Fitness Evaluator Module", evaluates a population of distinct single path networks based on an objective criteria. This module consists of a number of sub-modules designed to handle distinct classes of optimization problems. These modules have been described earlier in Sections 3.2.1 and 3.2.2. Finally, the fourth module includes GA operators, i.e. the reproduction and the recombination

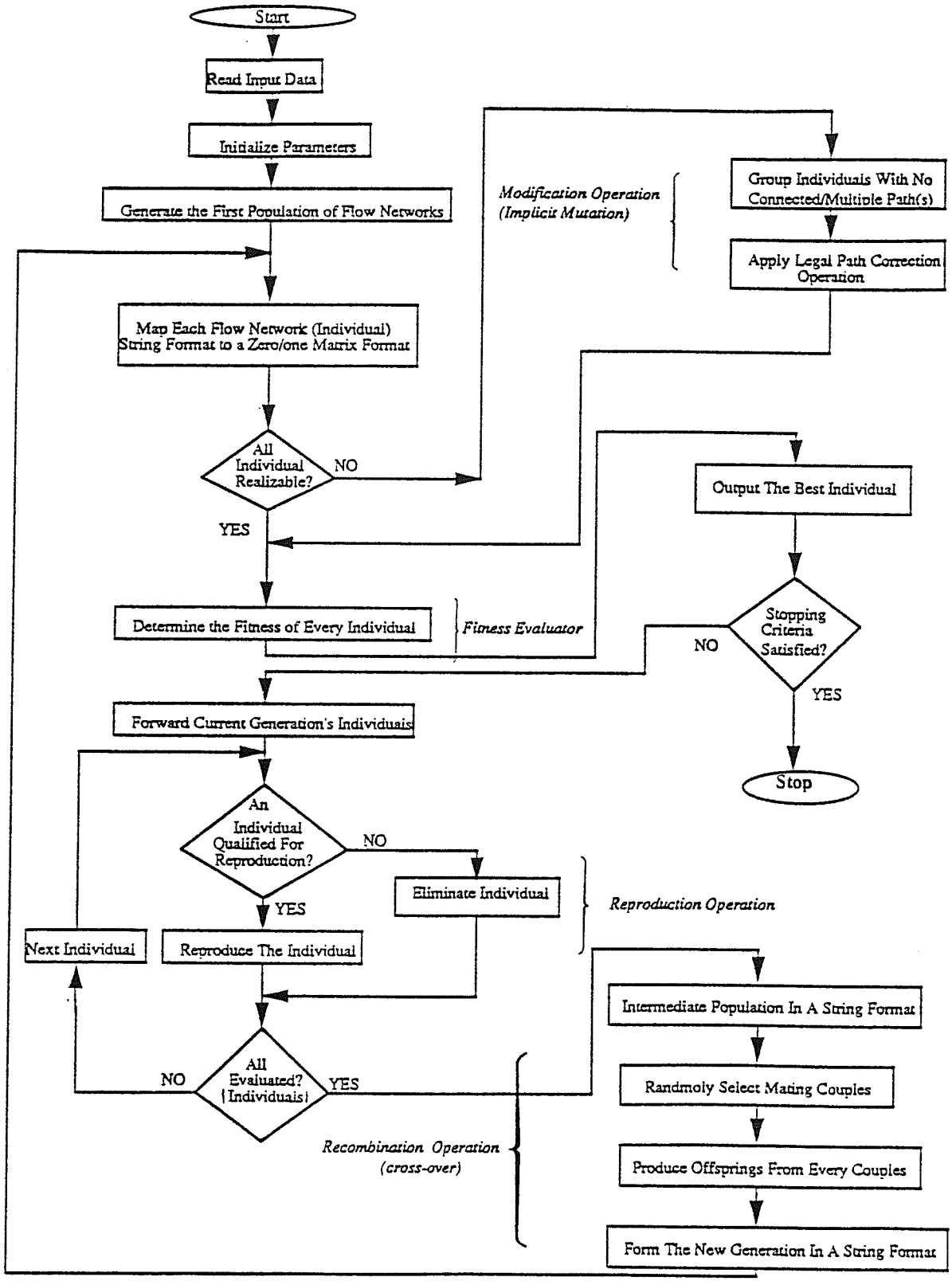


Figure 3.4: Overview of flow routing network program.

3.3.1 Flow Network Generator

The input to the program is carried out through the flow network generator procedure. It includes all the data necessary to initialize the program. These data are:

a– parameters describing the flow network model such as the number of stages, the number of nodes and the number of arcs; costs or other objectives associated with each arc.

b– parameters related to genetic algorithms operators i.e., crossover probability, size of the population, etc.

c– stopping criteria, e.g., the maximum number of generations, the number of runs or the percentage of the best individuals in a population.

d– The type of optimization problem (single, multi-objective or bi-criteria flow networks) along with their corresponding parameters such as the upper and lower limits on the objective values for all network stages and the bias factor applied, to these values, to enhance the search of the solution space points.

3.3.2 The Modification Procedures

In order to apply genetic algorithms, each path must be represented as a unique (feasible and complete) individual in a population of individuals that describe the solution space of the entire network. Therefore, a condition of one path per individual member of a population must be satisfied [47]. The network containing multiple paths or no complete path must be modified by a procedure called the "path correction operation". This operation is a structured randomised modification procedure that eliminates or adds extra links to such networks so that only one path exists. The modification procedure involves the following functions:

(1)- Path Identifier:

multiply all stage matrices sequentially; if the final outcome of the multiplication is 'one', proceed to the next individual (network) in the population pool; if the outcome is greater than 'one', implement the multiple path correction procedure (step two); if the outcome is 'zero', implement the disconnected path correction procedure (step three). For example, in the flow network shown in Figure 3.2, the stage matrices, representing this network, were multiplied together and produced a value of 2 indicating two possible paths.

(2)- Multiple Path Corrector:

multiply the stage matrices, one at a time, until a product matrix with an element greater in value than 'one' is reached; the stage corresponding to the latest matrix is the probable cause of multiple paths; identify the column of the stage matrix which caused this multiplicity; randomly assign a 'zero' value to an entry of that column and produce a new matrix; continue to multiply the remaining stage matrices, including the new one, together until another multiple path is identified. This procedure has been illustrated earlier in Section 3.1.

(3)- Disconnected Path Corrector:

multiply the stage matrices, one at a time, until an all zeros product (null) matrix is reached; the stage corresponding to the latest matrix is the cause of the disconnected path; pivot on the all zeros row of the stage matrix which is the cause of disconnection and randomly assign a 'one' value to an entry of the pivotted row; replace the stage matrix with the new one and continue to multiply all the remaining stages, including the new one, together until another disconnection is identified. The mechanism of this operator is similar to the multiple path corrector.

The modification procedure described here is structured because it operates on potential offspring which are the products of well-performing parents. The procedure also has elements of randomness because it pivots on a portion of the parents' networks. It then synthesizes, in a random fashion, the remaining portion of the network by deleting or adding extra links towards a distinct path network.

Another novel contribution of this work lies in the fact that extra links were intentionally allowed in each individual coding so long as they did not violate the condition of one path per individual. From Figure 3.3 it is seen that more links than are necessary to form a single path exist in the network such as: $3 \rightarrow 4$, $3 \rightarrow 6$, $5 \rightarrow 8$, etc. During the course of this study, an observation was made that these extra links add richness and variety to possible combinations of the reproduction and the crossover operators, without incurring any extra cost. This is because only the costs associated with a distinct path are calculated in the fitness function. The justification for allowing extra links has precedence in nature, where hidden traits (or recessive genes) can be passed through generations of individuals in many different forms.

3.3.3 The Fitness Evaluator Module

Once all the individuals in the population are determined to be realizable, ie., found to contain a single path, they are thenevaluated by the fitness evaluator. The fitness evaluator utilizes an evaluation function in order to determine a performance index for each network. This is described in detail in Sections 3.2.1 and 3.2.2. Also, several examples based on this method will be analyzed in Section 3.5.2 of this chapter.

3.3.4 GA Procedures

Every individual in the population represents a realizable path in the network. Once a path in an individual network is selected, it is evaluated for its performance by means of a fitness function, as described previously. Individuals with high fitness values are reproduced by the reproduction GA operator. Individuals with below average fitness values are eliminated from the reproduction population. This operator makes one or more copies of a well performing individual compared to the rest of individuals in the population. It eliminates those individuals with poor fitness values.

The above procedure allows the GA to keep the best performing individuals in a population. These best performing individuals are then randomly selected to be mated at a pre-specified probability (crossover probability). This operation is carried out by the crossover operator on the string format of the individuals. A swapping operation of bits (nodes or links) to the right-hand side of a mutual crossover site, chosen randomly, on all selected couples is carried out by the crossover operation [31, 38, 39]. An example of the crossover operation is shown Figure 3.5, where two individuals (paths) are mated and the outcome of the operation is two individuals that possess some traits inherited from both parent individuals. It can be observed that the first offspring has a single distinct path with some extra links; where as the second offspring comprises two paths. Therefore, the modification operation must be invoked to transform this offspring into a single path network (as was described in the modification operation section in this chapter).

As noted earlier, "good" individuals possess high fitness values and usually share substantial portions (partial solutions) of the optimal solution(s). When these individuals are mated together these portions are synthesized, normally leading to a better performing population of paths than the previous generation [31]. Conse-

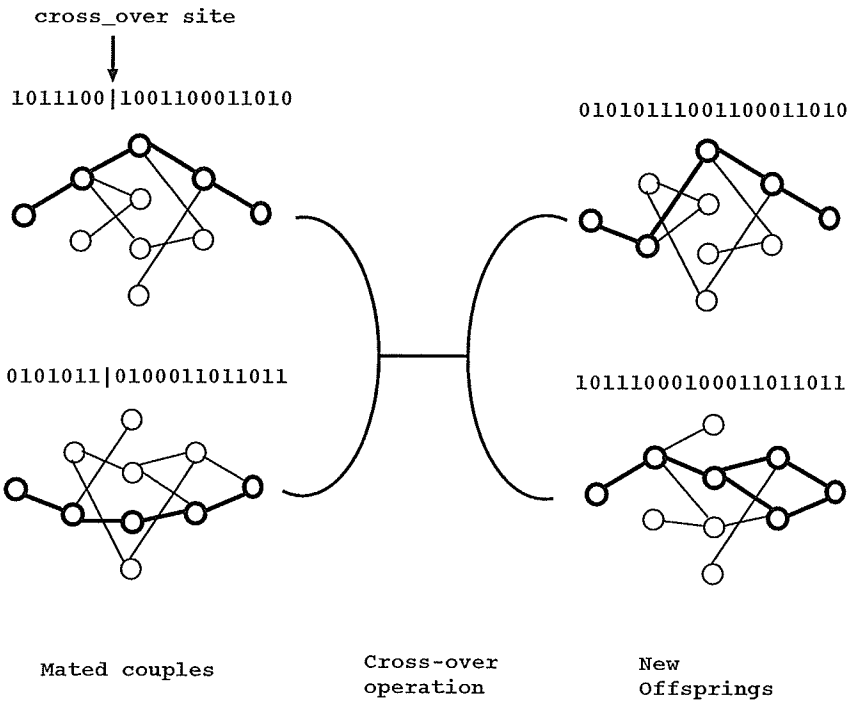


Figure 3.5: Recombination operation (crossover).

quently, new individuals (offspring) are created and are sent to the fitness function for evaluation of their performance. The cycle is repeated until a convergence to a single best solution is reached.

3.4 Demonstrative Examples

3.4.1 A Single Objective Case

The routing network shown in Figure 3.1 is now used to demonstrate the developed algorithm and to further illustrate the GA fitness function and operations. The objective is to find the minimum cost path. Figure 3.6 shows how GA generates the initial generation first as binary strings (column one in Figure 3.6) and then mapped these strings onto a zero/one stage matrices format (column three). The second column shows the flow network representation of these individuals.

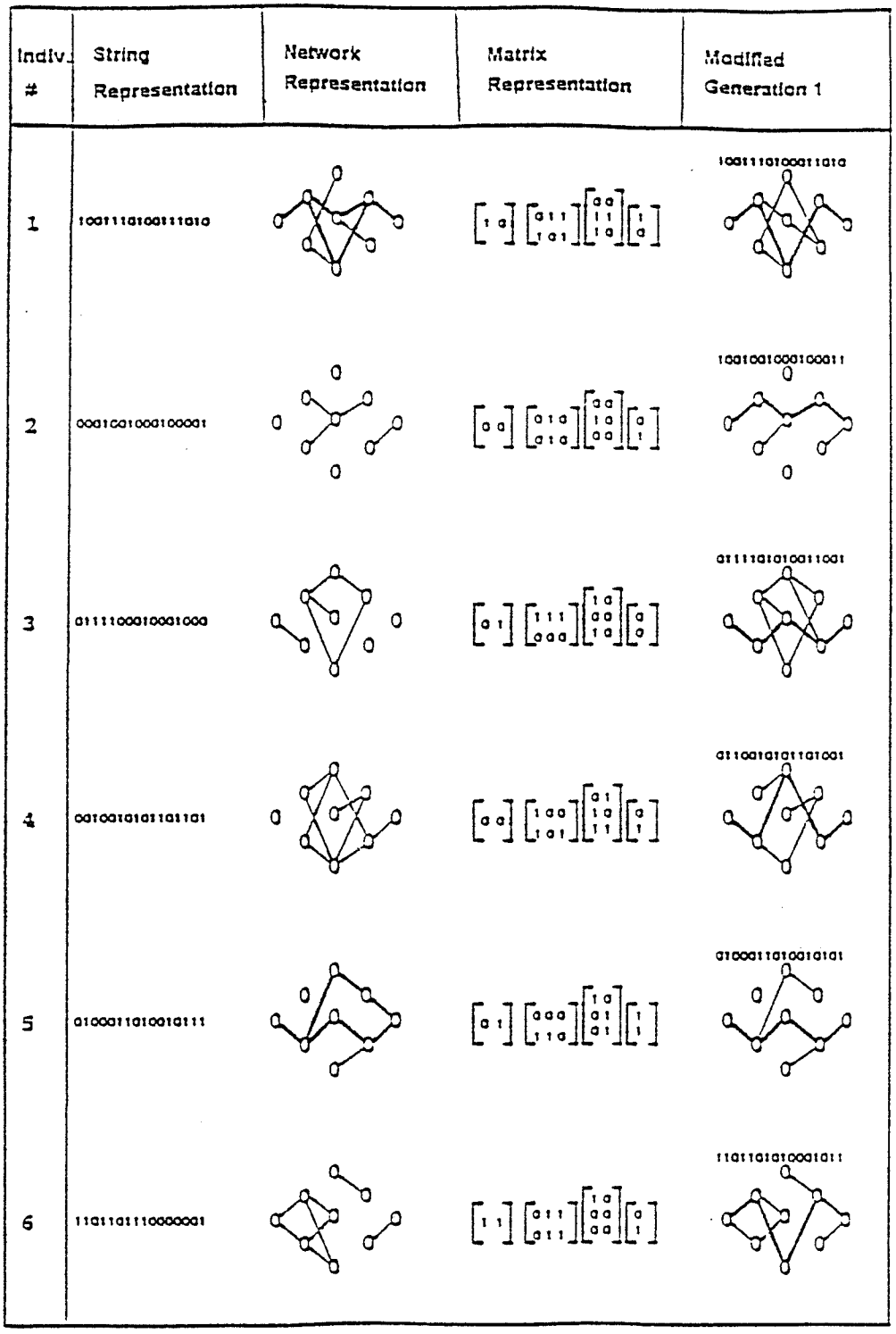


Figure 3.6: Production of initial generation.

The fourth column depicts the generation of modified individuals (flow networks) after applying the path modification operation. This initial generation is now ready to be manipulated by genetic algorithm operators. Every generation is evaluated by the fitness function and is evolved by the genetic algorithm operators. Figure 3.7 shows a typical transition from generation 3 to generation 4. As is seen, the best performing individual (with a high fitness value) in that generation was duplicated (more than one copy could be created) depending on the average performance of the entire group of individuals in the population. An instant of such reproduction is shown in the third column of Figure 3.7: the first individual in the third generation had a high fitness value, therefore two copies of this individual (which happens to be the minimum cost solution for this particular example) were generated. The fifth individual was eliminated from the pool of solution paths since it had a low fitness value.

The newly reproduced population after the crossover operation is shown in the fifth column of Figure 3.7. The crossover sites (shown as vertical bars) and the mating couples are chosen randomly. The result of this operation is shown in the sixth column. Once a new population of offspring was obtained, the modification operation was applied to ensure that only one distinct path exists in each network. This is shown in the last column of Figure 3.7 as generation 4; it will subsequently be subjected to the same process in order to bring forth an optimal generation. A sample of the evolution process is shown in Appendix C; where 9 generations are evolved and are depicted.

In this example, the population size chosen was 6. Note that the number of alternative routes was 12. The optimal solution obtained is $1 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 9$, this process plan calls for a turning operation to be performed on the second node (machine); a planing operation to be carried out on the fifth machine, a drilling.

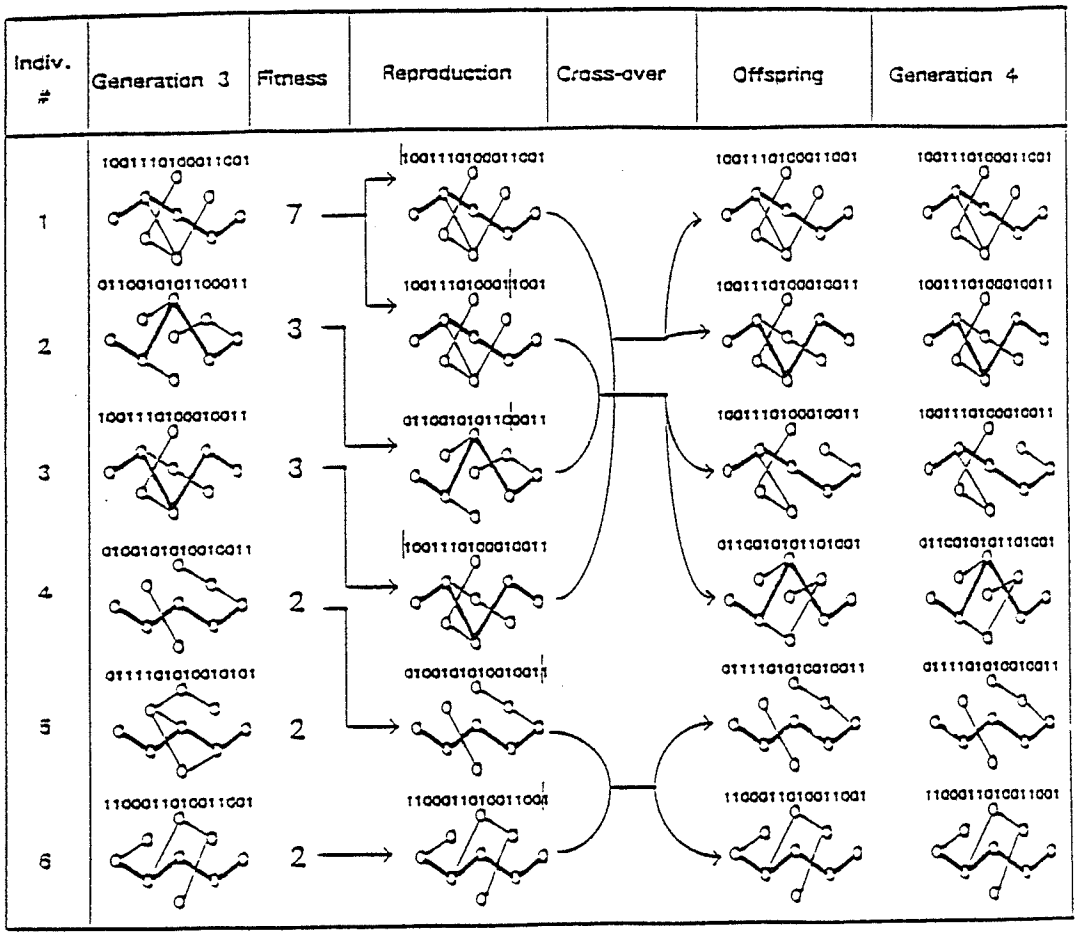


Figure 3.7: Evolution of a generation treated by GA operators: generation no. 3.

operation to be performed on machine 8 and a finishing operation to be done on machine 9, respectively. Figure 3.8 shows the convergence behavior. One optimal solution is reached after only two generations with the algorithm steadily improving its performance in later generations, until it reaches the ninth generation where all individuals are optimal. The algorithm then maintained such performance throughout the trial run's thirty generations.

Figure 3.9 displays typical convergence histories of the algorithm. The objective function of the best fit individual along with the fitness function and the average fitness of individuals within a population are plotted versus the number of generations. It is seen that the shortest path solution (the best solution) is found after the second generation. The algorithm kept discovering the optimum solution throughout the subsequent generations. Studying the behavior of the average fitness values of each generation revealed a steady improvement of the algorithm's performance; i.e., progressively better population of individuals (solution points) with good fitness values were consistently generated by the algorithm. This behavior was consistently observed over many runs of the computer program.

Although the mutation operator was not explicitly employed in this work, an implicit application of a similar form of mutation was, however, carried out within the modification operator.

3.4.2 A Multi-Objectives Case

The previous section presented a single objective example. The same example is employed here for the multi-objective case where another objective is added to the earlier network example. The emphasis, in this section, is on the difference between the two formulations, i.e. the fitness function evaluation and not on the GA implementation since GA arrive at a single value that is obtained from applying

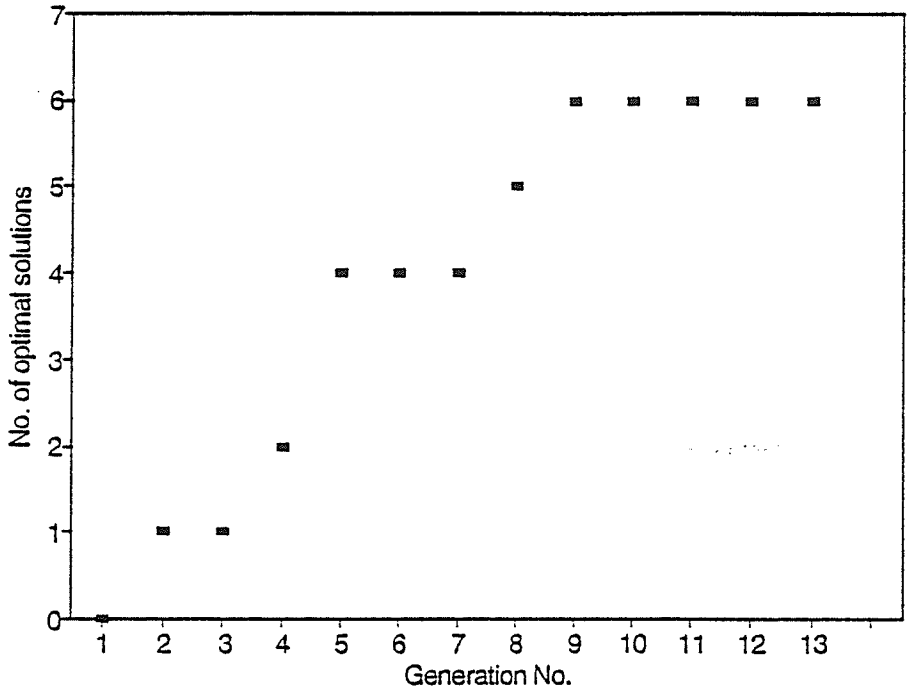


Figure 3.8: Number of optimal solutions per generation.

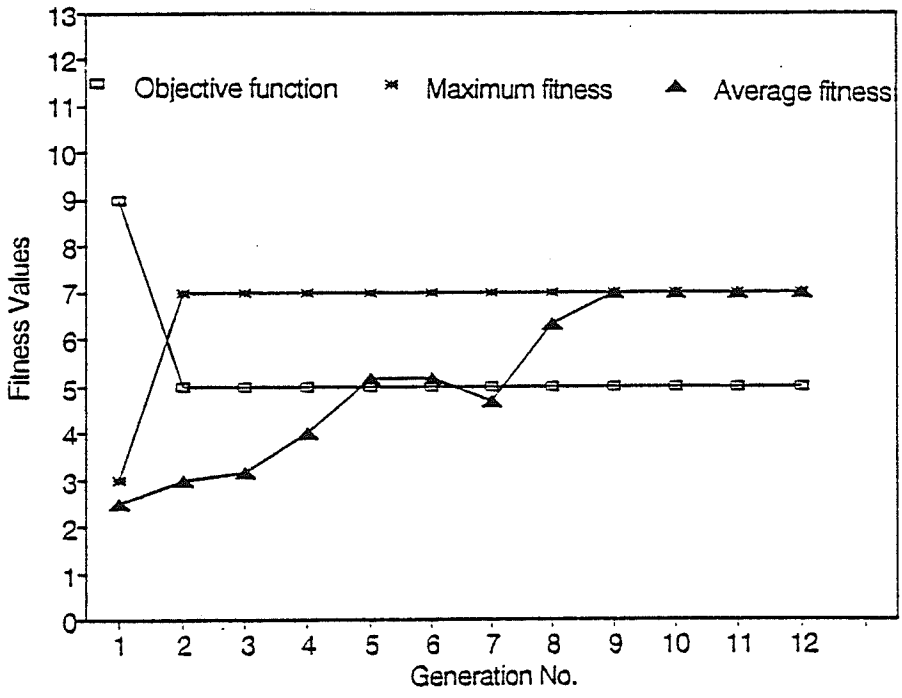


Figure 3.9: Convergence behavior.

the fitness function to evaluate both cases.

Figure 3.10 shows a flow network having two objective criteria – minimizing cost and maximizing quality. This network presents 12 alternative paths. Assuming that all possible alternative paths were selected to form a population of individuals, the result of calculating the fitness values of all these distinct paths, using the multi-objective formalism described by Equation (3.2), are summarized in Table 3.1. Equal weights were given to both criteria ($w_{k(=1,2)} = 0.5$). Every criterion considered at each machine or machining center is dependent on the age of the machine, the type of tools, the quality of the operators, and technological requirements and limitations.. The sequences of paths are shown in column two in the table. The third and the fourth columns show the corresponding cost and quality criteria, respectively. The fifth column depicts the fitness values. The last column of Table 3.1 depicts the ranking of each path according to their performance indicated by the fitness evaluator.

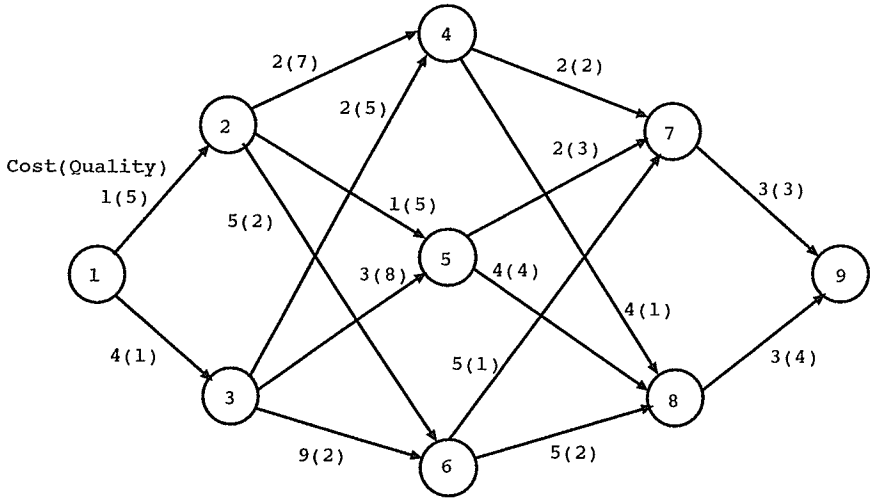


Figure 3.10: Typical multi-objective manufacturing flow network.

It is seen that the last two individuals have very poor fitness values compared to the rest of the population. Using GA operators, these individuals will be dropped

Table 3.1: Multi-objective network path evaluation.

Path	Sequence	1^{st} Objective (cost)	2^{nd} Objective (quality)	Solution Point (fitness values)	Rank
1	$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 9$	10	17	{ 1.333 }	3
2	$1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 9$	8	17	{ 1.429 }	1
3	$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9$	7	16	{ 1.428 }	2
4	$1 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 9$	9	18	{ 1.429 }	1
5	$1 \rightarrow 2 \rightarrow 6 \rightarrow 8 \rightarrow 9$	14	13	{ 0.953 }	6
6	$1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 9$	14	11	{ 0.857 }	8
7	$1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 9$	11	11	{ 1.00}	5
8	$1 \rightarrow 3 \rightarrow 4 \rightarrow 8 \rightarrow 9$	13	11	{ 0.904 }	7
9	$1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9$	12	15	{ 1.143 }	4
10	$1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9$	14	17	{ 1.143 }	4
11	$1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9$	21	10	{ 0.476 }	9
12	$1 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9$	21	9	{ 0.476 }	9

from the next population and will be replaced by the best performing individuals, i.e., individuals 2 and 4.

More examples, including GA operations for large networks that describe multi-objective problems, are presented and analyzed in the remaining sections of this chapter.

3.5 Experiments

3.5.1 Convergence Behavior

Consider the routing network shown in Figure 3.11 which is used to analyze the developed program. The objective was to find the least cost process plan (shortest path) amongst all possible paths in the network. The cost values are shown as integer value along every arc. Note that the network comprises 7 stages, 24 nodes, 80 arcs and the total number of 1440 possible paths.

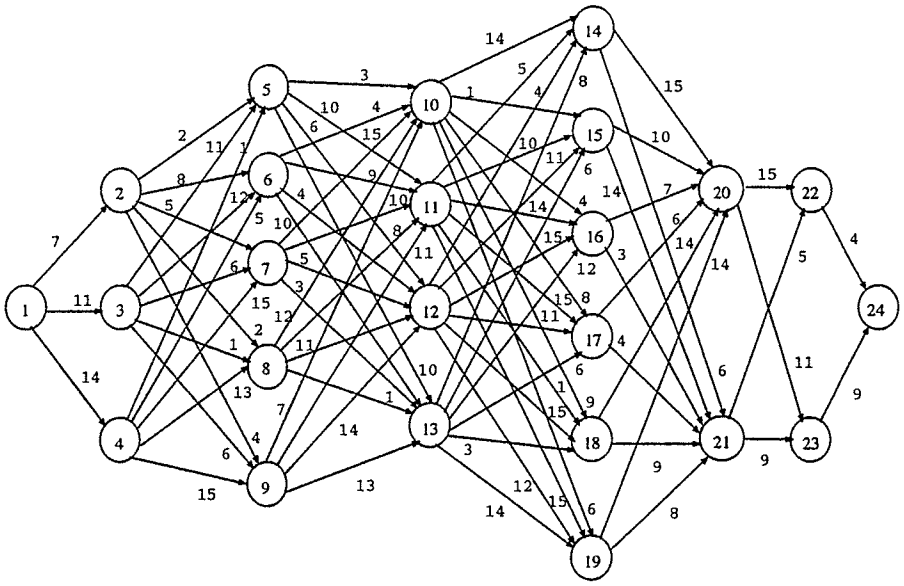


Figure 3.11: Typical single objective manufacturing flow network.

Each trial run of the program started with a randomly created generation of

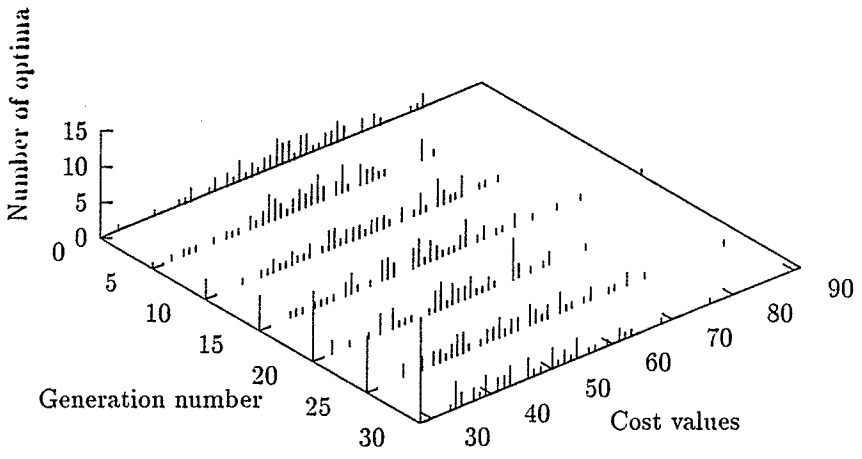


Figure 3.12: Distribution of solution points per generation.

individuals. The program was allowed to evolve this generation up to 30 times. Additionally the algorithm was allowed to run 20 times (number of trials), each time with a new starting generation. The twenty trial runs chosen for this application comprise twenty, potentially different, regions of the search space in order to globally scan that space towards an optimal solution.

Figure 3.12 shows the result of a typical trial run. It depicts the number of individuals of certain costs corresponding to a generation. For example, the tenth generation contains four individuals having a minimum cost of 28. The trend is towards a constant improvement from generation to generation, i.e., more optimal solutions are generated. Generation 30 consists of fifteen individuals which are the optimal solutions. The minimum cost path $1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$ was however found in the fifth generation. This result was also confirmed with an available commercial package [24].

Similar behavior was observed over other trial runs. The result of these observations is summarized in Figure 3.13, where the average fitness values of each

generation for all twenty runs are plotted. A steady improvement of the average performance at each generation is observed with a leveling off starting at generation number 21. The trend depicted by the solid line shows a good indication towards estimating the required number of generations in order to reach an optimal solution in this case study. It is also seen that the algorithm is consistent in finding the best solution and maintaining it throughout the future generations. In this example, over twenty runs, only one run did not find the optimal solution within the first 30 generations. However, the next best solution was discovered in that run and through most of other trial runs.

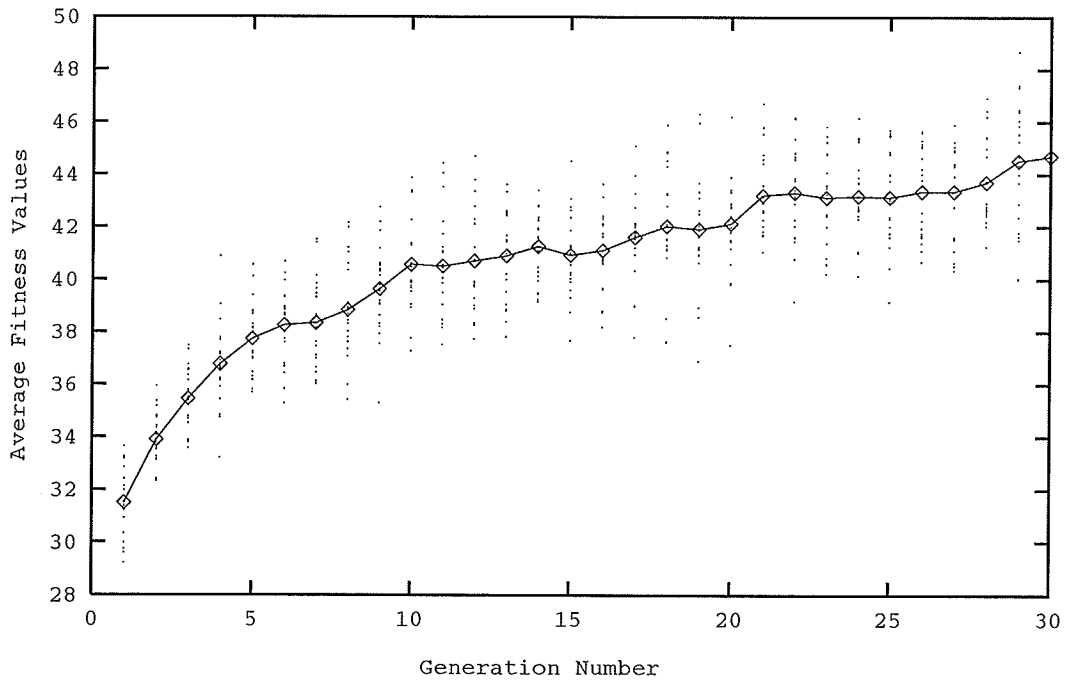


Figure 3.13: Mean fitness values at different runs.

In this experiment, the setting of the GA parameters were as follows:

Population size $p = 60$ – The larger the size of the population, the better the scanning ability of the algorithm of the solution space. However, an extra large population requires proportionally large memory space and computational time [31, 45]. Figure 3.14-(*curve a*) shows the number of trial runs terminating with

optimal solutions for various population sizes. The results are consistent with [50] who indicated that the best on-line performance can be obtained with a population size in the range of 30-100, while the best off-line performance is associated with a population size of 60-110.

Cross-over probability $P_c = 1.0$ – Referring to Figure 3.14-(*curve b*) it was found that a high probability of crossover range ($0.8 \rightarrow 1.0$) offered good performance in obtaining optimal individuals per trial run. A study by Gefenstette [50] suggested that for such a population structure, good performance is associated with either a high crossover rate combined with a low mutation rate, or a low crossover rate combined with a high mutation rate. This suggests that the mutation that is implicitly applied during the path correction operation in the algorithm must have a low probability rate.

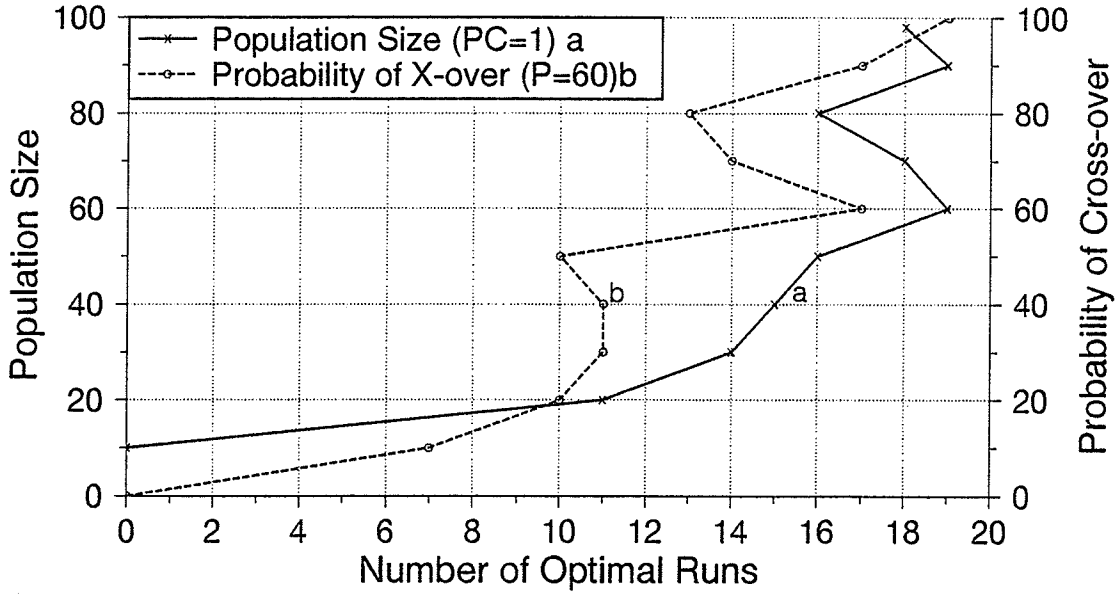


Figure 3.14: Effect of population size and probability crossover on convergence.

Bias value $U = 92$ is a mapping factor that transforms the objective function into a fitness function. U can simply be chosen as sum of the maximum cost at

each stage. A small value for U may eliminate many solution points; a very large value for U hindered the process of elimination towards the best solution. Further adjustment of this value focuses the search within the feasible solution space and improves the speed of the convergence to the best solution.

In order to demonstrate the efficacy of the developed techniques in handling multi-objective problems, a second objective criteria is added to the networks. Referring to Table 3.2, ten networks of various sizes were randomly generated: the number of stages were determined to be in the range of 7 to 15. For each stage, the number of output nodes were randomly chosen to be between 1 to 9. For each arc in the network, cost and quality values were assigned. These were randomly generated values between 1 to 15 and 1 to 50 and have weights of 0.4 and 0.6, respectively. Any other combinations of weights can be used, depending on the perceived importance of the corresponding criteria, by a decision maker.

The table depicts the size of these networks, the type of stages, total number of nodes and arcs. The upper and lower values for both cost and quality objectives, respectively, were chosen simply by summing the maximum cost or the minimum quality at each stage. These are listed in Table 3.2. The separate optimum values of minimum cost and maximum quality paths are listed. The next two columns of the table show the maximum (worst) cost and minimum (worst) quality. Finally, the last column gives the compromise solution for both cost and quality. A detailed analysis of the first example shown in Table 3.2 was carried out earlier. The parameters setting of the rest of the examples were treated as follows. The population size was selected as 100. The probability of crossover was set at $P_c = 0.80$ and the upper and lower values of biases were easily determined by summing the maximum cost and the maximum quality over the stages. These parameters were kept constant throughout the entire test. The strategy that is adopted to terminate the search is

Table 3.2: Single and multi-objective network parameters.

#	Type of Network Stages	Number of Nodes	number of Arcs	U Object. #1	U Object. #2	Minimum Cost Path	Maximum Quality Path	Maximum Cost Path	Minimum Quality Path	Compromise Solutions $\{Cost; Quality\}$
1	{1}{3}{5}{4} {6}{2}{2}{1}	24	80	98	148	27	144	93	28	{36; 117}
2	{1}{2}{2}{8} {3}{5}{5}{1}	27	91	95	208	28	193	91	28	{37; 182}
3	{1}{8}{3}{7} {2}{4}{8}{4} {1}	38	143	107	231	24	209	102	54	{29; 197}
4	{1}{7}{2}{4} {3}{6}{4}{3} {6}{1}	37	119	127	255	11	217	116	61	{56; 209}
5	{1}{8}{7}{2} {7}{2}{4}{7} {3}{5}{1}	47	183	141	288	19	266	135	62	{59; 249}

Table 3.2: cont'd.

#	Type of Network Stages	Number of Nodes	number of Arcs	U Object. #1	U Object. #2	Minimum Cost Path	Maximum Quality Path	Maximum Cost Path	Minimum Quality Path	Compromise Solutions (*Cost*; *Quality*)
6	{1}{8}{2}{5} {2}{5}{4}{6} {7}{6}{6}{1}	53	224	154	300	28	267	147	56	{61; 259}
7	{1}{6}{8}{7} {6}{5}{2}{10} {2}{9}{3}{3} {1}	63	285	173	352	33	328	169	75	{67; 295}
8	{1}{8}{5}{7} {4}{7}{5}{2} {6}{3}{8}{7} {8}{1}	72	358	183	377	35	370	179	61	{58; 309}
9	{1}{5}{2}{9} {4}{8}{9}{8} {5}{4}{8}{6} {5}{4}{1}	79	439	196	410	32	386	183	73	{66; 326}
10	{1}{7}{6}{9} {2}{5}{5}{8} {6}{8}{6}{5} {4}{7}{9}{1}	89	490	220	434	34	404	208	51	{60; 326}

as follows. The search continued until the number of the best found individual rose to 10% of the population or 6000 individuals were evolved, i.e. the best performing path, was found by the GA in all cases. For the single objective cases, the best individual found by GA was the same as the ones found by the available package [24]. For the multi-objective cases, the solution was confirmed through exhaustive search. Simple steps were added to the stage matrix procederes in order to carry out such an exhaustive search. In these steps, referring to Table 3.2 (columns 6 and 7), one first calculate the fitness value that corresponds to the separate optima of all objectives; this value corresponds to a hypothetical best fitness value. The procedure then calculates the fitness value that corresponds to the worst obtained solutions of all objectives (columns 8 and 9 of Table 3.2). This fitness value corresponds to a hypothetical worst fitness value. A range of several trial fitness values between these two extreme hypothetical values is thus established. Incremental fitness values are then attempted starting at the worst fitness value until a value is reached that is close to the best hypothetical fitness. Numerous trials can then be attempted near that fitness value. The increment value is decreased at the last verified fitness value and more fitness values are tried until one arrives at another fitness value that is not verifiable by the algorithm. The algorithm then backtracks to the last verifiable fitness value and decreases the increment value of the fitness values. The procedure is continued until a fitness value is reached that cannot be improved after substantial number of trial runs (roughly 200 trials); this fitness value is called the best compromise solution. The stopping criterion is thus chosen as the final fitness value that is discovered by the exhaustive search procedure. However, the convergence behavior study of the algorithm revealed that choosing the percentage of optimal solutions in a generation will yield the same results as the exaustive search for the stage matrix method.

A subset of the Pareto (compromise) solutions that have been generated by the algorithms is shown in Appendix D (Section D.1) . The input parameters to the model of these 10 examples is shown in Appendix D (Section D.2). For smaller networks, two methods outlined in [18, 11] were utilized in order to confirm the outcome of the methodology.

3.5.2 CPU Time Analysis

The trend of the average CPU time (under UNIX/SUN 4.1 Workstation) trend over 5000 individual evaluations, i.e. finding a single solution, for both the single objective case (minimization cost or maximizing quality) and the multi-objective case, finding the compromise solution, is shown in Figure 3.15.

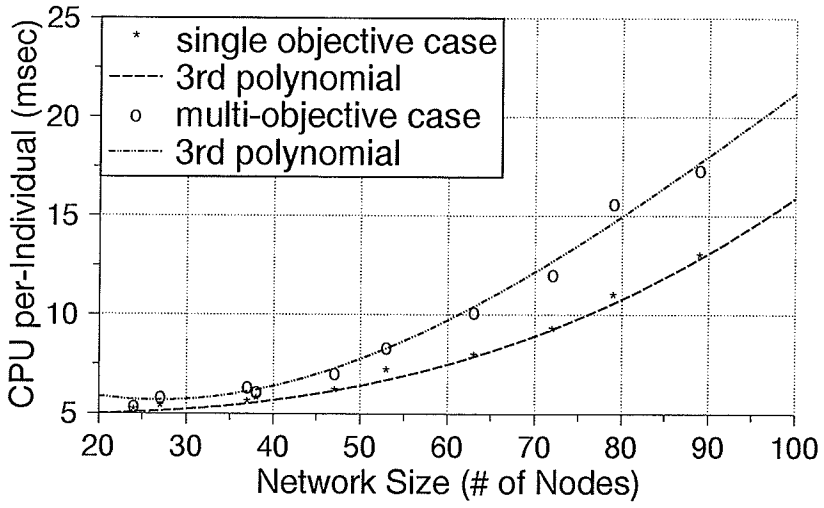


Figure 3.15: Growth rate per individual of stage matrix.

It is noticed that the algorithm progresses in a low polynomial time throughout the networks that have been experimented here. This is apparent for both cases, and is displayed in Figure 3.15 where a curve fitting procedure revealed a low polynomial trend for the CPU time required to find a single solution for the single objective case.

The multi-objective fitting curve (third polynomial) is displayed in the same figure; it is also of a low polynomial nature. The CPU time per individual for the multi-objective case is larger than that of the single objective case as depicted in Figure 3.15. This can be attributed to several factors: first, the storage space requirements for the extra objective(s) lead to an indirect increase in CPU time. Secondly, the number of objective functions (fitness functions) evaluations proportionally increases with the number of objectives, thus requiring more CPU time. Finally, there is an increase in the frequency of applying the modification procedure in any given generation for the multi-objective case. This is due to the added complexity of the extra objective. In other words, the number of superior solutions has been reduced causing an extra effort to modify more weak individuals than is observed for the single objective case. Thus, poorly performing individuals (inferior solutions) are causing the algorithm to apply the modification procedure more often than for the single objective case. This results in an increase in CPU time for the multi-objective case.

Referring to Figure 3.16 where the total CPU time which is required to find the optimal solution for the single objective cases, i.e. the termination time, is plotted versus the size of the networks (10 examples). Every generation consists of 100 individuals, that were allowed to evolve up to 20 generations; 24 different starting random generations (runs) were attempted. The growth rate of the algorithm (stage matrix) is, generally, of the increasing type as one moves from small network sizes to the large size ones. Figure 3.16, also depicts the average CPU time trend that is required for each example; a slow, rising, curve of this average is observed for the single objective case as the algorithm moves from the low size network end toward the high end of the networks.

Figure 3.17 shows the multi-objective extension of the same examples. The

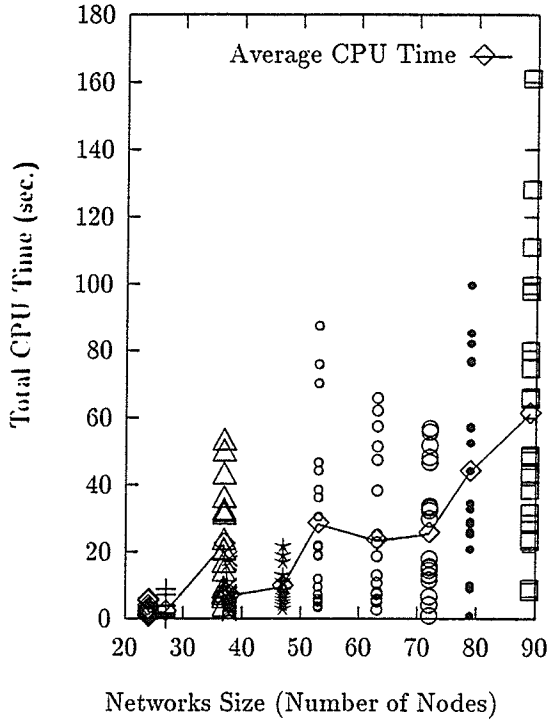


Figure 3.16: Total CPU of stage matrix (single objective).

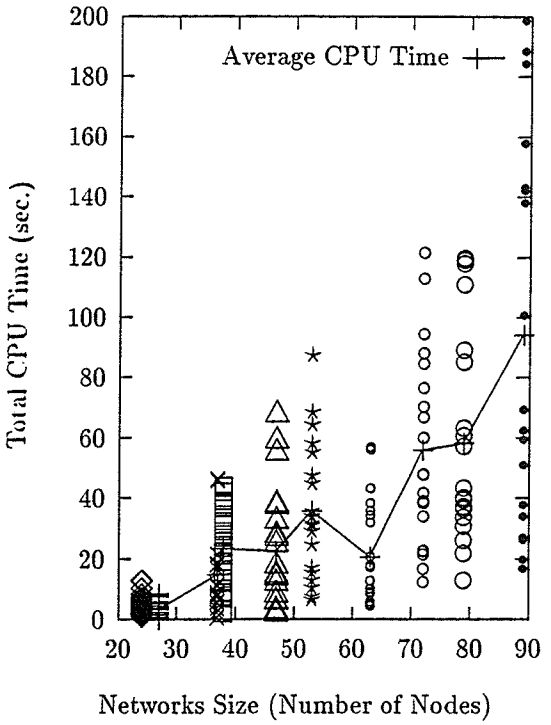


Figure 3.17: Total CPU of stage matrix (multi-objective).

algorithm was executed under the same conditions described earlier. A similar statement, such as the one already made for the single objective case, can be made here regarding the slow increasing trend of the total CPU time and the average CPU time trends of these examples; except one observes a general increase in their ranges. Figure 3.18 compares the average CPU time for both single and multi-objective cases. This difference in CPU time requirements is due to the increased complexity

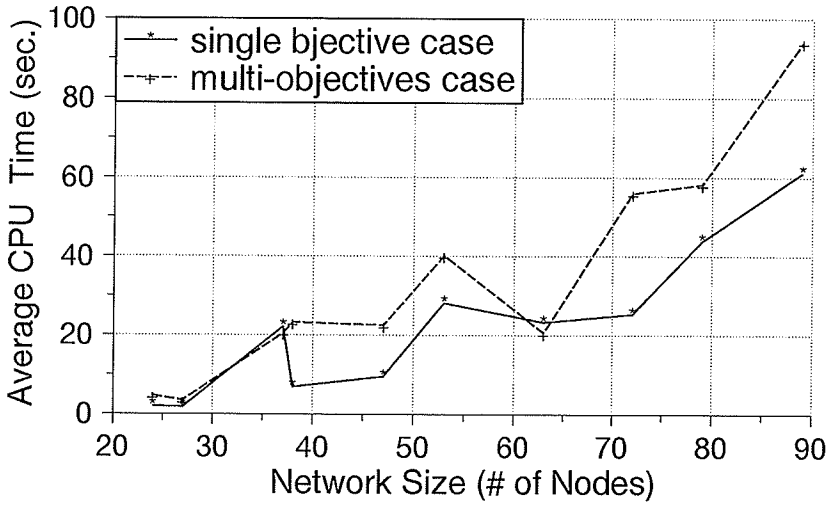


Figure 3.18: Growth rate for total CPU of stage matrix.

incurred by adding the second objective to the single objective optimization problem, as was described earlier, making the problem an NP-complete one [26].

Table 3.3 depicts the 10 examples with columns 1,2 and 3 as described earlier in Tables 3.2. The last two columns of Table 3.3 show the average CPU time (over 20 runs) for the single objective case. The last column of this table comprises three CPU time values that correspond to the minimum attainable time, the average attainable time and finally the maximum expected CPU time. The user can stop the algorithm at an earlier stage of the execution of the algorithm with an expected CPU time that corresponds to the minimum value depicted in the last column.

Naturally, the confidence level, that the obtained solution, is close to the optimal solution is lower for this case than for the other two cases, where the algorithm is allowed to run for longer times. A confident level of 90% for the average CPU time case was observed. The optimal solution, however, may be lost more often for the minimum CPU time case than for the other two cases.

Each of the last two columns corresponds to the respective, Out-of-Kilter and stage matrix optimization techniques. It is clear, from Table 3.3, that the Out-Of-Kilter technique is superior (for the average CPU time case) to the stage matrix technique for the large network cases.

The stopping criterion condition of the single objective case is now discussed. The algorithm is stopped after finding the best solution. Referring to the last column of Table 3.3, this column depicts three CPU time values that correspond to: a) the minimum time required to find the best solution, b) the CPU time that is expected to occur on the average and finally c) the maximum CPU time that is required to find the best solution. All these CPU time values correspond to various trial runs where the best solution was found by the algorithm. Consider the last example in the table. It reveals an expected minimum number of generations to find the best solution at ≈ 14 (out of possible 600) generations, i.e. the algorithm can be stopped as soon as this amount of generations is evolved. The algorithm is to stop at ≈ 93 generations with the average CPU time shown in that column. The average CPU times trend of all examples was used in the complexity analysis of the algorithm (next section). The last value of the CPU time reveals the worst (maximum) number of generations required to stop the algorithm ≈ 240 generations. These expected generation numbers were found as was described earlier for the single objective case.

The multi-objective case was handled by using the stage matrix only, since the Out-of-Kilter method cannot handle it. The CPU time results of this method are

Table 3.3: Total CPU time for single objective case.

#	Networks Size	Number of nodes	Optimum Cost	CPU Time for Out-Of-Kilter *Av. (sec.)*	CPU Time (sec.) for Stage Matrix *Min.*	*Av.*	*Max.*
1	7	24	27	3.0	≈ 0	2.01	6
2	7	27	28	2.85	0.6	1.82	7
3	8	38	24	4.31	0.7	22.12	57
4	9	37	11	4.25	0.6	6.85	22
5	10	47	19	4.48	2	9.52	24
6	11	53	28	4.58	2.5	28.16	87
7	12	63	33	4.69	2	23.27	62
8	13	72	35	5.08	1	25.32	58
9	14	79	32	5.19	0.7	43.90	100
10	15	89	34	5.35	9	61.23	160

shown in the last two columns of Table 3.4, where the first four columns are described earlier (for Tables 3.2). The last column consists of a three CPU time cases with the same implications of the previous table for the single objective case.

The stopping criterion (generation numbers) can be similarly obtained for the multi-objective case. The CPU times is shown in the last column of Table 3.4. Similar inspection of Figure 3.15 for the last example revealed the following expected generation number, the minimum required gneration to stop the algorithm is $\approx$ 19, the maximum number of genrations to stop the algorithm is $\approx$ 196 and the recommended number of generation of the average case is (92 out of possible 600 generations.

3.5.3 Computational Complexity Analysis

Complexity analysis of stochastic models is extremely difficult. Many researchers have used to a CPU time analysis of such models [48, 49]. In the last section, a CPU time study for analyzing the growth rate of stage matrix-GA based method was presented. In this section an attempt to quantify such complexity analytically is carried out. This analysis gives an approximate indication of how well the model that is transformed into a set of computer instructions (algorithm) will perform before the actual implementation of the model (in the form of a computer program). A well-developed terminology of computational complexity has been established over the last few decades [48, 49]. The conventions presented in [48] are adopted here; the notation of big O is used to indicate the worst time required to execute a particular set of instructions. This gives an upper limit to the worst expected performance of the entire algorithm.

A GA–based technique usually comprises of some determinestic procedures and some stochastic procedures. These stochastic procedures makes the computational

Table 3.4: Total CPU time for multi-objective case.

#	Networks Size	Number of Nodes	Compromise Solutions $\{Cost; Quality\}$	CPU Time (sec.) for Stage Matrix		
				Min.	*Av.*	*Max.*
1	7	24	$\{36; 117\}$	≈ 0	4.67	17
2	7	27	$\{37; 182\}$	0.2	3.43	12
3	8	38	$\{29; 197\}$	0.5	14.76	45
4	9	37	$\{56; 209\}$	0.5	23.26	44
5	10	47	$\{59; 249\}$	0.6	22.52	67
6	11	53	$\{61; 259\}$	4	35.89	84
7	12	63	$\{67; 295\}$	3	20.62	58
8	13	72	$\{58; 309\}$	9	55.93	121
9	14	79	$\{66; 326\}$	10	58.22	120
10	15	89	$\{60; 348\}$	19	94.27	200

complexity analysis of GA–based techniques difficult to achieved (if not impossible). However, a computational complexity analysis of the determinestic procedures within such techniques gives a useful insight as to the expected growth rate of the entire technique when proper upper bounds of the stochastic procedures are included. Furthermore, such analysis allows for further improvement of the determinestic procedures if possible.

A detailed complexity analysis is presented in Appendix E. The analysis is first concerned with the application of the modification procedure to modify a single individual path. The analysis is then extended to the entire population of individuals and for the span of several generations of these populations.

A detailed description of the modification procedure has been carried out in Section 3.3.2 of this Chapter. The computational complexity of the matrix multiplications and the rest of the modification procedure is derived in Appendix E. The final outcome of the complexity analysis of that procedure is as follows:

$$\simeq (n-2)^2/k = C(n-2)^2 \tag{3.4}$$

where n is the total number of nodes in a network and k is the number of stages. Equation (3.4) gives an estimate of the maximum number of expected computational operations for any type of modification operations described in this chapter including the identification procedure where all stage matrices are multiplied together. The constant C is related to the number of stages. The value expressed by Equation (3.4) gives the worst computational time that is expected from the application of the modification procedure for a single individual network. It is obvious that in big O terminology this computational time is expressed as $O(Cn^2)$ time unit. This means that the procedure has a growth rate that is related to the square number of the network's nodes. The growth rate of any procedure is usually measured by the

most dominant computational operation in any given algorithm [48]. The dominant operation in Equation (3.4) is the n^2 term. In other words, the growth rate is related to the number of links in any given network. This is obvious, since the number of links is roughly equal to n^2, for the type of the networks that are modeled using the stage matrix method.

The application of the modification procedure is related to the status of the generated, or evolved, network paths. If the network (individual) has a single path then there is no need for applying this procedure. Otherwise, the procedure is applied until a single path per-individual network is obtained. The latter case occurs in a stochastic fashion. The stochastic nature of this procedure makes an accurate growth rate of the algorithm an impossible task. However, the above estimate, shown in Equation (3.4), along with a CPU time study of the convergence of the algorithm, would give a reasonable description of the modification procedure growth rate.

There is no other operation within the modification or GA operations that have a growth rate larger than $O((k-2)n^4)$ (see Appendix E). This growth rate takes into consideration GA input data such as the size of population, the number of generations and the number of trial runs. Therefore, this growth rate is the final expected value that describes the computational complexity of the entire computer package. The worst growth rate is that of a fourth order polynomial in relation to the size network (n) and is also directly related to the number of stages (k) of the network. The actual growth rate observed in the last section was, at worst, a the third order polynomial (see Figures 3.15, 3.18).

3.6 Summary

The formulation of the process plans as flow networks was presented. The stage matrix method that facilitates the employment of a genetic algorithm to find the best process plan among a large set of alternative process plans was discussed. The formulation includes both single and multiple objective criteria for process planning selection problems.

The behavior of the stage matrix method was demonstrated for various examples and demonstrated a rather high CPU time trends for large networks. This is due to the fact that the stage matrix method computational growth rate is directly related to the number of arcs in the network. The extra links property of most individuals (networks) have offered a rich pool of traits that can be inherited by subsequent generations of networks causing a good convergence rate to an optimal solution. This property, however, contributed to a substantially increased number of multiple paths that must be modified, thus increasing the required CPU time in reaching an optimal solution.

A CPU time study showed a similar behavior, i.e. the predicted growth rate of the algorithm which is carried out earlier using big O notations, for both the single and multiple objective cases. This is an important property that offers a smooth and natural transition of the algorithm from handling single objective networks toward handling multi-objective networks, i.e. the growth rate does not change, on average, for both cases.

The complexity analysis of this method showed that it is on the order of a fourth degree polynomial in relation to the number of nodes of a given network. However, the actual CPU time from simulations indicates a lower growth rate (on average) of the algorithm.

As was mentioned earlier, GA's are inherently parallel search algorithms. Therefore the computational speed can be enhanced using parallel techniques [51, 52, 53, 54]. In fact in order to realize their full potential, GA's must be implemented on parallel computer architectures [35]. This property can be used to enhance the speed of the computations. For example, a similar study [37] showed that by dividing the population of 100 amongst 16 transputers, a speed-up factor of 12 can be achieved with the same performance.

Based on the observations and the methodology developed in this chapter, a new approach based on a set theoretic approach is presented in the next chapter. The approach is based on the notion that nodes can be represented in binary form instead of arcs. This method, as will be demonstrated through a number of examples, will address most of the drawbacks that have been observed in the stage matrix formulation.

CHAPTER 4

Stage Set Formulation

In the previous chapter, we discussed the stage matrix approach for modeling the process planning selection problem. In this chapter we discuss an alternative method based on set theory, i.e. the"stage set" nodal representation of paths in a network. This method facilitates the application of GA to handle PPSP more efficiently than the stage matrix approach. The stage set method can, like the stage matrix method, handle single objective and multiple objective problems as well as Constraint Shortest Path Problems (CSPPs).

4.1 Flow Networks Binary Formulation

In this approach, the nodes of a flow network that represent similar operations form a set. An unavailable node in a set is given a zero value. A one-valued node in a set indicates the availability of the node and possibly all paths linked into that node. For example, consider the flow network displayed in Figure 4.1. This network is based on the general network shown in Figure 3.1. The sets are depicted in Figure 4.1 under the graphical representation of the network. The first set represents the two output nodes of the first stage, signifying two machines. The second stage set has three output nodes which implies that three alternatives originating from the previous two machines and so on.

The activated nodes of every set are highlighted in Figure 4.1; the first and the last nodes of the network are always activated. A "one" entry in the set signifies

an active node that is selected from that stage set for possible connection with the previous and the following set of nodes in the network. It indicates that an operation can be chosen with that particular node. A "zero" entry indicates that that node of the network is not available; therefore, any path containing this node is not available as a solution.

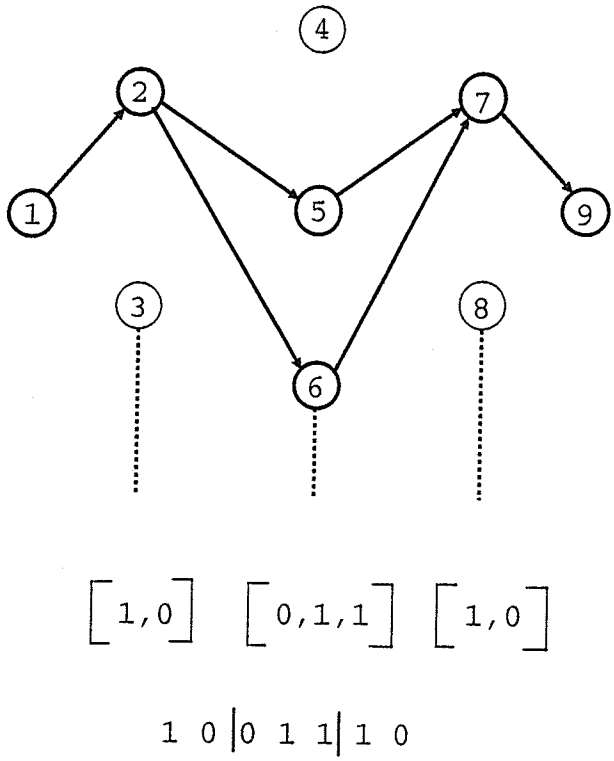


Figure 4.1: Multiple path flow network.

The above set notation can, alternatively, be represented in a string format by concatenating all the set members of the stages as shown Figure 4.1. This type of format allows the genetic algorithm operators to be readily implemented. Referring to Figure 4.1, it is also observed that the individual flow network indicates two alternative paths, shown in solid lines, namely: $1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9$ and $1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 9$. This individual must be transformed into a single path flow network before being evaluated by the fitness evaluator. A modification procedure

must therefore be applied to insure the existence of a single path per network. This procedure imposes a condition whereby every stage set of nodes in an individual network must possess a single activated member node, i.e. having a "one" value in the set. After the application of this procedure, a single path is obtained as shown in Figure 4.2. This figure shows the graphic representation of that single path. The set representation as well as the string representation of this single path network is shown in Figure 4.2.

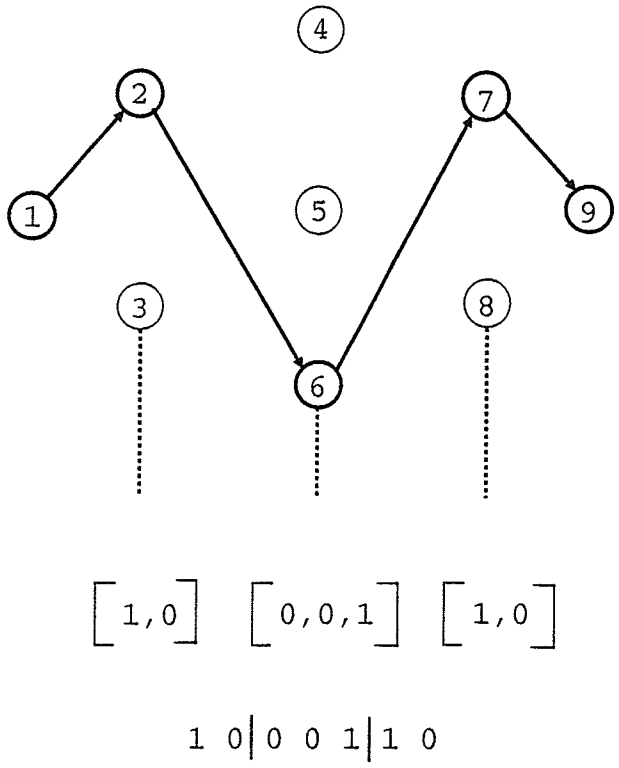


Figure 4.2: Single path flow network.

Using the stage set formulation, it is observed that the string size is substantially reduced when compared to that of the stage matrix method. It is further noticed that the modification operation is simpler, and thus requireing less computational effort than the stage matrix method. The stage set method, however, does not allow for extra links in the representation of an individual network. This trait implies lower

modification requirements than that of the stage matrix method.

4.2 Implementation

The software package was already described in detail in chapter 3. In this chapter, we only highlight the different procedures that are relevant to the stage set method. As described earlier, a path consists of consecutive nodes (one per set) from the source node to the sink node. Every set that corresponds to output nodes of a stage in the network must have a single node member that is activated. This is easily achieved by assigning a zero value to all but one set member, for all sets of the network. Every selected individual is initially ensured to possess a single path by applying this modification operation. The crux of the GA based modification procedure and GA operations is shown here:

*Step*1 Randomly generate a population of size p, Each individual in the population is of a size that is equal to the number of nodes in a network, excluding the source and sink nodes,

*Step*2 Map each string entry to the corresponding member of the set in the network,

*Step*3 Every set must have one member of status "true (one)" and status "false (zero)" for the remaining members,

*Step*4 For all population members, apply the fitness function and the GA operations to create the second generation of individuals,

*Step*5 If the stopping criterion is not satisfied, go to *Step* 1, or else terminate. The stopping criterion employed here is the number of evolved generations.

After the application of the cross-over operator, some individuals may again consist of either multiple paths or one complete path. The first case corresponds to a situation with more than one activated node per set in one or more sets. The second, corresponds to having every set in the network with a single member node that is activated . The modification procedure is applied, for the first case, in order to satisfy the single path per individual condition.

The mutation operation is not explicitly applied in this approach. However, an implicit application of this operation is carried out through the application of the modification operation. This mutation is slightly different from the stage matrix approach. The difference stems from the fact that the modification operation, in the stage set approach, has to deal only with multiple paths situation; unlike the case for the stage matrix method where some individuals may possess no complete path as well as other individuals that may comprise multiple paths. The implicit application of the mutation operator is carried out within the modification procedure. Since the frequency of applying the modification operation is lower than that of the stage matrix method, it follows that the probability of mutation operation is lower than that of the stage matrix case.

The stage set method has an advantage, in terms of implementation, over the stage matrix method due to the availability of efficient set theoretic operations. Some of these operations have already been implemented in the current software. All possible and relevant set theoretic operations can be added to this implementation. This will substantially enhance the speed and storage requirements of the software for large network problems. The computational superiority of the stage set procedure over the stage matrix method is demonstrated next.

4.3 Experiments

4.3.1 Convergence Behavior

Consider the same routing network which was discussed in Chapter 3. This network is shown in Figure 4.3. It is used here to analyze the performance of the stage set based method. The objective is to find the least cost process plan amongst all possible paths in the network. The cost values are shown as integer values along every arc. Note that the network comprises of 7 stages, 24 nodes, 80 arcs with a total number of 1440 possible paths. The algorithm is analyzed here in terms of convergence

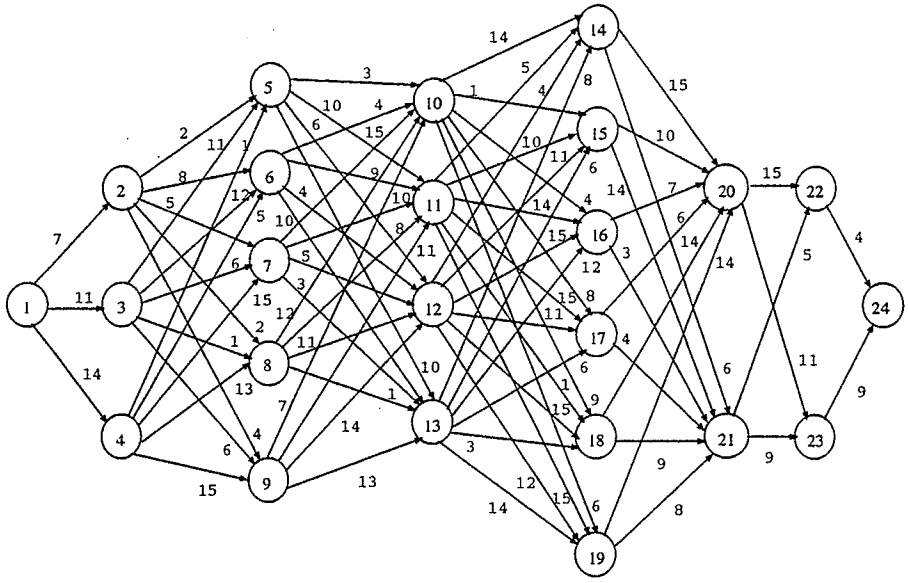


Figure 4.3: Typical single objective manufacturing flow network.

to the best solution. A generation of 100 individuals (solution points) were first generated randomly by GA. They were evolved over 30 generations. Twenty trial runs were attempted. Every trial run represents a different starting generation of potentially different regions of the solution space. A cross-over probability $P_c = 1.0$ was used.

A sample of the convergence behavior of the algorithm is shown in Figure 4.4 where the initial trial run, the twelfth and the seventeenth trial runs (a, b and. c respectively) are plotted. In Figure 4.4, the fitness values of each individual in a generation (the mapped objective values) are plotted for all evolved generations. A general trend of improvement of the convergence to a better performing generation is observed. That is, the fitness values of every succeeding generation improve steadily as the algorithm proceeds from one generation to the next. The optimal solution was preserved through the evoluation process of the initial generation and throughout the trial runs of the algorithm. As can be seen in these figures, the trend is not continually of an increasing nature. It can also be seen, e.g. from Figure 4.4 (b and c), that the algorithm can be stopped at an earlier generation than the one selected for this demonstration, e.g. the third or fourth generations. The optimal solution was the path that had a minimum cost of 28 or a fitness value of 64, and it was: $1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$. This is the same path that was found when the stage matrix method is applied to the same network. This result was also confirmed with an available commercial package [24].

An interesting metric of the convergence behavior of the algorithm is the average fitness value of each generation; this is depicted in Figure 4.5. The figure shows an increasingly improving trend of the average of fitness values at each generation. Figure 4.5 also shows little improvement after the $12th$ generation. This trend of convergence behavior may be used as one factor in determining the stopping criterion for future applications of the algorithm.

The trends of the average of the fitness values for both the stage matrix and the stage set methods are depicted in Figure 4.6. This figure also depicts the trend of the stage set method where a population size of 200 is used. It can be seen from Figure 4.6 that the trend of the stage matrix convergence to an optimal solution is

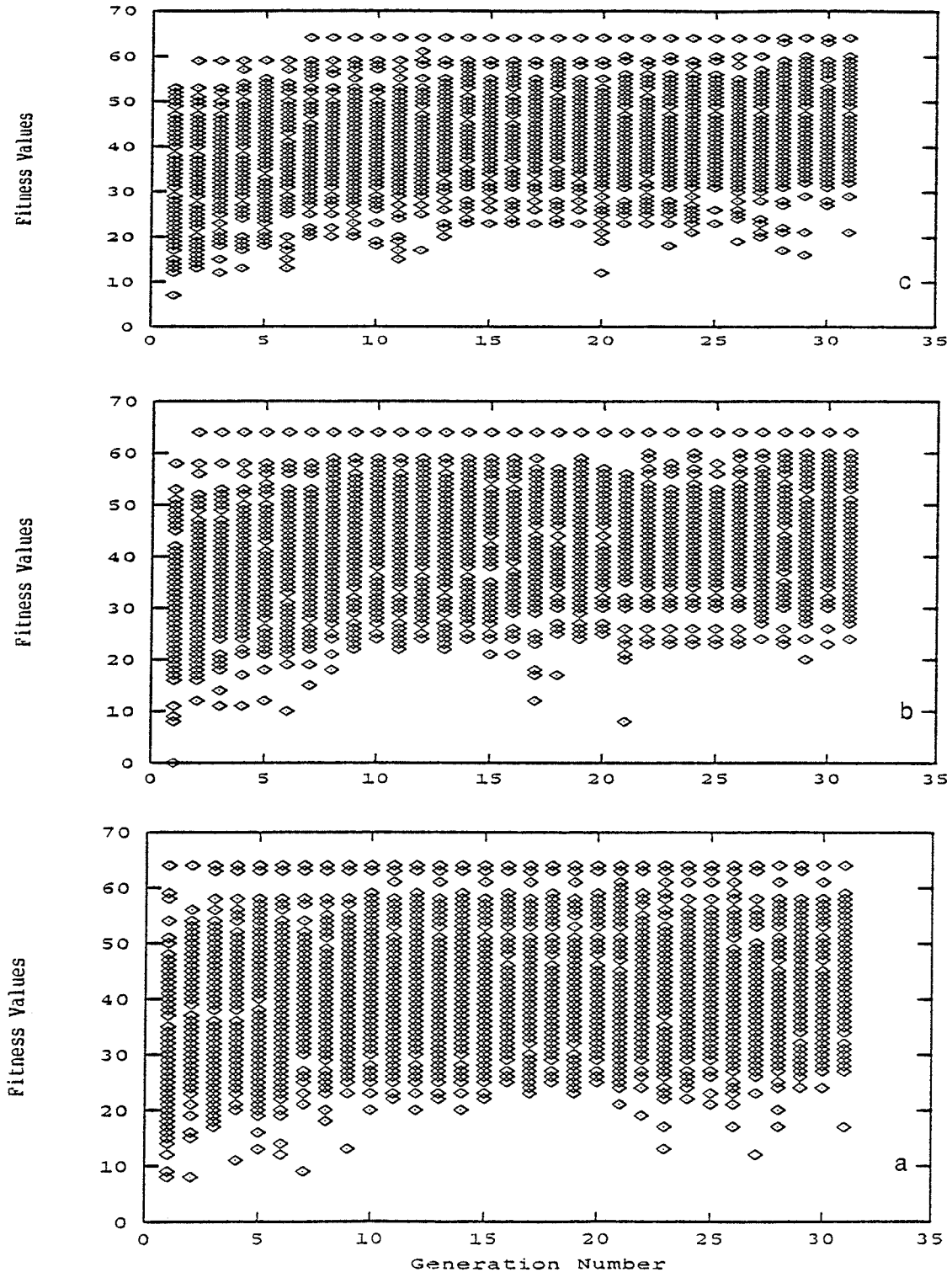


Figure 4.4: Convergence behavior; typical trial runs.

slightly better than that of stage set method trends at higher generations. This is

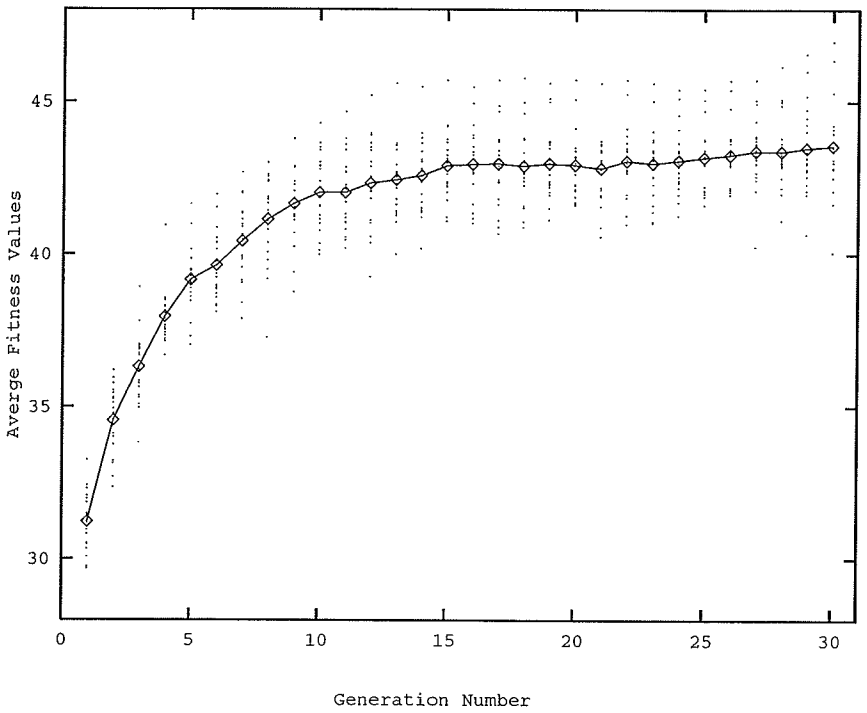


Figure 4.5: The average trend of fitness values (popsize=100); stage set method.

obvious from noticing the steady improvement of the convergent trend for the stage matrix method. This trait may be attributed to the extra links property which is characteristic of the stage matrix method. Having extra links in each individual adds richness to the search procedure. However, these extra links render the stage matrix method more stochastic than the stage set method which is observed (see Figure 4.6) from the smoother nature of the average trends of the stage set method compared to the stage matrix method. The steady improving convergence trend of the stage matrix method is realized only with a substantial increase in computational time. Referring to Figure 4.6, the trends of convergence reveal that the stage set method can be operated at, relatively, low generation size with a better convergence rate than the stage matrix method. Furthermore, the figure shows that an increase in the population size (from 100 to 200) for the stage set method yields a better

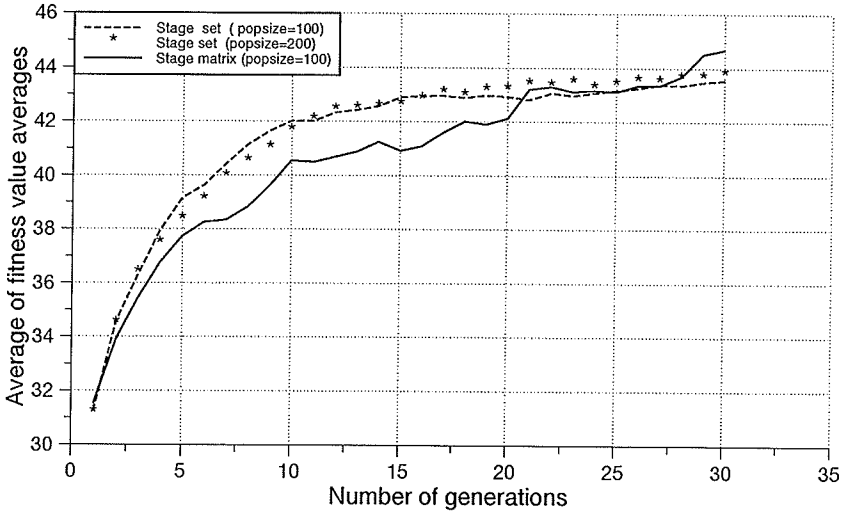


Figure 4.6: Average trend of fitness value averages.

convergence trend than the lower population size case (100). Although this increase is small, it has yielded an earlier convergence to an optimal solution for the 200 population size case. The convergence trend of the stage set method of a population size of 200 is shown in Figure 4.7. The levelling-off of the latter case implies that the stage set method can be analyzed at low generation sizes (e.g. between $15-30$). This stablization of the improvement trend of the averages of fitness values suggests that the algorithm can be stopped at any generation in the range of $15-30$ generations (eg. the 17th generation can be used as the stopping criterion of the algorithm).

Figure 4.8 shows the general trend of the average fitness values plotted versus the number of generations over 50 trial runs.

4.3.2 CPU Time Analysis

In this section, a computational time study of ten flow networks is presented; these are the same examples used for the stage matrix approach (Section 3.5.2) in order to

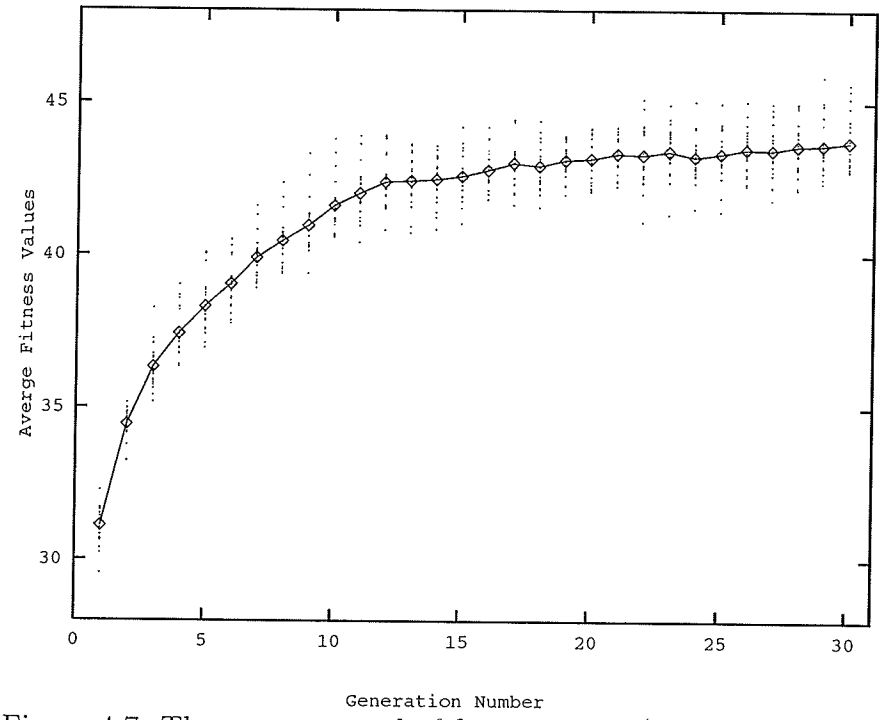


Figure 4.7: The average trend of fitness values (popsize=200).

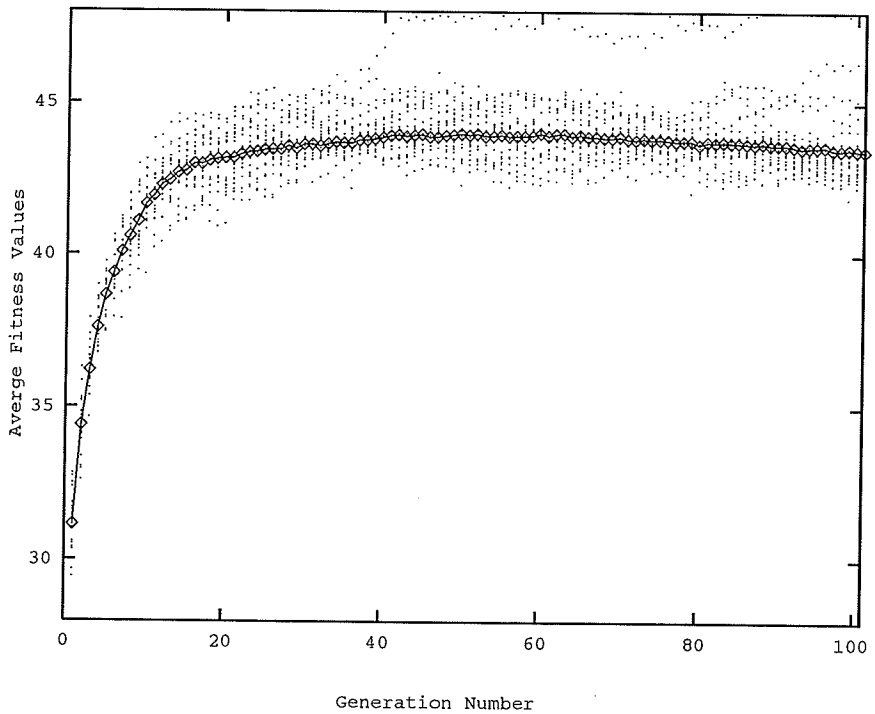


Figure 4.8: The average trend of fitness values (general trend).

allow for a comparison of the two methods. These network are first analyzed under a single objective condition (minimizing the cost). A second objective criterion is then added to the networks, e.g. quality. For the single objective case, the method is also compared with the Out-of-Kilter procedure.

Table 4.1 depicts the 10 examples. A population size of 200, 20 generations, 20 different runs (different starting generations) and a probability of crossover of 1 were selected for all these examples. The choice of a high crossover value is taken due to the fact that the expected mutation operation value for the stage set method is low as was described in the previous section; a good performance is associated with either a high crossover rate combined with a low mutation rate, or a low crossover rate combined with a high mutation rate [50]. The choices of the number of generations and the population size were selected on the basis of the convergence analysis that was conducted in the previous section.

The results are shown in Table 4.1 where columns 1, 2, 3 and 4 have the same meaning as that of Table 3.2. Columns 5, 6 and 7 of Table 4.1 show the average CPU time for the single objective case. Each column corresponds to the respective optimization technique: Out-of-Kilter, stage matrix and stage set techniques. It is clear from Table 4.1, that the stage set technique is superior to both of the other two techniques. The algorithm has stopped after finding the best solution. The last two columns of Table 4.1, depict three CPU time values, corresponding to the minimum time required to find the best solution, the average expected time and finally the maximum CPU time. These CPU time values correspond to various trial runs where the best solution was found by the algorithm. Consider the last example in the table. It suggest the expected minimum number of generations required to find the best solution to be ≈ 7 evolved generations, i.e. the algorithm can be stopped as soon as this number of generations is evolved. The algorithm stops, on average, at ≈ 28

Table 4.1: Total CPU time for single objective case.

#	Network Size	No. of nodes	Optimum Cost	CPU Time for Out-Of-Kilter *Av. (sec.)*	CPU Time (sec.) for Stage Matrix *Min.*	*Av.*	*Max.*	CPU Time (sec.) for Stage Set *Min.*	*Av.*	*Max.*
1	7	24	27	3.0	≈ 0	2.01	6	≈ 0	0.087	0.4
2	7	27	28	2.85	0.6	1.82	7	≈ 0	0.048	0.25
3	8	38	24	4.31	0.7	22.12	57	≈ 0	0.404	1.00
4	9	37	11	4.25	0.6	6.85	22	0.1	0.575	0.92
5	10	47	19	4.48	2	9.52	24	0.1	0.637	1.20
6	11	53	28	4.58	2.5	28.16	87	0.2	0.758	1.35
7	12	63	33	4.69	2	23.27	62	0.15	0.36	0.75
8	13	72	35	5.08	1	25.32	58	1.8	0.3	1.42
9	14	79	32	5.19	0.7	43.90	100	0.5	1.755	3.25
10	15	89	34	5.35	9	61.23	160	0.4	1.53	3.75

generations giving the average CPU time shown. The average CPU times trend of all examples was used in the complexity analysis of the algorithm. The last value of the CPU time reveals the worst (maximum) number of generations required to stop the algorithm is ≈ 68 generations. These expected generation numbers were found by inspecting the CPU times per single individual for the tenth example in Figure 4.9. The total CPU times shown in the last column of Table 4.1 were divided by the CPU per individual and the population size.

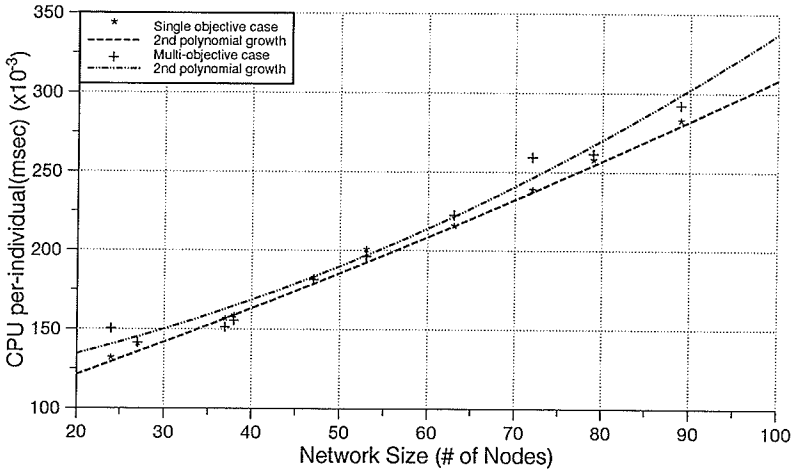


Figure 4.9: Growth rate per individual of the stage set method.

The number of expected generations that can be used to stop the stage matrix algorithm is restated here in order to compare the two methods. The best solution was found at the minimum, average and maximum number of evolved generations of ≈ 14, ≈ 93 and ≈ 240 respectively. These expected evolved generations reveal that the stage set method compares favorably (in terms of convergence) to the stage matrix method.

The multi-objective case was handled by using only the stage matrix, described in the last chapter and the stage set approach, since the Out-of-Kilter method cannot

Table 4.2: Total CPU time for multi-objective case.

#	Networks Size	Number of Nodes	Compromise Solutions $\{Cost; Quality\}$	CPU Time (sec.) for Stage Matrix			CPU Time (sec.) for Stage Set		
				Min.	*Av.*	*Max.*	*Min.*	*Av.*	*Max.*
1	7	24	$\{36; 117\}$	≈ 0	4.67	17	≈ 0	0.292	1.9
2	7	27	$\{37; 182\}$	0.2	3.43	12	≈ 0	0.154	0.7
3	8	38	$\{29; 197\}$	0.5	14.76	45	≈ 0	1.892	3.95
4	9	37	$\{56; 209\}$	0.5	23.26	44	0.1	1.080	4.9
5	10	47	$\{59; 249\}$	0.6	22.52	67	0.1	4.818	9.9
6	11	53	$\{61; 259\}$	4	35.89	84	1.5	4.007	8.9
7	12	63	$\{67; 295\}$	3	20.62	58	0.1	1.818	6.1
8	13	72	$\{58; 309\}$	9	55.93	121	1.8	5.965	14
9	14	79	$\{66; 326\}$	10	58.22	120	0.3	3.144	11
10	15	89	$\{60; 348\}$	19	94.27	200	1	9.607	26

handle it. A subset of the Pareto (compromise) solutions that have been generated by the algorithm (using both methods) is shown in Appendix D.

The CPU time results of these methods are shown in the last two columns of Table 4.2, where the first four columns are described earlier (for Table 3.2. The population size was chosen as 200, the number of evolved generations was selected as 20, the number of starting generations was 20 and the probability of crossover was 1. The input parameters to the model for these 10 examples are shown in

Appendix D. Table 4.2 reveals that the stage set approach demonstrates a superior performance over the stage matrix for the 10 network examples.

The program has stopped after finding the best compromise solution. Referring to the last two columns of Table 4.2, each of these columns depicts three CPU time values that correspond to the minimum time required to find the compromise solution, the CPU time expected to occur on the average and finally, the maximum CPU time required to find the compromise solution. All these CPU time values correspond to various trial runs where the compromise solution was found by the algorithm. Consider the last example in the table, the last column shown in the table reveals an expected minimum number of generations to find the compromise solution at $\approx$ 17 generations, i.e. the algorithm can be stopped as soon as this number of generations is evolved. The algorithm is to stop at $\approx$ 160 generations with the average CPU time shown in that column. The average CPU times trend of all examples was used in the complexity analysis of the algorithm. The last value of the CPU time reveals the worst (maximum) number of generations required to stop the algorithm $\approx$ 433 generations. The stage matrix expected generations that can be used to stop the algorithm is restated here in order to compare the two methods. The best solution was found at minimum, average and maximum number of evolved generations of $\approx$ 19, $\approx$ 92 and $\approx$ 196 respectively. Even though these generation numbers compare favorably (in terms of the average CPU times for convergence) for the stage matrix method over the stage set method, their corresponding CPU times are substantially worse than the stage set method.

As was the case for the stage matrix method, the CPU time trend for finding a single solution point for both single and multi-objective cases is plotted in Figure 4.9 versus the size of the networks. The multi-objective trend is slightly higher than that of the single objective one. This indicates a marked improvement over the stage

matrix method. The polynomial behavior of both of these cases is favorable since a low polynomial growth is observed as shown in Figure 4.9. Both trends of the single objective and the multi-objective cases are of second polynomial growth. The single objective trend is slightly better than the multi-objective growth trend. In order to appreciate the CPU time growth trend of both stage set and stage matrix methods, they are illustrated together in Figure 4.10. The CPU time, for a single individual (solution point), trend for both single and munti-objective cases, is plotted versus the size of the networks; these cases are handled by the stage set method and are compared to the stage matrix method. A favorable trend of the stage set method over the stage matrix method is observed.

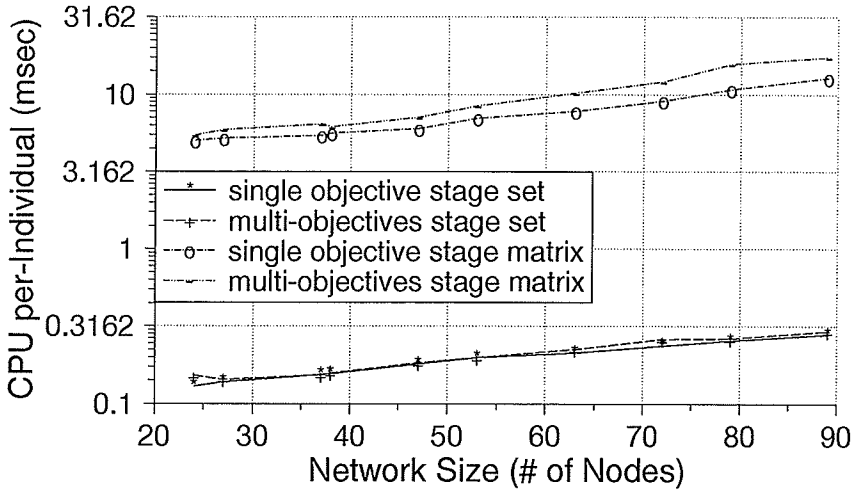


Figure 4.10: Stage set and stage matrix growth rate.

Figure 4.11 depicts the total CPU time versus the size of the networks for the single objective case. A trend generally similar to that of the stage matrix method is observed. However, this trend is at a much lower growth rate level than the earlier method. The average CPU time trend (as solid line) is also shown in that figure; a smoother (low polynomial) increase trend than for the stage matrix method is

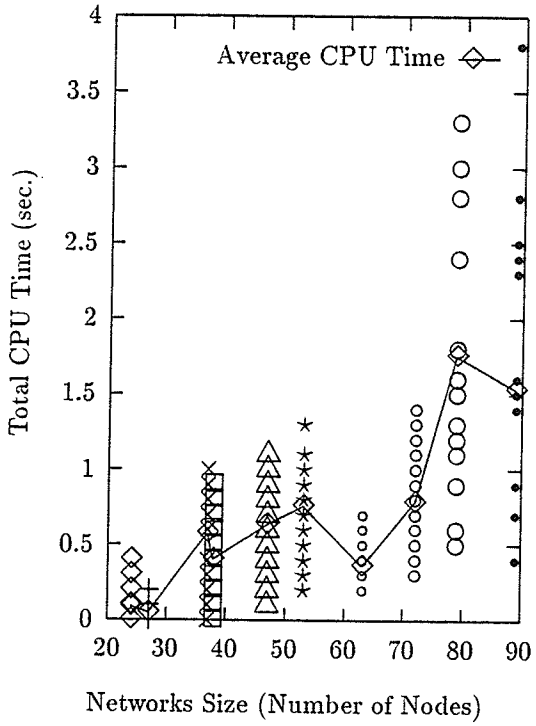


Figure 4.11: Total CPU of stage set (single objective).

observed.

Figure 4.12 shows the CPU time for the multi-objective case. A higher trend of the total CPU time and the averages than that of the single objective case is also noticed here though at much lower scale than that of the earlier method. Both of the average CPU time trends of the single and multi-objective for the stage set approach are shown in Figure 4.13. A steeper trend is observed for the multi-objective case than that of the single objective case. This difference is due to the added complexity of the second objective to the single objective optimization problem which affects the convergence of the algorithm to the optimal solutions.

The superiority of the stage set approach over the stage matrix approach stems from the fact that the stage set method represents the nodes of the network in a binary string, whereas, the stage matrix approach represents the arcs of the networks in a binary string. Obviously, the growth rate of the algorithm in any technique that

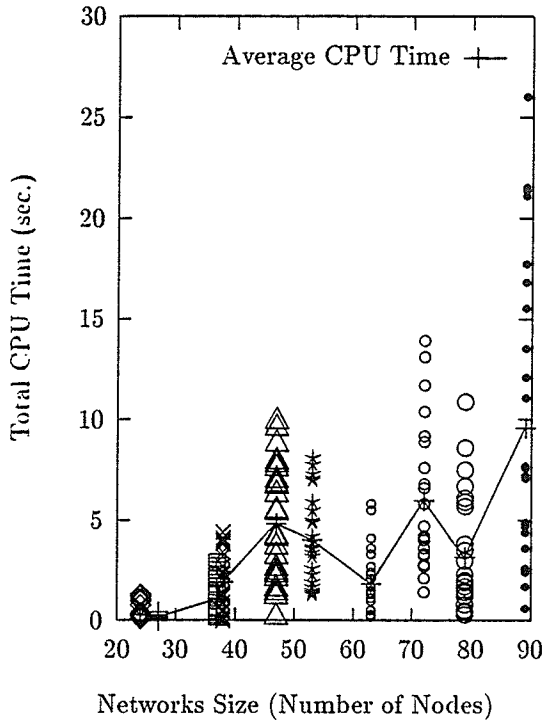


Figure 4.12: Total CPU of stage set (multi-objective).

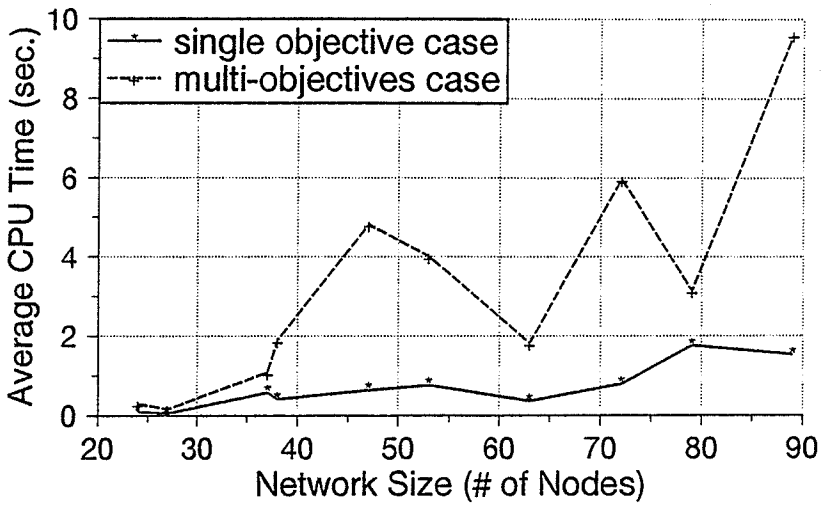


Figure 4.13: Growth rate for total CPU of stage set.

has such a representation, i.e. proportional to network's arcs, is much higher than the nodal one which is proportional to network's nodes. There are typically many more arcs than nodes in any given network: approximately n^2 of arcs correspond to n nodes of the same network; thus, the string size in the stage matrix approach is much larger than that of the stage set approach. This is also demonstrated by the various comparisons of the stage set growth rate (CPU time growth) with the Out-of-Kilter procedure, which has a growth rate related to the number of arcs in a given network [22].

The stage set procedure is basically a nodal based approach for representing flow networks, thus exhibit much lower CPU time growth rate than those procedures which are based on arcs representation of the network. It is, therefore, adopted as the prime procedure for the remainder of this work. Further speeding-up of the stage set procedures is possible by using parallel computer architecture in the same manner discussed in the last chapter. More CPU time analysis of the stage set approach, for some other examples, will be carried out later in this chapter.

4.3.3 Computational Complexity

The detailed computational complexity analysis is carried out in Appendix E. The final expected growth rate of our computational analysis of the entire stage set and GA procedures is $O(npG)$ time units; where n is the total number of nodes in a network, p is the population size and G is the number of evolved generations. This growth rate can rarely approach n^3 as was demonstrated in the CPU time study in the previous section (a second polynomial growth in n). Therefore, this growth rate can be maintained at a much lower rate, some fraction over the square of n. It was shown, in the previous section, that this growth rate is better than that of the classical technique based on the Out-Of-Kilter procedure, called netflow

procedure [24]. However, it is inferior to the Dijekstra's algorithm for the single objective shortest path problem only, since the latter has a growth rate of $O(n^2)$ [22]. Dijekstra's algorithm, however, is not applicable for multi-objective cases [26]. The growth rate per individual, described in the previous section, is actually of a linear type in relation to the size of the network. The second polynomial growth, observed there, is due to the computational overhead incurred by GA procedures that was not accounted for the complexity analysis of this section as well as the overhead incurred due to the emulation of the set theoretic operations in the current implementation of the software.

Both approaches presented in this thesis can be easily extended to handle multi-objective network problems with a modest increase in the computational complexity, due to the addition of the extra objectives.

4.4 Case Studies

An analysis of a bi-criteria problem is first carried out. Several examples of networks with varying degrees of complexity, i.e. increasing number of objectives are then presented and comparatively studied. Furthermore, a case study of the Constraint Shortest Path Problem (CSPP) is carried out. Finally, a network representation of multiprocessing, for a single or multiple parts is outlined with several examples of various size networks.

4.4.1 Bi-Criteria Case Study

The formulation of the bi-criteria case accompanied with a simple example is shown in Appendix B. A larger example flow network of two conflicting objectives to be minimized is introduced in this section. The network comprises 128 nodes, 1160 arcs and 14 stages. This network represents 1154 alternative solutions (paths). The

input nodes to each stage are as follows: 1, 2, 12, 16, 14, 12, 10, 4, 20, 4, 18, 6, 6, 2. In order to reflect a conflict between the first and the second objectives, the values of the two objectives were generated as follows: the first objective values (representing costs) were first randomly generated (a range of number between 1-15 was chosen); the second objective is then generated by subtracting the largest value of the first objective at each arc (plus one) from the first objective values at that stage, this is done throughout the network stages. on set of these objective values is listed below for each arc in the network:

2 4 5 11

Each arc input (the first two entries of the rows, e.g. $2 \rightarrow 4$) and the corresponding two objective values (e.g. 5 and 11 for that entry) are shown above. The remaining of these entries are included in a computer disc.

A sample of the result of applying Equation (B.1) of Appendix B to the various paths of this network example, is shown below. The first row has two entries. The first entry denotes the fitness value of each individual path, this value is useful in ranking the various selected paths according to their performance). The second entry of that row indicates the compromise solutions of a particular path. The second row shows the actual sequence of the path:

402.504547 {99.000000; 93.000000}

$1 \rightarrow 2 \rightarrow 5 \rightarrow 22 \rightarrow 34 \rightarrow 50 \rightarrow 62 \rightarrow 68 \rightarrow 78 \rightarrow 92 \rightarrow 104 \rightarrow 114 \rightarrow 121 \rightarrow 126 \rightarrow 128$

402.504547 {93.000000; 99.000000}

$1 \rightarrow 2 \rightarrow 5 \rightarrow 17 \rightarrow 32 \rightarrow 50 \rightarrow 61 \rightarrow 68 \rightarrow 76 \rightarrow 92 \rightarrow 100 \rightarrow 116 \rightarrow 120 \rightarrow 126 \rightarrow 128$

402.501434 {87.000000; 105.000000}

$1 \rightarrow 2 \rightarrow 8 \rightarrow 24 \rightarrow 32 \rightarrow 46 \rightarrow 62 \rightarrow 69 \rightarrow 72 \rightarrow 93 \rightarrow 101 \rightarrow 116 \rightarrow 122 \rightarrow$

126 → 128

402.503387 {90.000000; 102.000000}

1 → 2 → 4 → 16 → 36 → 46 → 60 → 69 → 79 → 93 → 101 → 114 → 121 → 126 → 128

402.411713 {116.000000; 104.000000}

1 → 2 → 6 → 16 → 42 → 48 → 59 → 68 → 73 → 92 → 98 → 119 → 123 → 126 → 128

The remaining compromise solutions (Pareto solutions) are included in a computer disc.

The above results are also displayed in Figure 4.14, where the cost and time objectives are plotted for a population of 200 individuals. A generation size of 20 was chosen and 24 initial generations were evolved (number of runs).

Referring to Figure 4.14, a linear function that describes the set of compromise solutions (paths) of the bi-criteria problem is shown. The best compromise solution(s), found by the technique, can be obtained by observing the corresponding fitness values of these solutions. This is depicted in Figure 4.15, where the compromise solutions' cost and time objectives, of the 13th evolved generation, are plotted versus their fitness values. It can be seen that several best compromise solutions are found. These solutions have high fitness values compared to the rest of the solution pool.

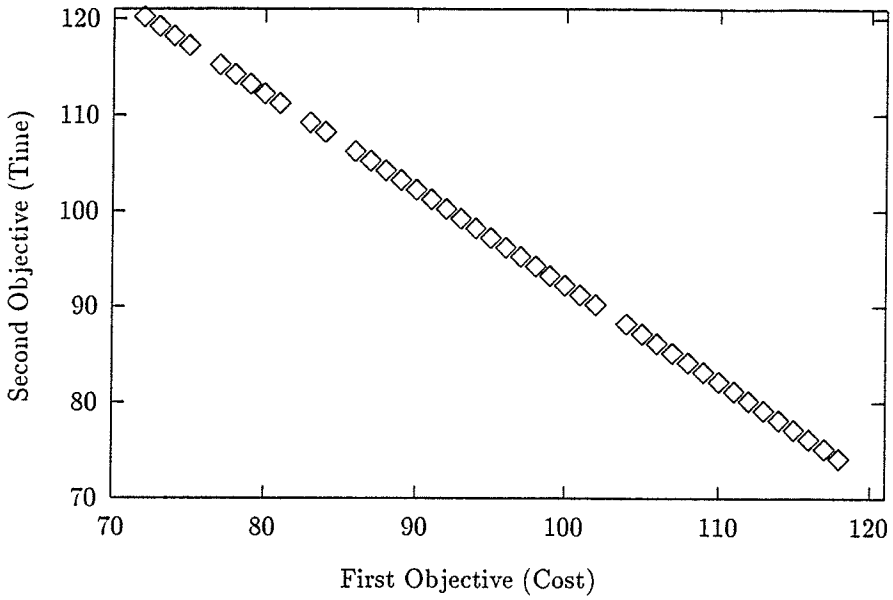


Figure 4.14: Conflicting criteria compromise solutions.

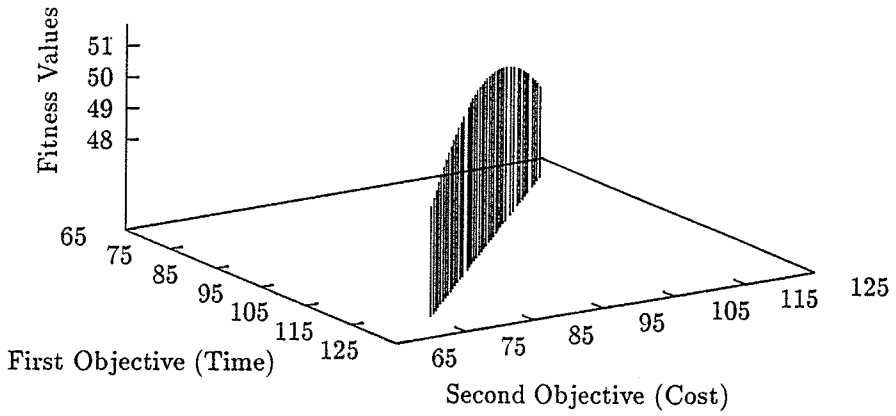


Figure 4.15: Solution points of the final generation.

4.4.2 Multi-objectives Case Study

The single objective problem is to find the optimal solution in a network of paths. This concept of optimality must be dropped for multi-objective problems because a solution which maximizes (minimizes) one objective will not, necessarily, optimize the rest of the objectives [30]. Instead, a new notion of finding compromise solutions (Pareto solutions) is adopted for the multi-objective case [8]. This notion states that a solution is Pareto optimal (compromise) solution when for every improvement in one objective of that solution, one or more objectives of the compromise solution deteriorate.

The multi-objective case does not affect the size of the basic single objective case; it, however, adds a different kind of complexity and computational expense that is related to the number of objectives [30]. This section deals with the same network example of Section 4.4.1.

The single objective case of this network, namely finding the path of the least cost criterion is treated first. Then, another single objective is added, one at a time, to the network arcs. A total of four objectives are attempted here. All objectives are of equal importance, i.e. possess equal weights. The values of the objectives were generated as follows: the first objective (cost) values were randomly generated between $1-15$ units, the second (quality), third (time) and fourth (distant) objective values were generated between $1-50$ units. A comparative study is carried out for the increase in computational complexity as the number of objectives increases. Finally, a typical convergence study is performed for the three-objective case.

Equation (3.2) is used in order to determine the fitness values of each path. Table 4.3 depicts a comparative study of the network examples. The first column shows the number of objectives, attempted at a time. The second column displays the upper/lower values of each objective criteria. The separately attainable optima

for each objective are stated in the third column. The fourth column shows the worst attainable solutions for these objectives. These values were used to determine the best compromise solution exhaustively, as was described in Chapter 3, Section 3.4.2. The results of the exhaustive procedure was used as the stopping criteria for the stage set method. For example, Table 4.3 depicts the best and worst separately obtained solutions in the third and fourth columns. The combined best criteria values of the third column yield a hypothetical fitness value of 401.8, whereas the combined worst criteria values of the fourth column of Table 4.3 yield a worst hypothetical fitness value of 400.0. The exhaustive procedure picks a fitness value in order to stop the algorithm at a value situated between these two extreme hypothetical values, say at 401.4. The procedure will continually increment this last value until the algorithm yields no new solutions, after a long period of running time. Our procedure then backtracks to the last verified fitness value and starts a new increment, e.g. a lower increment than the previous one. The procedure continues in this manner until no further improvement of the latest verified fitness values can be made. This fitness value is then used as the stopping criterion of the algorithm in order to analyze the CPU time growth of the algorithm. Alternatively, the algorithm can be stopped at a predetermined number of evolved individuals or when the percentage of optimal solutions in a generation reaches a certain predetermined level. The fifth column of Table 4.3 shows the best compromise solutions for each case. The pool of compromise solutions of the 2, 3 and 4 objectives cases along with the input parameters are shown in Appendix F. The sixth column of Table 4.3 shows the average CPU time to find the best compromise solution. The seventh column shows the average CPU time required to evolve a single individual; a low order of relative increase in CPU time is observed throughout these cases. Finally, the last column depicts the ratio of relative increase of the average CPU times. It is noticed from.

4.3: Table 4.3: Comparative study of the multi-objective network. Table

# of Object.	U of Objectives $\{Cost; Quality; Time; Distance\}$	Separate Optima	Separate Worst Solutions	Pareto (Compromise) Solutions	Average CPU per Optimum sec.	Average CPU per Solution msec.	Relative Increase to 1-Object.
1	$\{173; -; -; -\}$	$\{33; -; -; -\}$	$\{169; -; -; -\}$	$\{33; -; -; -\}$	1.7307	0.383	1
2	$\{173; 412; -; -\}$	$\{33; 388; -; -\}$	$\{169; 53; -; -\}$	$\{49; 338; -; -\}$	5.4966	0.419	3.175
3	$\{173; 412; 226; -\}$	$\{33; 388; 36; -\}$	$\{169; 53; 217; -\}$	$\{53; 312; 66; -\}$	6.1900	0.452	3.576
4	$\{173; 412; 226; 345\}$	$\{33; 388; 36; 71\}$	$\{169; 53; 217; 326\}$	$\{52; 310; 78; 106\}$	6.7307	0.499	3.889

the last column of Table 4.3 that the increase in CPU time is rather low, i.e. low growth of computational complexity of the algorithm, as an additional objective is added to the second objective criterion case up to the fourth objectives case. However, there is a relatively large increase of CPU time values in moving from the single objective case towards the two objectives case. This is due to the radical transformation of the problem domain itself. The algorithm does however exhibit a smooth and natural behavior in handling both the single objective case for the multi-objective cases that are attempted in this thesis. The single objective average CPU time was 1.73 seconds which compares favorably with the 5.04 seconds result obtained from the Out-of-Kilter technique, under the same computer environment (UNIX/SUN 4.1 Workstation).

The findings in this section are encouraging in that, classically, the complexity of multi-objective problems grows exponentially with the increasing number of objectives towards finding a compromise solution [30].

The three-objectives case, shown in the third row of Table 4.3, is analyzed here in terms of convergence to the best compromise solution. A generation of 200 individuals (solution points) were initially, randomly, generated by GA. They were evolved over 20 generations as described earlier; 24 initial generations (number of trial runs) were attempted. Every trial run represents a different starting, random, generation of potentially different regions of the solution space. The convergence behavior of the initial run is shown in Figure 4.16 (a). A sample of the convergence behavior of the algorithm, for other runs, is shown in Figure 4.16 (b and c). In this figure, the fitness values of each individual in a generation (the mapped objective values) are plotted for all evolved generations. A general trend of improvement of the convergence to a better performing generation is observed. That is, the fitness values of every succeeding generation were steadily improving as the algorithm.

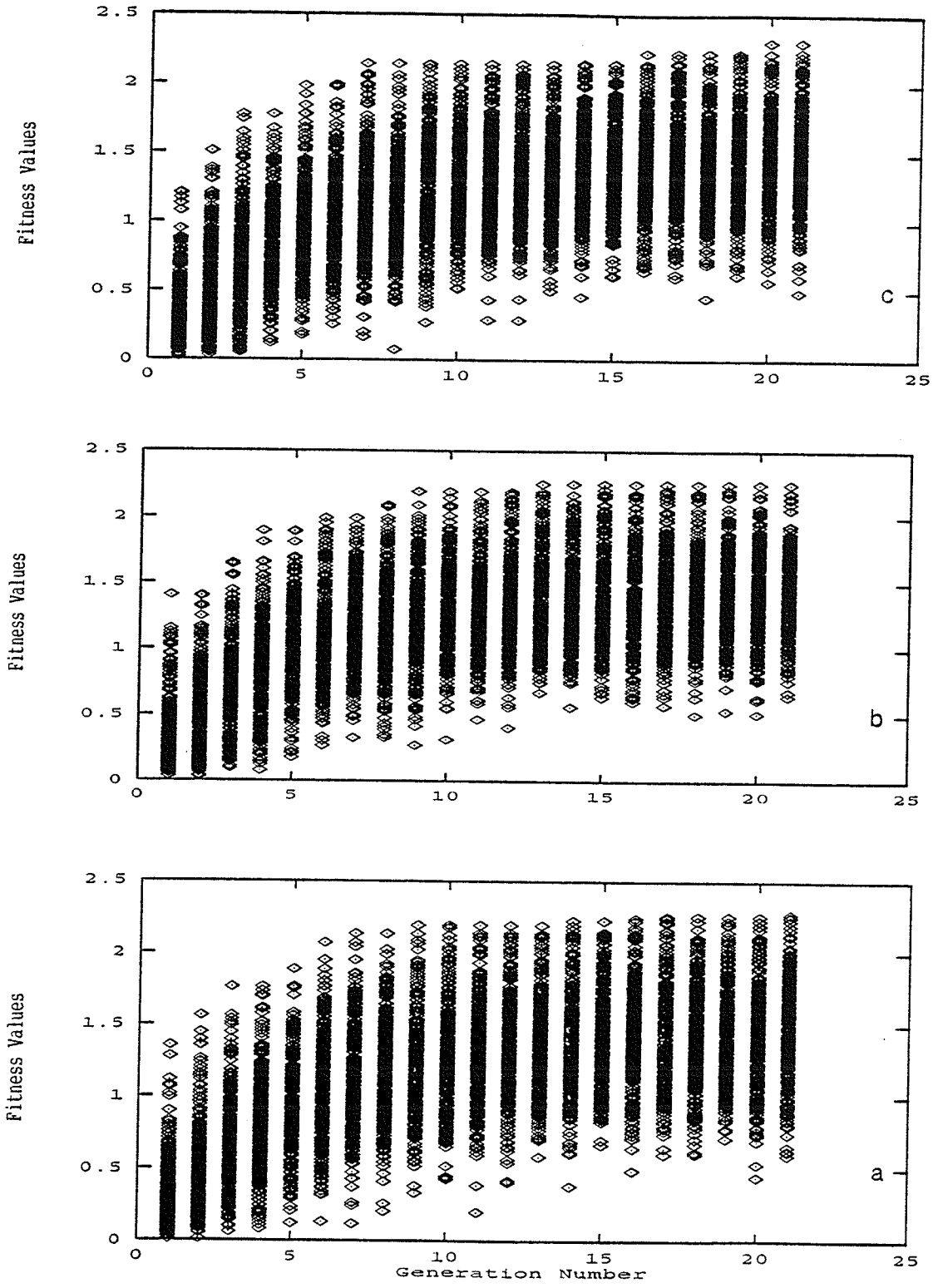


Figure 4.16: Convergence behavior; typical trial runs.

proceeded from one generation to the next. As can be seen in these figures, the trend is not continually improving. The levelling-off of the trend indicates that the algorithm can be stopped at an earlier generation of an earlier trial run. Consider for example Figure 4.16 (b): the algorithm can be stopped at the thirteenth generation of the eleventh run at the next best solution.

It is also observed that the final generation of all trial runs comprises a range of individuals with a variety of fitness values, i.e. the final generation does not necessarily converge to a single solution. This range includes the best compromise solution as well as several sub-optimal solutions that can be useful to know so as to present the decision maker with a variety of alternatives.

A useful indicator of the convergence behavior of the algorithm is the average fitness value of each generation; this is depicted in Figure 4.17. This figure shows an increasingly improving (rising) curve of the average of fitness values for each

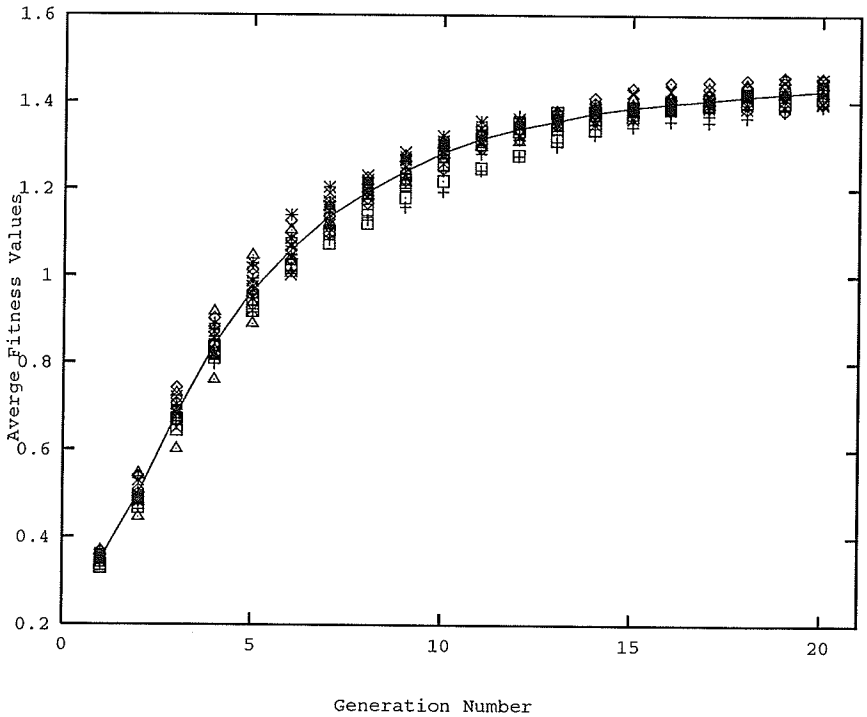


Figure 4.17: The average trend of fitness values.

generation. Figure 4.17 also shows only slight improvement after the $17th$ generation. This trend of convergence behavior may be used as one factor in determining the stopping criterion for future applications of the algorithm, as described earlier in this thesis, however, these results are application dependant.

4.4.3 Constraint Shortest Path Case Study

The shortest path problem with an added constraint is termed the constraint shortest path problem. This problem is an NP-Complete problem [23, 26, 27]. A problem is considered an NP-Complete problem when there exists no solution technique that solves it within a computational growth rate that is polynomial in relation to the input data of the problem [48].

The conventional formulation of this problem was described earlier in Chapter 2. In this work, the CSPP is formulated in conventional manner except that each individual in a population is tested for satisfying the constraint shown by Equation (4.1); In order to apply GA to solve this problem, the constraint described in Equations (2.4 - 2.5), is rewritten here.

$$\sum_{i=1}^{m} t_i X_i <= T \tag{4.1}$$

t_i is traversal time of a single path, X_i is the flow at each arc and T is the total allowed time for the best path and . individuals that do not satisfy this constraint are eliminated from the population and those that satisfy the constraint are further tested for their fitness values by applying a fitness function. The main objective function (Equation 2.1) is mapped onto a fitness function, $Y_{j(j=1,2,..,n)}$, as shown below:

$$Y_{j(j=1,2,..,n)} = U - \sum_{i=1}^{m} C_i X_i \tag{4.2}$$

The symbols of this equation are described in Chapter 3. In order to satisfy the

constraints shown in Equations (2.2 - 2.4), the formulation presented in this work assumes a single flow item at each node in the network. Also, an item is supplied at the source node and is received from the sink node of the network. The constraints shown in Equation (2.2 - 2.4) are satisfied in the application of the modification procedure of Section 3.3, i.e. satisfying the condition of one path per individual network.

The evaluated individuals are then evolved by GA operators towards the best solution, as described earlier in Section 3.2.1. A network example of this formulation is presented next.

The network example, previously described for the two–objective (multi-objective) case in Section 4.4.2, is slightly modified to represent the CSPP. This is done by insisting that the total time (T) allowed per solution point (network path) to be some given values, also the second objective (quality) is now called 'time'. Each value of T represents a single example of the network problem. The network's T restrictions are described in Table 4.4. The second objective values is assumed here to represent the t_i values for the CSPP.

The first column of Table 4.4 shows the type of the example handled by the algorithm. The second column depicts the various T values for every example. The motivation behind using these T values is the fact that for each case of T, a different number of paths would be infeasible, thus rendering the search for an optimal solution more difficult from the previous (large T value). The third column of Table 4.4 shows the average CPU times for all T values that are required to reach the optimal solutions. The fourth column of the table shows the average CPU times for all T values that are required to reach a single solution, i.e. the CPU time per individual. The first row of that table depicts the single objective case outcome that is comparable to the second row which shows the outcome for a T value of 389. The

Table 4.4: Results of CSPP for various T values.

Type of Examples	T Values	Average of CPU Time per Optimum sec.	Average of CPU Time per Solution msec.	Optimal Solutions $\{Cost; Time\}$
single-objective case	–	1.73	0.383	$\{33; -\}$
subject to $T1$	389	1.89	0.447	$\{33; 217\}$
subject to $T2$	339	2.003	0.432	$\{33; 217\}$
subject to $T3$	288	2.290	0.465	$\{33; 217\}$
subject to $T4$	200	2.40	0.44	$\{35; 184\}$
subject to $T5$	188	2.873	0.4397	$\{35; 184\}$
subject to $T6$	128	2.638	0.432	$\{58; 123\}$
2-objectives case	–	5.49	0.419	$\{84; 383\}$
2-objectives-CSPP	384	7.04	0.468	$\{84; 383\}$

next five rows of the table show the various T values examples.

Figure 4.18 shows the convergence behavior of the $T5$ case which is shown in Table 4.4. The percentage of eliminated individuals, ie. individuals of greater values than that of $T5$ and $T6$ cases of Table 4.4, is shown in Figure 4.19.

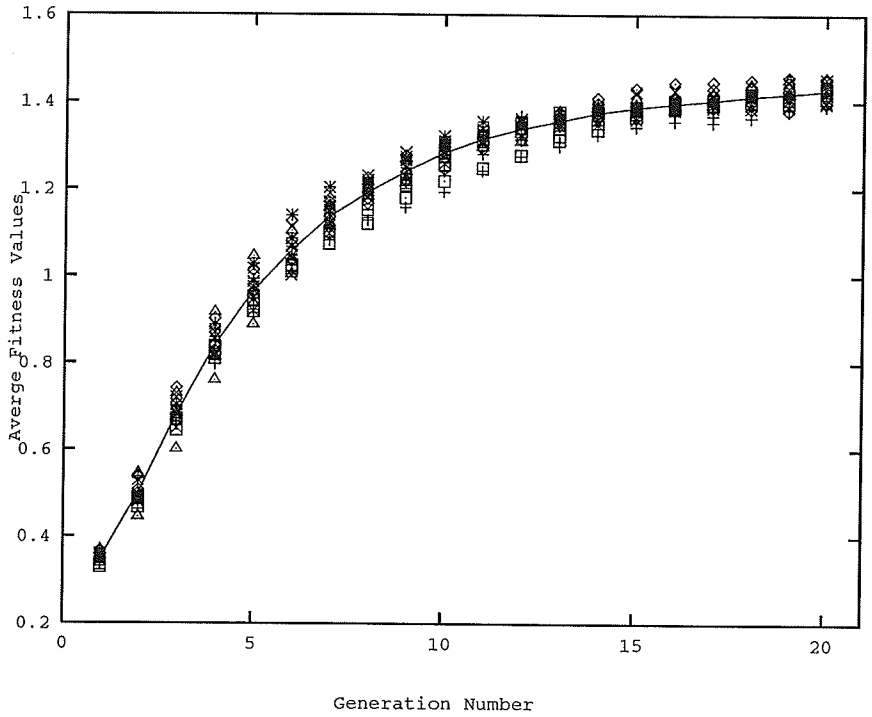


Figure 4.18: The average trend of fitness values.

It is possible, in real life applications, that CSPP may be required to be solved under the previously described conditions in addition to finding the feasible solution which is the best compromise solution. Formulating this new problem involves the same formulation of CSPP, described above, with the exception of using the multi-objective formulation, Equation (3.2), described in Section 3.2.2, instead of Equation (4.2). The constraint described in Equation (4.1) is first satisfied (as described in the previous section). The compromise solutions of the multi-objective network are then identified.

The trend shown in the third and fourth columns of Table 4.4, as predicted, was

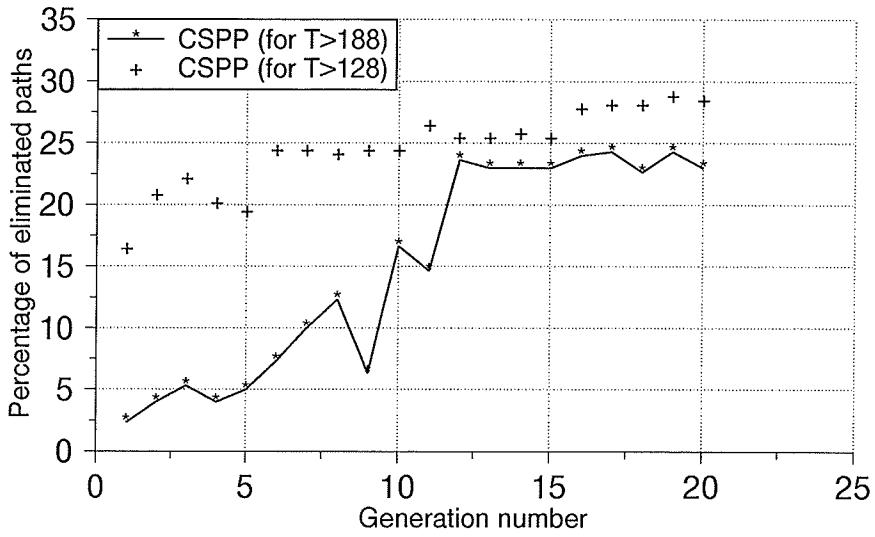


Figure 4.19: CSPP elimination of individuals.

that of increasing CPU time for increasing T values except for the last trial where, in this particular example, the solution space was richer than the previous cut-off range of T value $= 188$. The amount of required evaluations in order to find the optimal solution varied substantially in accordance with the new solution space of feasible solutions at every range of T values. This is noticed by comparing the last two column of Table 4.4, where the lower values of CPU times to finding a single solution point did not guarantee a similar lower trend for finding the optimal solutions. We attribute this phenomenon to the stochastic nature of the GA and also to the clustering of some solution points in close or sparse regions of the solution space. The latter reason is problem dependent, while the former is GA related. However, the differences of the average CPU time of the majority of T values examples, shown in the third column of Table 4.4, are small enough and should not affect the general trend of convergence for other T values or for other examples of networks. The last column depicts the optimal solution (or the best compromise solution) obtained for

each case.

The last two rows of Table 4.4 display the multi-objective case that is handled by using the formulation described by Equations (4.1 and 3.2), and finally the CSPP of the multi-objective case with an added constraint that requires a T value to be no more than 384 is shown in the last row. The corresponding average CPU time for finding the optimal solution and a single solution are shown in the second and third columns of that table. We utilized the multi-objective formulation , described in Chapter 3, in order to find the best compromise solution of this case. The network has two objectives, cost and time, criteria of equal importance (the weights are of equal values). The best compromise solution was found after 5.49 seconds as shown in the table. This is clearly larger than that for CSPP of all T values; we attribute this difference to the increased complexity of the problem domain compared to either the CSPP or to the single objective problem. This complexity continually arises in multi-objective cases for every solution point evaluated, since the problem exhibits a nonlinearity that is proportional to the number of objectives.

A comparison of the CSPP with the multi-objective case reveales that the difference of the average CPU time values, for finding a single solution, is very small for all T values; we can infer that for some problems both methods may possess similar CPU time values. The seventh row of Table 4.4, however, shows a much larger difference with the basic CSPP. We attribute this to the fact that in obtaining the result of the last row, the algorithm had to find the best compromise solution as well as having to satisfy the T values constraint. The single objective row (the first row) shows, predictably, a lower value of the average CPU time than for the other problems; the Out-of-Kilter method, however, required 5.09 seconds to find the optimal solution. The single objective case took less time than either case because infeasible solutions are not included in the search space of the single objective case.

Every solution that does not satisfy the T value constraint is considered infeasible in the other methods, thus incurring an extra computational cost for generating it.

The CPU time trend for all T values revealed a low polynomial time behavior, using the procedure introduced in this work. This is an encouraging result since CSPP is proven to be NP-Complete problem [23, 26]. However, our method is stochastic in nature, so no guarantee is given, at all times, that the final outcome of the algorithm is the optimal solution; though for all previous cases, the optimal is found or at worst, the next best solution is.

4.4.4 Multi-Processing Planning Case Study

In earlier chapters we demonstrated the developed algorithms for handling the single and multiple objectives cases in finding the best process plan (process routes) for a single part that is machined in various alternative machines. We now show that the same methodology is applicable for handling the process planning issues, modeled by flow networks, of multiple parts that require the same or similar machining operations (family of parts). These parts can be either machined in single machines as demonstrated earlier, or machined in multiprocessing machining centers as well.

A multiprocessing network example is borrowed from [56] and is shown in Figure 4.20. The process route for each part of Figure 4.20 comprises multiple operations [56]. It can be seen that these operations (planing and drilling) can be represented as two networks in series. The first network starts at node 1 and ends at node 6 for the planing operation of the three parts. The second network, in the series, starts at node 6 and ends at node 11 for the drilling operation of the three parts.

The parts, shown in Figure 4.20, can be machined in multistage fashion. Each stage in multi-stage manufacturing systems represents a required operation that can be carried out in a number of alternative machines. These operations can be of the

single processing type, i.e. operating on a single part, or of multiprocessing type, e.g. many operations can be carried out on the same part or many parts sequentially. The multiprocessing machines are represented by a single arc. This arc, however, is situated (Figure 4.20) such that one or more stages are skipped due to the fact these machines can carry out many operations, i.e. stages, at the same time.

Figure 4.20 also shows group processing machining centers, M1 and M2 shown.

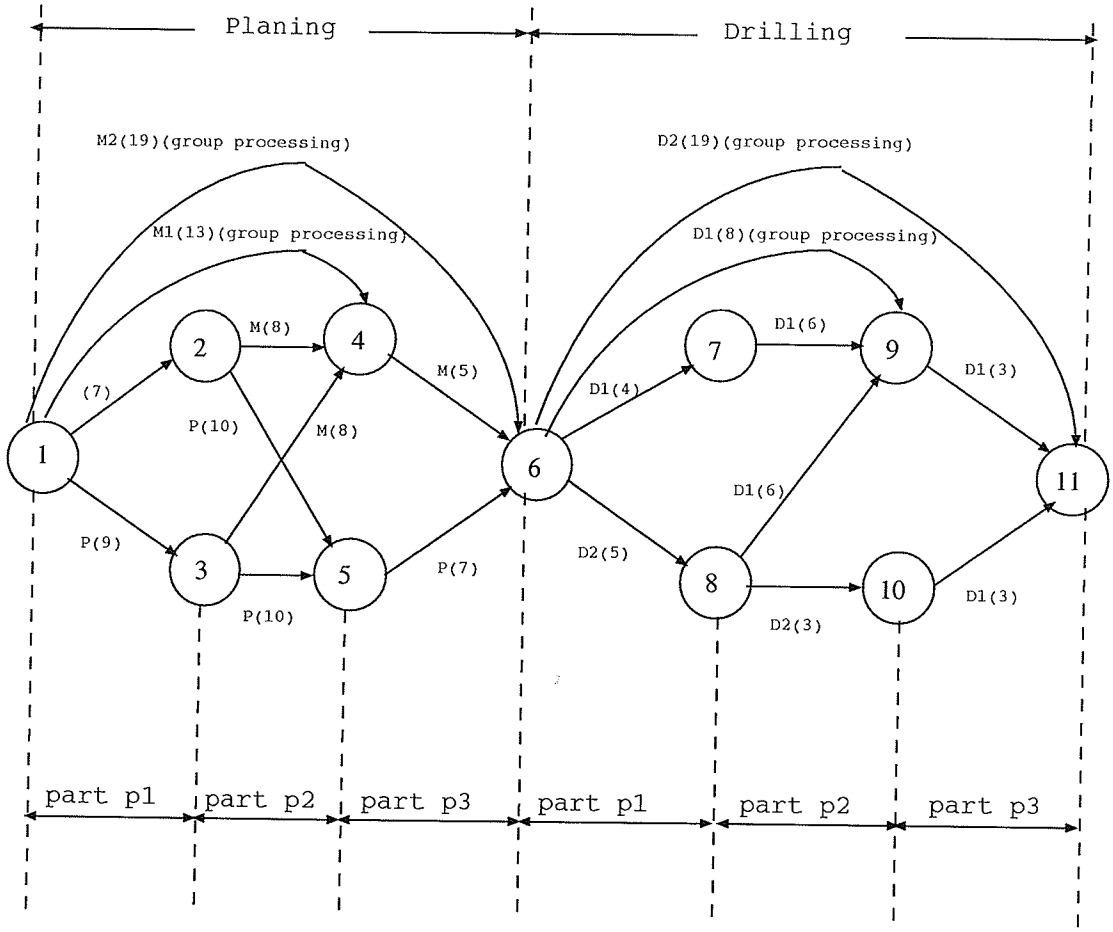


Figure 4.20: Multi-parts and multi-processing flow network.

as arcs linking the first node with the fourth and sixth nodes respectively, where two or three parts can be machined in these centers, respectively. These machining cen-

ters are shown as extra alternatives in the network, in addition to the conventional machines that comprise the rest of the network alternatives, discussed earlier.

The skipped stages, in our formulation, must be configured, by using dummy nodes, in order to appropriately accommodate them using the formulation that has been presented earlier in this chapter. All links, input and output, of these nodes carry a 'zero' value (objective value) for those links leading to the multiprocessing machine (node); and a large value for the rest of the links (Figure 4.21), for the minimization of objectives case (the opposite is true for the maximisation case). Referring to Figure 4.21, an alternative representation of the multiprocessing activities is depicted.

The $M1$ group processing alternative shown in Figure 4.20 is represented by a dummy node 2 which becomes an output for the first stage with zero processing time and has a large processing value (shown as infinity in Figure 4.21) for the arcs leading to any other node that is not the original alternative intended to reach (node number 4 in Figure 4.21). The actual total multiprocessing time is applied as shown in the arc linking node 2 and node 4 in Figure 4.21. The same logic is applied for the rest of multiprocessing alternatives as depicted in Figure 4.21. The algorithms, described earlier, are then readily aplicable to handle the modified network of Figure 4.21. The extra modification, required to represent the dummy nodes, adds some overhead to the estimated complexity of the algorithm; however this overhead is negligible, as described in Appendix E.

The optimal process plan is found to be $1 \rightarrow 2 \rightarrow 4' \rightarrow 6 \rightarrow 7' \rightarrow 9 \rightarrow 11$ as shown in Figure 4.21. This optimal path corresponds to $1 \rightarrow 4 \rightarrow 6 \rightarrow 9 \rightarrow 11$ in Figure 4.20. The solution has a total processing time of 29 hours, the same result as in [56]. In this process plan parts 1 and 2 are to be processed on both machines M1 and D1 in group processing and part 3, on machines M and D1 in conventional

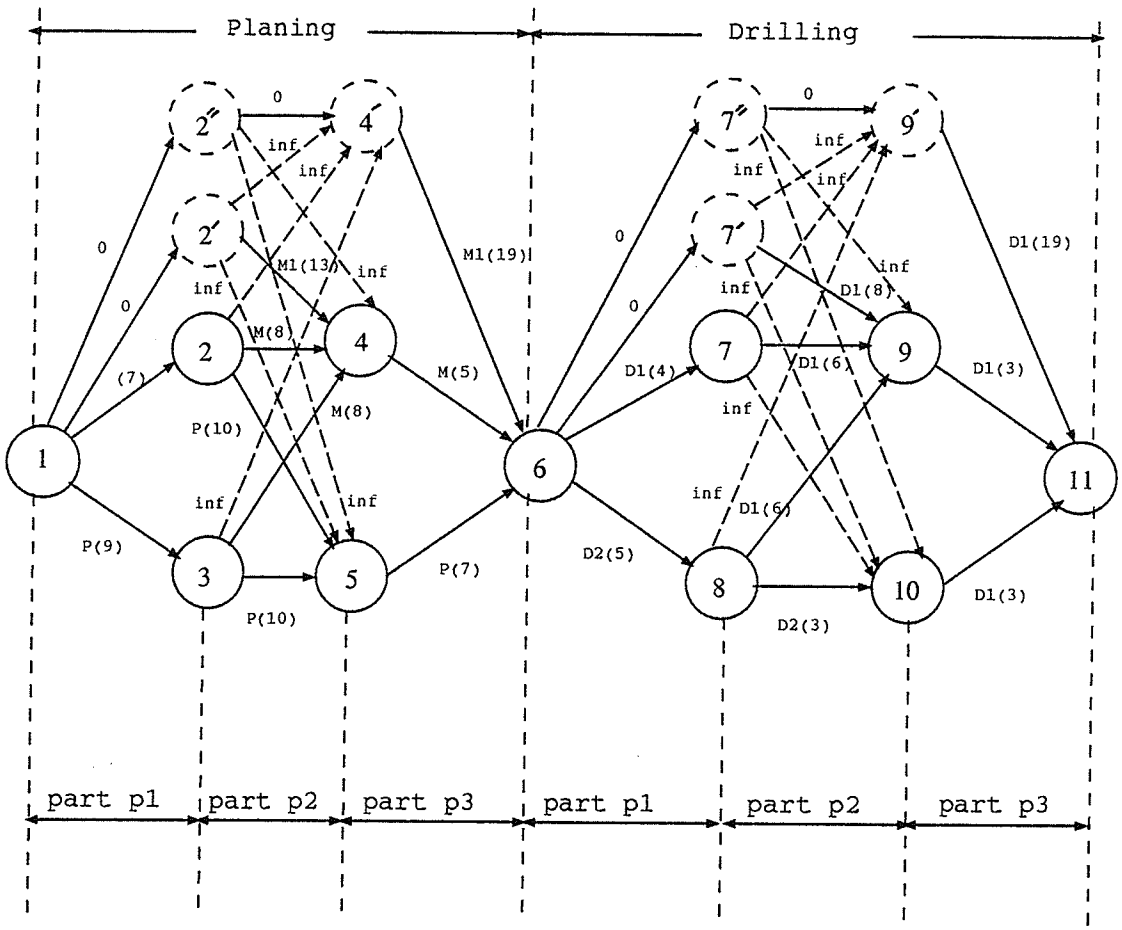


Figure 4.21: The modified flow network.

processing.

The average CPU time required by the stage set procedure was 3.846 milliseconds whereas the CPU time of the Out-of-Kilter procedure was 2.71 seconds; the average CPU time for finding a single solution was 0.121 milliseconds. Obviously, the same formulation for the multi-objective case can be applied for the corresponding multi-objective network in the same fashion that is described in this work.

4.4.5 Dense Networks Case Study

Five examples of large size networks of dense arc distribution are attempted here. The nodes at each stage were connected with all other nodes of the following stages. Nodes situated at the same stage were not connected with arcs. All arcs were of a forward nature. The objective values (costs) were randomly generated between $1 - 100$.

Consider the network shown in Figure 4.22 which is modified from Figure 3.1. The basic arcs shown in Figure 3.1 remain the same. In addition to these arcs, multiprocessing operations arcs are added at every node in every stage of the network (see Figure 4.22). These arcs connect the nodes of any given stage to the alternate stages that follow it, i.e. these arcs skip the immediately succeeding stage and connect with the nodes of the rest of the stages in the network, see for example arcs $1 \rightarrow 4$, $4 \rightarrow 9$, $2 \rightarrow 7$ an $5 \rightarrow 9$; the costs associated with all arcs in this network is not shown in the figure (for clarity). The costs of this network and the rest of the examples (shown in Table 4.5) were generated to reflect this particular structure and are included in the computer disc.

The results of applying the stage set GA procedure to the above example and. four more examples are depicted in Table 4.5. In this table, the first column shows the number of examples that were randomly generated. The second column shows

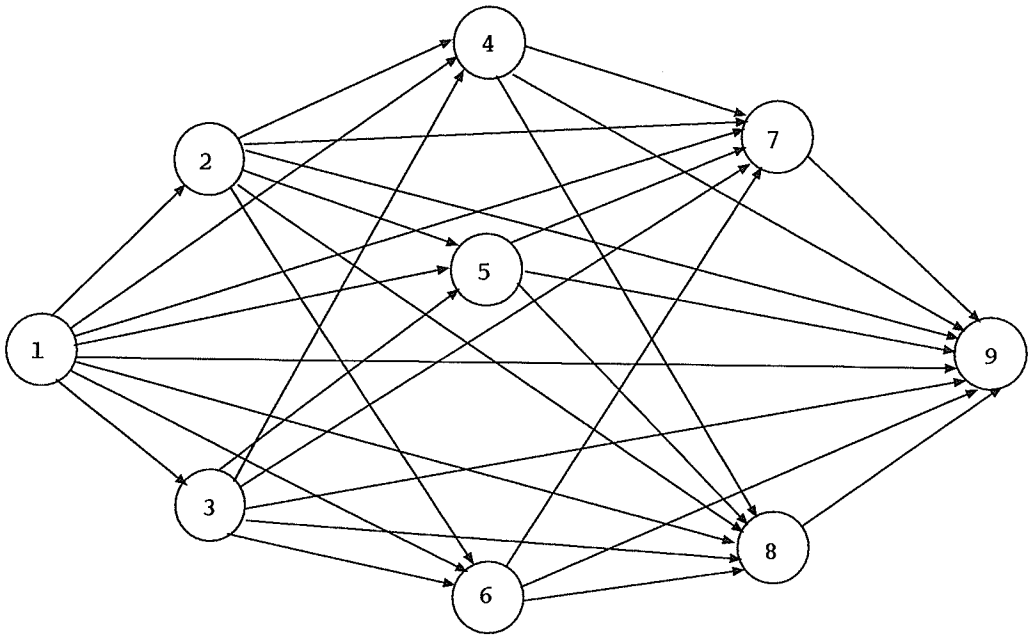


Figure 4.22: Typical dense network.

the size of each network, in terms of the number of stages. The third column depicts the configuration of the networks where the source node and the output nodes of each stage are shown. The fourth and fifth columns show the network sizes, in terms of the number of nodes and arcs. The sixth column consists of the upper U values (described in Chapter 3) of the processing time of each network that is needed for the mapping of the fitness function. The seventh column shows the optimal values of the optimal path of each network. The fifth column depicts the CPU time results of the Out-of-Kilter method. The last column shows the average CPU time required to find the optimal solution.

The population size selected for the first and second examples was 10 and 100 population size was chosen for the rest of the examples. The probablity of crossover was selected as 1. The optimal solution for the first example was $1 \rightarrow 2 \rightarrow 9$. The second example has an optimal path of $1 \rightarrow 3 \rightarrow 12 \rightarrow 13 \rightarrow 18 \rightarrow 20$; the optimal path for the third example is $1 \rightarrow 4 \rightarrow 11 \rightarrow 18 \rightarrow 32 \rightarrow 37 \rightarrow 40$; the optimal path

Table 4.5: Results of single objective dense networks.

#	Networks Size (Stages)	Type of Network Stages	# of Nodes	# of Arcs	U Object.	Optimal Solutions	CPU Time Out-Of -Kilter Av. sec.	CPU Time Stage Set Av. sec
1	4	{1}{2}{3}{2} {1}	9	64	300	50	3.45	0.05
2	5	{1}{3}{8}{4} {1}	20	72	189	43	5.99	0.18
3	6	{1}{4}{8}{12} {10}{4}{1}	40	296	194	38	5.6	2.7
4	8	{1}{10}{19}{30} {20}{10}{5}{4} {1}	100	1644	586	7	20.04	11.86
5	11	{1}{49}{50}{50} {50}{50}{50}{50} {50}{50}{49}{1}	500	22498	546	4	42.43	14.4

for the fourth example is $1 \rightarrow 49 \rightarrow 93 \rightarrow 100$ and finally, the last example has an optimal solution path of $1 \rightarrow 222 \rightarrow 449 \rightarrow 500$. It is evident, from Table 4.5, that the developed procedure exhibits a good performance, in terms of convergence to the optimal solution in reasonable time, and in terms of robustness for various sizes of networks. The CPU time requirement for these examples is substanially lower than that for the earlier ten example, presented in this chapter, since the application of the modification procedure is minimal. More case studies are discussed in Appendix G.

Finally, the formulation presented in this thesis can easily accommodate for situations where machine failures occur. The set of nodes that correspond to the output nodes of every stage can be reduced by the number of failed machines, or, the related criteria entries are assigned very high or low values (according to the type of optimization). A similar treatment can be used when an alternative process route is no longer needed or desired by a decision maker.

4.5 Summary

A stage set approach was introduced in this chapter with the goal of reducing the amount of computation required to solve the multi-objective problems. In this method, the characterstic of complete enumeration simplicity is coupled with a convenient representational technique, based on set theory, that allows for the application of GA in order to handle multi-objective problems as well as CSPP.

Obviously, the computational requirements for this method are much lower than that of other classical methods (e.g., Out-of-Kilter for the single objective case). It is observed that the GA scans vastly large regions of the solution space by searching a small portion of that space. The complexity and CPU time study of the algorithms have demonstrated a good convergence rate for several sizes networks. A comparison

of the CPU time complexity analysis for the multi-objective cases with other classical techniques was not possible. To the best of our knowledge no CPU time study was available in the literature.

The actual set operations were emulated under a C-Unix programming environment. The implementation can be substantially improved if a specialized stage set procedure were to be employed (instead of their emulations). The implementation can be further improved by using an optimized and larger C compiler than the current one, with a single precision floating point option. This is especially important for the ability to handle larger sizes of populations needed as the network problems grow. Finally a parallel implementation, of all the GA procedures, the stage set and the stage matrix procedures is possible. Indeed, the GA parallelisations have been implemented by many researchers [37, 53, 54, 55]. The stage set and stage matrix methods can be similarly parallelized. A full realization of the true computational capabilities of these methods will only be appreciated in their parallel implementation version, especially for network problems larger than the ones illustrated in this thesis.

CHAPTER 5

Concluding Remarks

In this thesis a novel model has been developed for routing problems that are represented by flow networks. These networks delineate the multiple alternatives that normally exist in process planning activities. Optimization techniques based on a genetic algorithm, which suits the combinatorial nature of such flow networks, were developed.

The bit string representation of alternative paths was initially facilitated by introducing a novel stage matrix modeling approach. Structured randomized modification procedures were added to ensure that every solution point in the search space, evaluated by the genetic algorithm, is represented as a single path. These modification procedures provided with the ability to scan neighborhoods of the solution space points and not just the immediate descendants of the previous generation. This quality is of paramount importance in global optimization issues. The stage matrix formulation introduced a new notion to genetic algorithms, the reccessive genes concept that markedly improved the convergence behavior of these algorithms towards the best solution. These genes were represented as extra links that exist in the unique path network and can be exchanged through many future generations.

This initial formulation led to the development of a new model, "the stage set method". The stage set technique represents the nodes of the network in "zero" and "one" format. This method compares favorably to the stage matrix method since it requires a shorter string size than the stage matrix method. A simpler modification procedure than that of the stage matrix method is still possible with the

characteristic of less frequent applications than that of the stage matrix modification operation. Finally, a faster computational time was observed for the stage set method than for the stage matrix method with a similar convergence trends towards the optimal solution.

The computer software, based on the two models, was tested for different sizes of networks having single or multiple objective criteria and was also observed to solve the constrained shortest path problem. The algorithms have shown promising performance in terms of robustness and ability to handle large sizes of networks. The genetic algorithm parameters chosen were very similar to those suggested by other researchers for traditional genetic algorithms and were found to work well across a variety of problems types.

This study showed that, by using genetic algorithms, the transition from handling the single objective class of optimization problems to the multiobjective class can be of a smooth nature in terms of formulation and computational time. This was clearly observed in the transition from the bi-criteria case to the multiple criteria cases. This smooth transition is an important trait which genetic algorithm-based techniques possess, with regard to formulating a general optimization procedure and designing computational procedures that can be implemented to handle a wide variety of optimization problems.

It was also observed that the stage set algorithm has a low computational polynomial time of growth as the number of objectives is increased in a particular network problem. This is an important feature since classical optimization techniques (e.g. linear, goal, dynamic programming techniques) have an exponential growth rate in relation to the increased number of objectives [30].

Genetic algorithms are inherently parallel search algorithms. Their computational speed can be enhanced using parallel techniques [37, 53, 54, 55]. In fact, in

order to realize their full potential, genetic algorithms must be implemented on parallel computer architectures [53]. This property can be utilized to increase the speed of computation especially for the multi-objective and the constraint shortest path problem cases. This increase in computational speed is important for handling large and very large networks. It is suggested that such parallelisation would substantially enhance the performance of the models presented in this thesis, in comparison to other classical techniques currently employed to handle process planning selection problem.

The contributions and the observed results of this study are promising in terms of the ease of formulation of the single objective process planning selection problem (or the shortest path problem), its extension to solve the multi-objective process planning selection problem and the convergence to an optimal, or best efficient (for the multi-objective case), solution.

REFERENCES

REFERENCES

[1] A. A. G. Requicha and J. Vandenbrande, Automated Systems for Process Planning and Part Programming, In *Artificial Intelligence: Implication for CIM* (Edited by A. Kusiak), IFS Pub., Kempston, UK , pp. 301–326 (1988).

[2] T. C. Chang and R. A. Wysk, *An Introduction to Automated Process Planning Systems.* Prentice-Hall, New-Jersey (1985).

[3] A. Kusiak, Integer Programming Approach to Process Planning, *Journal of Advanced Manufacturing Technology* **1**, pp. 73-83 (1988).

[4] K. Hitomi, *Manufacturing Systems Engineering.* Tylor and Francis Ltd, London, England, 1979.

[5] I.M. Sobol, An Efficient Approach to Multicriteria Optimum Design Problems, *Surveys on Mathematics for Industry* **1**(4), pp. 259–281 (1992).

[6] R. Steuer, Multiple Criteria Optimization: Theory, Computation and Application, John Wiley & Sons Inc., New York, (1986).

[7] D.J. White, the Set of Efficient Solutions for Multiple Objective Shortest Path Problems , *Computers and Operation Research* **9**(2), pp. 101–107 (1982).

[8] J.L. Cohon, *Multiobjective programming and Planning*, Academic Press Inc., New York (1978).

[9] S. Goyal, *Flexibility Trade-offs in a Random Flexible Manufacturing System: A Simulation study*, MSc. Thesis, University of Manitoba, Manitoba, Canada, 1989.

[10] K. E. Stecke, Formulation and Solution of Nonlinear Integer Production Planning Problems for Flexible Manufacturing Systems *Management Science*, **29**, pp. 273–288, (1983).

[11] J. Szadkowski, An approach to machining process optimization , *Int. J. Prod. Res.* **9**(3), pp. 371–376 (1971).

[12] K. Shankar and Y. J. Tzen, A loading and dispatching problem in a random flexible manufacturing system, *Int. J. Prod. Res.* **23**, pp. 579–595 (1985).

[13] P. Kumar, N. Singh and N. K. Tewari, A nonlinear goal programming model for the loading problem in a flexible manufacturing system, *Engineering Optimization* **12**, pp. 312–323 (1987).

[14] J. C. Ammons, C. B. Lofgren, and L. R. McGinnis, A large scale work station loading problem, *Proc. First ORSA/TIMS Special Interst Conf. on Flexible Manufacturing Systems.* Ann Arbor, MI, USA, pp. 249–255 (1984).

[15] N. Singh, Y.P. Aneja and S. P. Rana. Multi-objective modelling and analysis of process planning in a manufacturing system. *Int. J. Sys. Sci.* **21**(4), pp. 621–630 (1990).

[16] A. Osyczka, *Multi-criterion Optimization in Engineering* , Chiclester: Ellis Howard Ltd. (1984).

[17] N. Singh and B. K. Mohanty. A fuzzy approach to multi-objective routing problem with applications to process planning in manufacturing systems, *Int. J. Prod. Res.* **29**(6), pp. 1161–1170 (1991).

[18] N. G. Sancho, A multi-objective routing problem, *Engineering Optimization* **10**, pp. 71–76 (1986).

[19] H. Bellman, *Dynamic Programming* , Princeton University Press, Princeton, New Jersey, 1957.

[20] M. Sniedovich, a Multi-Objective routing Problem Revisited , *Engineering Optimization* **13**, pp. 99–108 (1988).

[21] E. Minieka, *Optimization Algorithms for Networks and Graphs*, Marcel Dekker Inc, New York NY (1978).

[22] A. P. Jensen and J. W. Barnes, *Network Flow Programming*, Wiley, New York (1980).

[23] R. K. Ahuja, L. T. Magnanti and B. J. Orlin, *Network Flows: theory, algorithms, and applications.* Prentice-Hall, New-Jersey (1993).

[24] J. K. Whatley, *SAS/OR User's Guide: Version 5 Netflow Procedure*, SAS Institute Inc., Cary, NC, pp. 211–223 (1985).

[25] G. D. Jr. Forney, The viterbi algorithm, *Proceedings of IEEE* **61**(3), pp. 268–278 (1973).

[26] M. M. D. Hassan, Network reduction for the acyclic constraint shortest, path problem. *Eur.J.OR.* **63**. pp. 124–132 (1992).

[27] R. Hassin, Approximation schemes for the restricted shortest path problem. *Mathematics of operation Research* **17**, No. 1, pp. 36–42 Feb,(1992).

[28] Y. Swaragi, H. Nakayama and T. Tanino, Theory of multiobective optimization, Academic Press Inc., New York (1985).

[29] J. Ignizio, *Linear Programming in Single and Multiple Objective Systems*, Prentice-Hall, New-Jersey (1982).

[30] M. Zeleny, *Linear Multiobjective Programming* , Springler-Verlag, New York NY (1974).

[31] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Pub., New-York (1989).

[32] I.P. Androulakis and V. Venkatasubramanian, A Genetic Algorithm Framework for Process Design and Optimization, *Computers Chem. Eng.* **15**(4), pp. 217–228 (1991).

[33] F. Glover and H.J. Greenberg, New Approaches for Heuristic Search: A Bilateral Linkage with Artificial Intelligence, *European Journal of Operational Research* **39**, pp. 119–130 (1989).

[34] J. E. Biegel and J. Davern, Genetic Algorithms and Job Scheduling, *Computers and Industrial Engineering* **19**(1 – 4), pp. 81–91 (1990).

[35] K. Kristinsson and G.A.M. Dumont, System identification and control using genetic algorithms, *IEEE Transactions on Systems, Man and Cybernetics* **22**(5), pp. 1033–1046 (1992).

[36] L.B. Booker, D. E. Goldberg and J.H. Holland. Classifier systems and genetic algorithms , *Artificial Intelligence* **40**, pp. 235–282 (1989).

[37] N. Sepehri, F. Wan, K. Kristinsson, G.A.M. Dumont, and P.D. Lawrence. Parallel implementation of a genetic algorithm towards identification of compliance in a hydraulically-actuated robotic arm , *Proc. IASTED Int. Conf. on Control and Robotics.* Vancouver, Canada, pp. 241–244 (1992).

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

[39] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New-York (1991).

[40] G.A. Vignaux and Z. Michalewicz, A Genetic Algorithm for the Linear Transportation problems, *IEEE Transaction on Systems, Man and Cybernetics*, **21**(2), pp. 445–452 (1991).

[41] D. E. Goldberg and R. Lingle, Alleles, Loci, and the Traveling Salesman Problem, In *Proc. Int. Conf. on Genetic Algorithms and their Applications* (Edited by Grefenstette J.), (July 1985).

[42] S. R. Thangaia and M. Hobbs, GIDEON: A genetic algorithm system for vehicle routing with time windows. *Proc. 17th IEEE Conf. on AI applications* , pp. 422–425 (1990).

[43] P. HAJELA, Genetic Search–An Approach to the Nonconvex Optimization Problem, *AIAA Journal*, pp. 1205–1210, July (1990).

[44] C. L. Huntley and D. E. Brown, Parallel Heuristic for Quadratic Assignment Problems, *Computers and Operation Research* **18**(3), pp. 275–289 (1991).

[45] K.A. Dejong, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, Univ. of Michigan, Ann Arbor, MI 1975.

[46] H. Muhlenbein, M. Gorges-schleuter and O. Kramer Evolution Algorithms in Combinatorial Optimization, *Parallel Computing* **7**, pp. 65–85 (1988).

[47] B. Awadh, N. Sepehri, O. Hawaleshka, Process Planning Selection in Manufacturing Systems Using a Genetic Algorithm , *Pacific–Rim International Conference on Modelling, Simulation and Identification,* Vancouver, Canada, pp. 162–168 (1992).

[48] M. R. Garey and D. S. Jhonson, *Computers and Intractibility* , Bell Telephone Lab., Inc., USA (1979).

[49] D. H. Green and D. E. Knuth, *Mathematics for the Analysis of Algorithms*, Birkauser, Boston USA (1981).

[50] J.J. Grefenstette, Optimization of Control Parameters for Genetic Algorithms, *IEEE Transactions on Systems, Man and Cybernetics* **SMC-16** (1), pp. 122–128 (1986).

[51] J.P. Cohoon *et al.*, FloorPlan Design Using Distributed Genetic Algorithm, *IEEE Int. Conf. on Computer-Aided Design*, Santa Clara, pp. 452–455 (1988).

[52] C.C. Pettey, M.R. Leuze and J.J. Grefenstette, Genetic algorithms on a hypercube multiprocessor, *Proc. 2nd International Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 333–341 (1987).

[53] R. Tanese, Distributed genetic algorithms, *Proc. 3rd International Conference on Genetic Algorithms*, Arlington, VA, pp. 434–439 (1989).

[54] C.C. Pettey and M.R. Leuze, A theoretical investigation of a parallel genetic algorithm, *Proc. 3rd International Conference on Genetic Algorithms*, Arlington, VA, pp. 398–405 (1989).

[55] B. Manderick and P. Spiessens, Fine-grained parallel genetic algorithms, *Proc. 3rd International Conference on Genetic Algorithms*, Arlington, VA, pp. 428–433 (1989).

[56] R. D. Ham, et al. *Group technology, applications to production management*, Kluwer Nijhoft Pub., Boston-Dordercht-Lancaster (1985).

[57] R. Das and D.E. Goldberg, Discrete-time parameter estimation with genetic algorithms, Preprints from *Proc. 19th Annual Pittsburge Conference Modelling and Simulation* (1988).

APPENDICES

APPENDIX A

Min/Max Based GA Fitness Evaluator

The Min/Max approach was developed by Osyczka [16] and was applied to multi-objective criteria of modeling and analysis of process planning by [15, 17]. This approach considers criteria of equal importance. In this method, separately attainable minima of each objective are first found using some optimization technique. The solution that consists of the minimum deviation, among all other solutions, is the best compromise solution. Relative deviations are calculated as follows [15]:

$$Z_i(\bar{x}) \;=\; | f_i(\bar{x}) - \dot{f}_i | \,/\, | \dot{f}_i | \tag{A.1}$$

or

$$Z_i(\bar{x}) \;=\; | f_i(\bar{x}) - \dot{f}_i | \,/\, | f_i(\bar{x}) | \tag{A.2}$$

where,

$\dot{f}_i$ represents the separately attainable optima.

$f_i(\bar{x})$ represents a given solution.

Equation (A.1) define relative increments of a minimized function. Equation (A.2) operates conversely.

The Min/Max approach finds the least increment/decrement of the vector:

$$Z(\bar{x}) \;=\; \{Z_1(\bar{x})\; Z_2(\bar{x}) \ldots Z_k(\bar{x})\} \tag{A.3}$$

where, k is the number of objectives.

In order to accommodate the application of GA to find the best compromise solution of the network, the Min/Max formulation is modified in this work. Each

separately attainable optimum is obtained by using the formulation described in chapter 3. Either one of the above two equations (depending on the type of the multi-objective optimization problem) is then applied. The mapping of the objective function to its equivalent fitness function counter part is shown below:

$$Y_{j(j=1,2,...,p)} = U \; - \; \sum_{k=1}^{l} \omega_k \; Z_k(\bar{x}) \tag{A.4}$$

where,

Y_j is the fitness function for individual j,

$Z_k(\bar{x})$ is as described earlier,

U is an upper/lower bound on the maximum or minimum value of all objectives,

p is the population size,

l is the number of objectives,

$\omega_k > 0$ is the weighting factor associated with each objective criterion and is determined based on its degree of importance in relation to the other objective criteria.

There are two distinct differences between this modified Min/Max approach and the classical one: the first is that we do not find the minimum of the maximum deviation of each objective from the separately attainable optimum. Instead, we calculate the deviations from the optima and find the contribution of their total outcome to the fitness function. In other words, the deviations are summed and an upper value of the optima is reduced by that amount in order to find the actual contribution of each objective in the form of a fitness function. This fitness function is a metric to evaluate the performance of each individual path in the network based on its contribution towards its constituent objectives.

The second difference is that a weighting factor can be assigned to each objective criterion by the decision maker. This trait allows for handling both equally and non-equally important objectives.

The method presented in this Appendix requires additional computational overhead cost in the form of finding the separate optima for each objective. However, It can be readily applied to solve large size network problems, if specialized single objective optimization techniques such as Dejkstra's algorithm were available.

APPENDIX B

Two-objective Fitness Evaluator.

B.1 A Minimum Cost Minimum Time Case.

The same network shown in Figure 3.10 (in Chapter 3, Section 3.4) is used here for the bi-criteria analysis. A different input data of the second objective that reflect a conflicting nature with the first objective is shown in Figure B.1. This figure depicts the same stage flow network that is discussed in Chapter 3 (Section 3.4). The second objective, however, is set-up to reflect a conflict with the first objective (the bi-criteria case); It is generated by subtracting the largest value of the first objective at each stage (plus one) from all the first objective values at that stage, this is done throughout the network stages.

The fitness function that describes the behavior of a bi-criteria network is described here.

$$Y_{j(j=1,2,...,p)} = U - Z_j$$

$$where\ Z_j\ is$$

$$Z_j = \sqrt{\sum_{i=1}^{l} \omega_i \{\{C_i - Min_i\}/\{Max_i - Min_i\}^2\}} \qquad \text{(B.1)}$$

where,

Y_j is the fitness function for individual j,

C_i is the kth minimization (conflicting) objective value associated with link i,

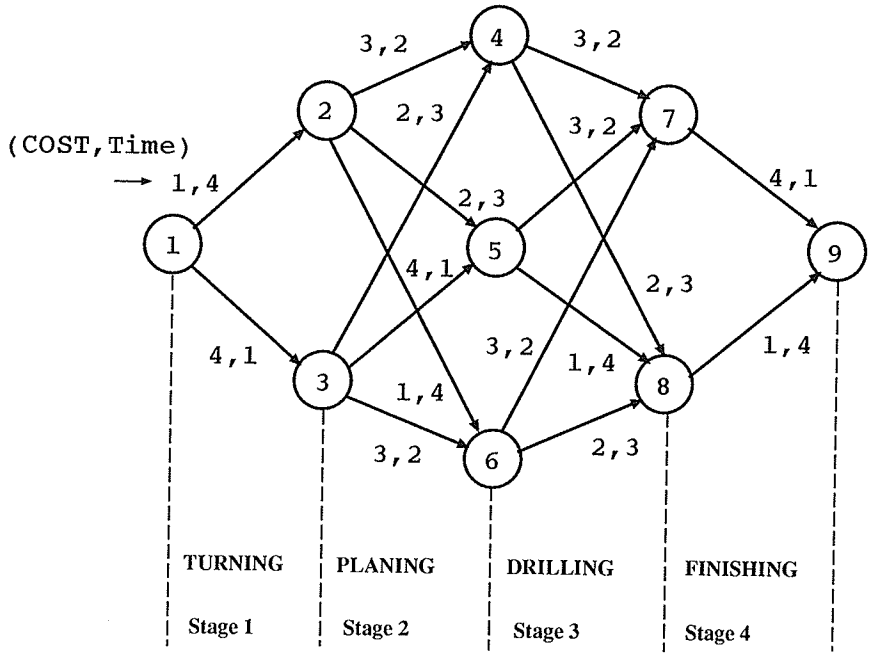


Figure B.1: Bi-criteria flow network.

Z_j is the deviation from both of the separately optima,

U is an upper bound of one objective,

l=2 is the number of conflicting objectives,

$0 \leq \omega_i \leq 1$ is the weighting factor,

Max_i is the maximum value of individual j of the i objective,

Min_i is the minimum value of individual j of the i objective,

The U value is calculated by summing the largest values of one objective, or the other, at each stage of the network across all stages. The Max_i and the Min_i are calculated in a similar fashion across the network for both objectives, i.e. summing the largest values at each stage for the first (Max_i), and summing the minimum values for the second value (Min_i). The deviation of paths from their separately

attainable optima is first calculated, i.e. the Z term in Equation (B.1). In this method, however, the separate optima need not be calculated; instead the Min_i values of both objectives were found by summing the minimum values of each objective at every stage over the entire network stages (as was described in Chapter 3 for calculating the U values in the formulation presented there). This calculation (approximation of the separate optima) yielded the same results as the ones obtained by using the separate optima. The calculation of the separate optima is substantially more time consuming than calculating the Min_i of these objectives (as described above).

The fitness value of each individual is calculated by subtracting the deviation value of each path from a large value, i.e. U, as shown above. Obviously the less a path deviates from its optimum value, at each optimum, the better the fitness value of that path.

Table B.1 displays all the paths of Figure B.1 in the first column. The second column shows the actual sequence of paths in that network. The last column shows the solution values obtained separately for both objectives, i.e. cost and time.

Table B.2 depicts the results of applying the formulation of Equation (B.1). Four different weights (degree of importance) have been assigned to both objectives of Figure B.1. The result of applying Equation (B.1) is shown for each case in the "fitness columns" of Table B.2. The ranking of each case is also shown in the "rank" columns of that table. it is seen that the best compromise solution of each case reflects the importance attached to the two objectives, e.g. for the equally weighted case the fifth path is found to be the best compromise solution which is (by inspecting Table B.1) represents the middle point between the maximum and minimum values of both objectives. This is expected, since the best compromise solution of the bi-criteria case with equal weights must be the middle point.

Table B.1: Bi-criteria network parameters.

Path #	Sequence	Solution Values {Cost;Time}
1	$1 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 9$	{5;15}
2	$1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9$	{15;5}
3	$1 \rightarrow 2 \rightarrow 6 \rightarrow 8 \rightarrow 9$	{5;15}
4	$1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 9$	{13;7}
5	$1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9$	{10;10}
6	$1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9$	{14;6}
7	$1 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9$	{10;10}
8	$1 \rightarrow 3 \rightarrow 4 \rightarrow 8 \rightarrow 9$	{9;11}
9	$1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 9$	{11;9}
10	$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 9$	{7;13}
11	$1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 9$	{9;11}
12	$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9$	{10;10}

Table B.2: Bi-criteria network path evaluation with different weights.

Path #	Fitness values ($w_1 = 0.5$ $w_2 = 0.5$)	Rank	Fitness values ($w_1 = 0.3$ $w_2 = 0.7$)	Rank	Fitness values ($w_1 = 0.1$ $w_2 = 0.9$)	Rank	Fitness values ($w_1 = 0.9$ $w_2 = 0.1$)	Rank
1	{14.292 }	5	{14.164 }	6	{14.052 }	7	{14.683 }	1
2	{14.292 }	5	{14.453 }	4	{14.683 }	2	{14.052 }	7
3	{14.292 }	5	{14.164 }	6	{14.052 }	7	{14.683 }	1
4	{14.410 }	3	{14.632 }	1	{14.683 }	2	{14.238 }	5
5	{14.500 }	1	{14.500 }	3	{14.500 }	4	{14.500 }	3
6	{14.359 }	4	{14.500 }	3	{14.700 }	1	{14.145 }	6
7	{14.500 }	1	{14.500 }	3	{14.500 }	4	{14.500 }	3
8	{14.490 }	2	{14.453 }	4	{14.417 }	5	{14.576 }	2
9	{14.490 }	2	{14.531 }	2	{14.576 }	3	{14.417 }	4
10	{14.410 }	3	{14.322 }	5	{14.239 }	6	{14.683 }	1
11	{14.490 }	2	{14.453 }	4	{14.417 }	5	{14.576 }	2
12	{14.50 }	1	{14.500 }	3	{14.500 }	4	{14.500 }	3

The rest of the weights show a predicted bias towards the respective objective that possesses the highest weight, e.g. the rest of the columns in Table B.2. A larger size bi-criteria example is shown at the end of this section.

B.2 A Minimum Cost Maximum Quality Case

We have presented, in Chapter 3 (Section 3.4), a multi-objective approach to handle two-objective network problems. Alternatively, for networks with bi-criteria objectives (of non-conflicting nature) a different solution technique, is introduced here, which is based on calculating a utility value (analogous to the fitness values) associated with a path.

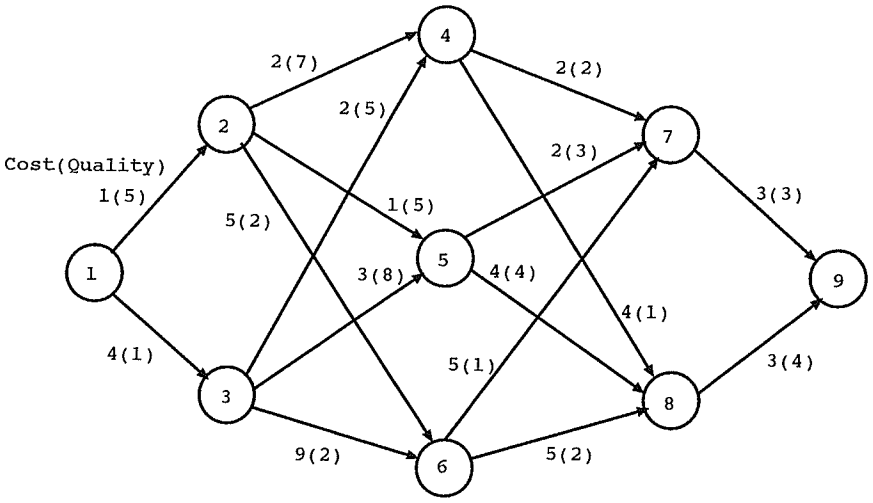


Figure B.2: Typical multi-objective manufacturing flow network.

Referring to Figure B.2, in order to establish a utility value, linear membership functions are set up with regard to their respective cost and quality objectives as shown in Figure B.3. These functions are of the following forms:

$$I_{cost} = |\frac{f_c(x) - f_{c0}}{f_{c1} - f_{c0}}| \tag{B.2}$$

$$I_{quality} = |\frac{f_q(x) - f_{q0}}{f_{q1} - f_{q0}}| \tag{B.3}$$

where the suffix 1 represents the maximum attainable value of the objective function and 0 represents the minimum permissible value of the objective function. The multi-objective problem can then be formulated as a *Max-Min* optimization problem as follows:

$$Max \ \{Min \ \ \mathcal{G}(I_{cost}, I_{quality})\} \tag{B.4}$$

where $\mathcal{G}$ is the grade of membership function. As an example, consider path $1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 9$ (Figure B.2) having total cost of 8 and quality of 17. The indices associated with these goals (i.e., 0.066 and 0.714 , respectively) are first calculated using Equation (B.2) and Equation (B.3) and are plotted using Figure B.3 by using Equation (B.4). The intersection of the indices yields a particular utility value that ranges between 0 to 0.5 (e.g., 0.468 in Figure B.3). The utility value is a compromise solution of equally weighted objective criteria. The goal is to find a solution point that has the highest utility value.

Applying the same procedure to all possible paths of the network shown in Figure B.2, one can form Table B.3. The first four columns of this table are similar to the ones in Table 3.1 (in Chapter 3, Section 3.4). The fifth column depicts the normalized values of both objective criteria that are calculated from Equations (B.2 and B.3). The sixth column shows the utility values. It is seen that individuals number 11 and 12 are ranked low according to the utility function, and thus will be removed from the population pool when GA operators are implemented.

The solution found by this method is, as expected, similar to the one resulting from the multi-objective formalism. However, the bi-criteria method resulted in fitness values all within the range of 0.0 to 0.5, whereas the limit in which the other method would evaluate the individuals can vary. It is believed that this difference

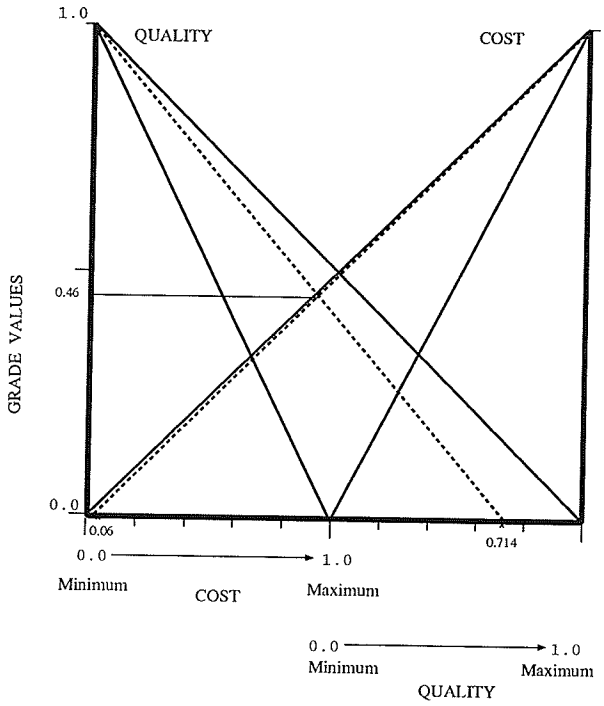


Figure B.3: Construction of utility values.

may alter the speed in which the algorithm converges to the optimal solution. This effect has not yet been evaluated.

Table B.3: Network path evaluation.

Path	Sequence	First Object. (cost)	Second Object. (quality)	Solution point (normalized)	Member-ship Grade	Rank
1	$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 9$	10	17	{ 0.20;0.714 }	0.421	3
2	$1 \rightarrow 2 \rightarrow 4 \rightarrow 7 \rightarrow 9$	8	17	{ 0.066;0.714 }	0.468	1
3	$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 9$	7	16	{ 0.0;0.642 }	0.450	2
4	$1 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 9$	9	18	{ 0.133;0.785 }	0.468	1
5	$1 \rightarrow 2 \rightarrow 6 \rightarrow 8 \rightarrow 9$	14	13	{ 0.466;0.428 }	0.32	7
6	$1 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 9$	14	11	{ 0.466;0.285 }	0.276	9
7	$1 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 9$	11	11	{ 0.266;0.285 }	0.33	6
8	$1 \rightarrow 3 \rightarrow 4 \rightarrow 8 \rightarrow 9$	13	11	{ 0.40;0.285 }	0.298	8
9	$1 \rightarrow 3 \rightarrow 5 \rightarrow 7 \rightarrow 9$	12	15	{ 0.333;0.571 }	0.40	4
10	$1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 9$	14	17	{ 0.466;0.714 }	0.389	5
11	$1 \rightarrow 3 \rightarrow 6 \rightarrow 7 \rightarrow 9$	21	10	{ 0.933;0.214 }	0.12	10
12	$1 \rightarrow 3 \rightarrow 6 \rightarrow 8 \rightarrow 9$	21	9	{ 0.933;0.142 }	0.093	11

APPENDIX C

GA Evolution of the Initial Generation

Figures C.1 through C.9 depict a typical evoluation process of the example dicussed in Chapter 3 (Section 3.4).

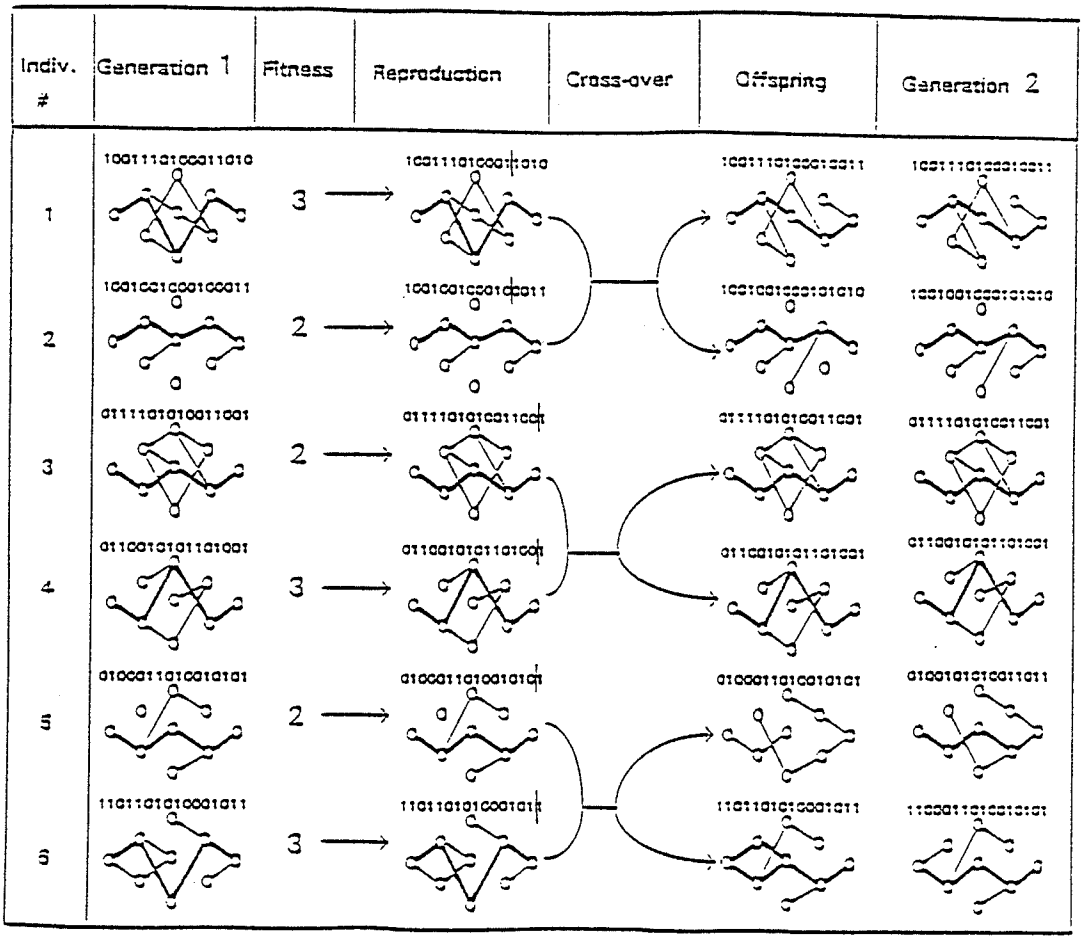


Figure C.1: Evolution of a generation treated by GA operators; generation no. 2

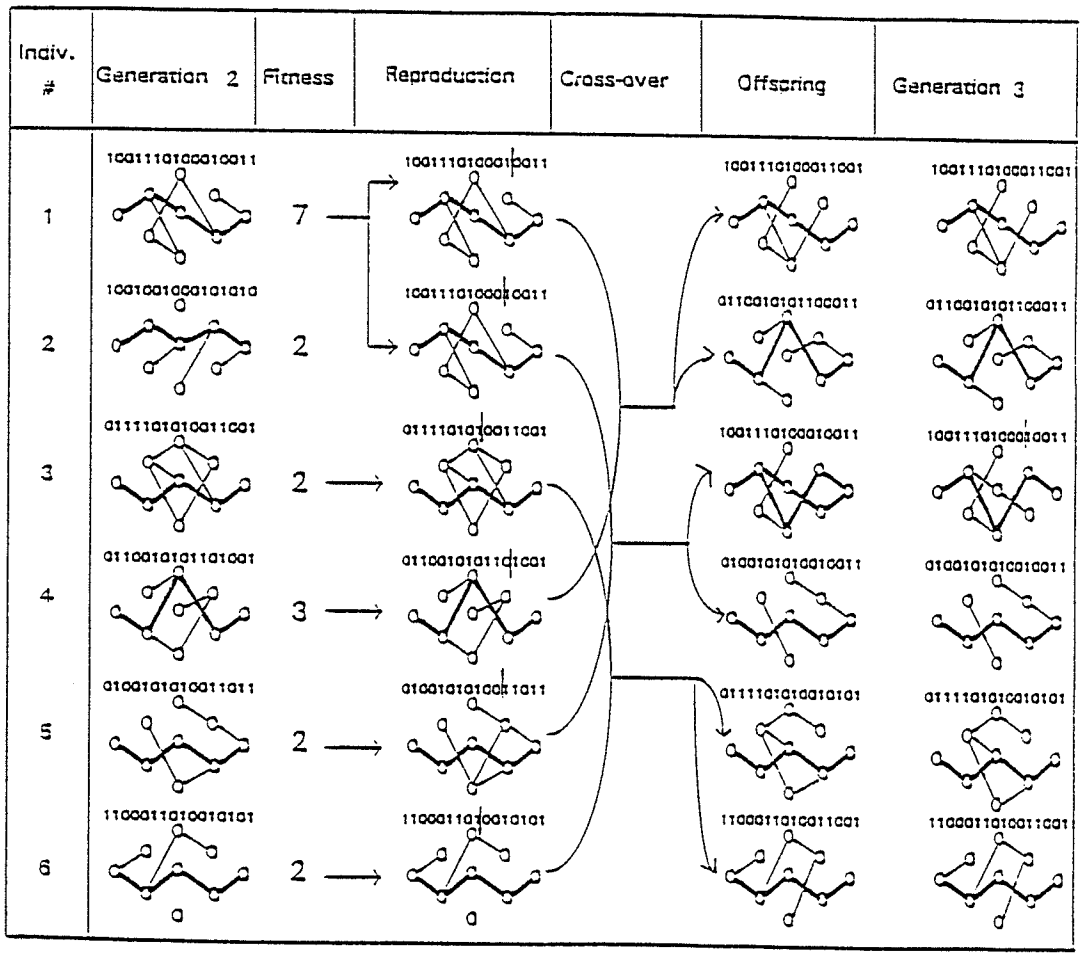


Figure C.2: Evolution of a generation treated by GA operators; generation no. 3

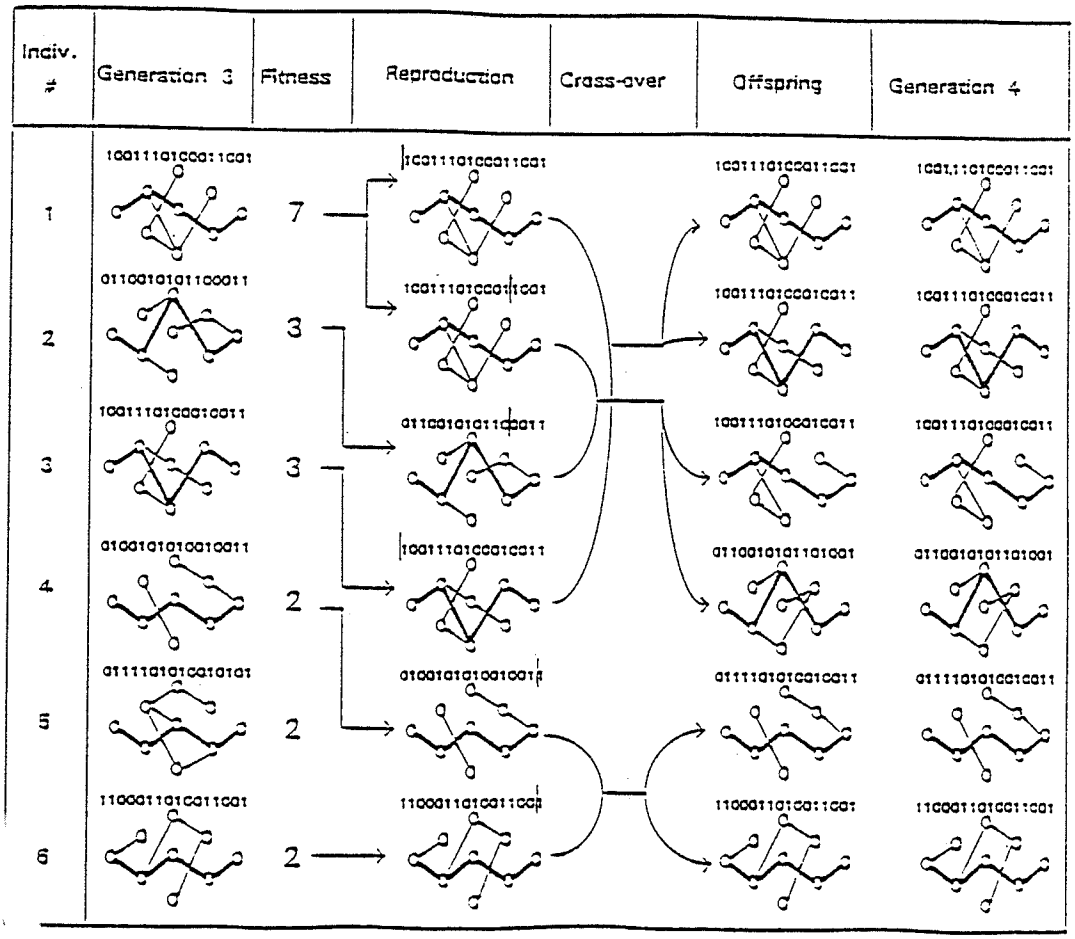


Figure 3.3: Evolution of a generation treated by GA operators; generation no. 4

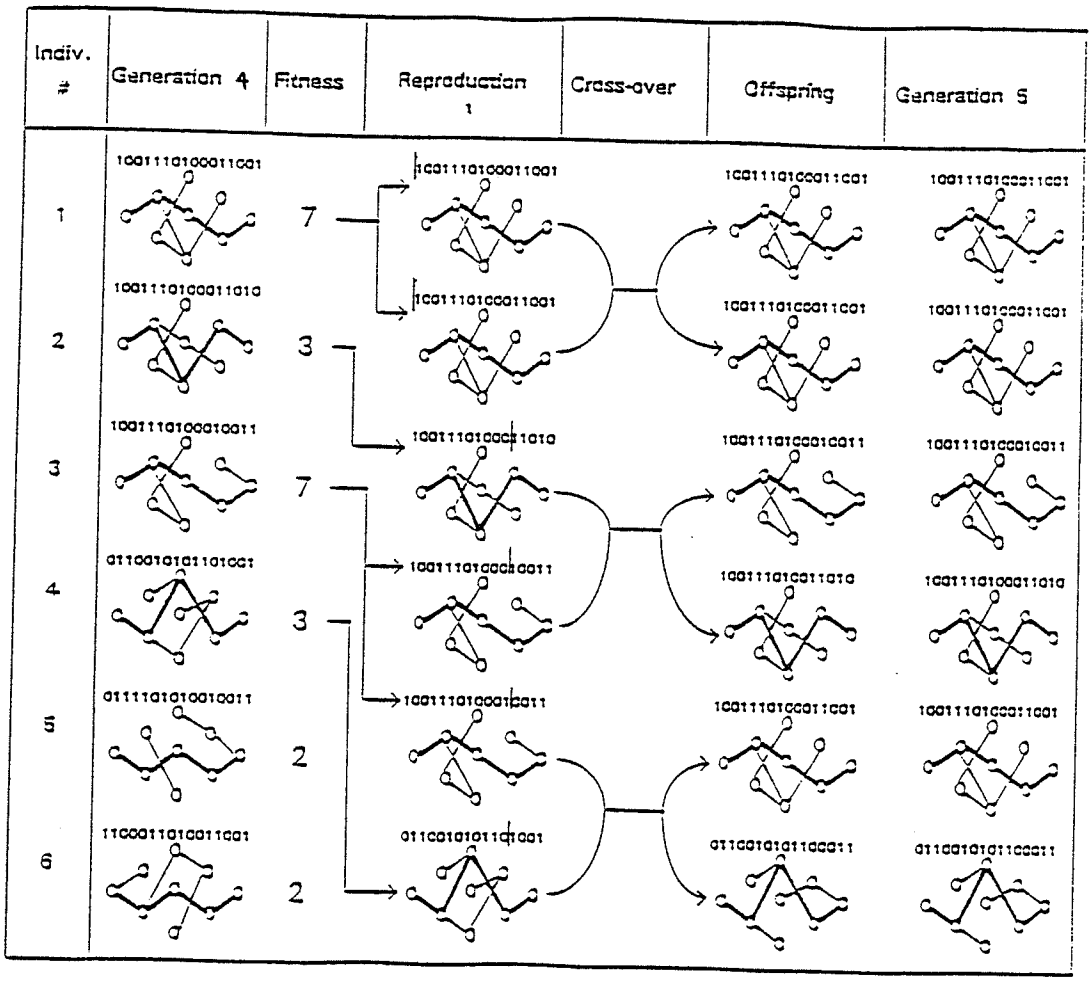


Figure C.4: Evolution of a generation treated by GA operators; generation no. 5

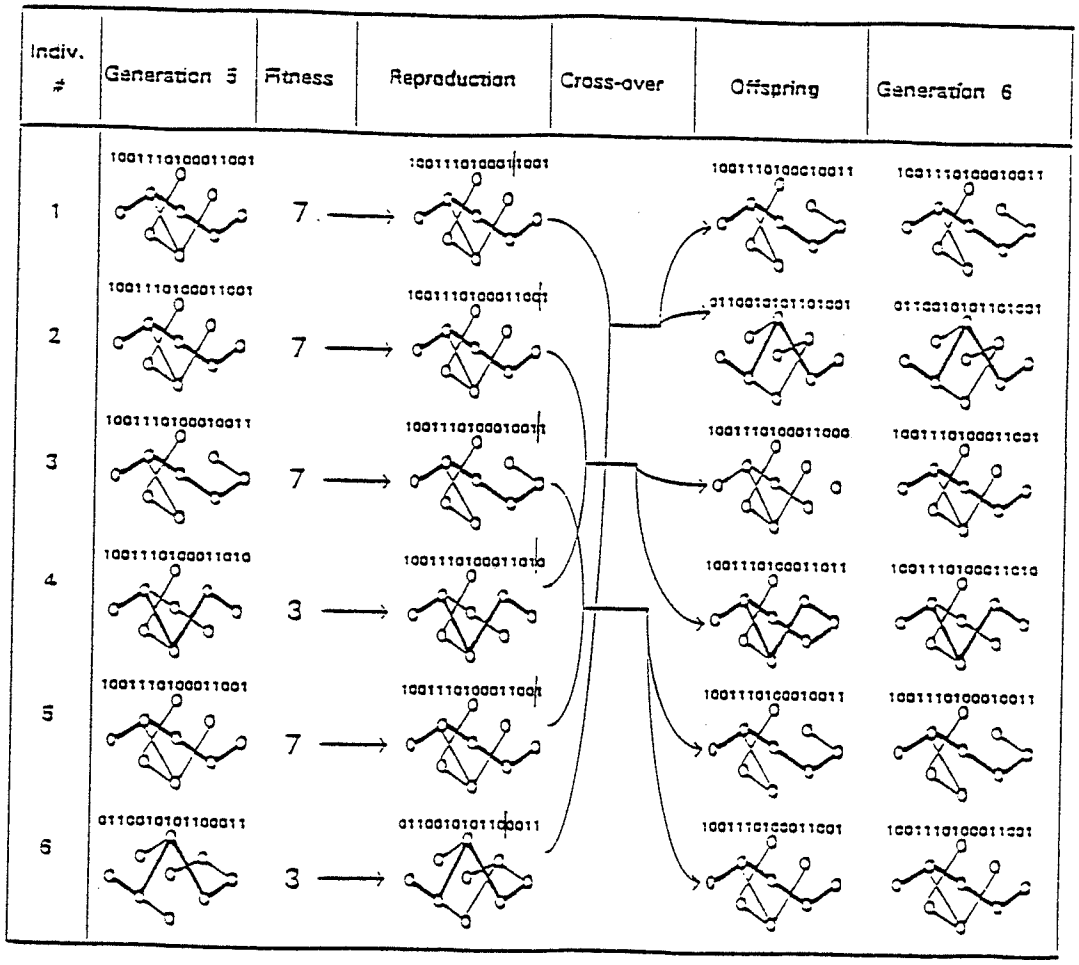


Figure C.5: Evolution of a generation treated by GA operators; generation no. 6

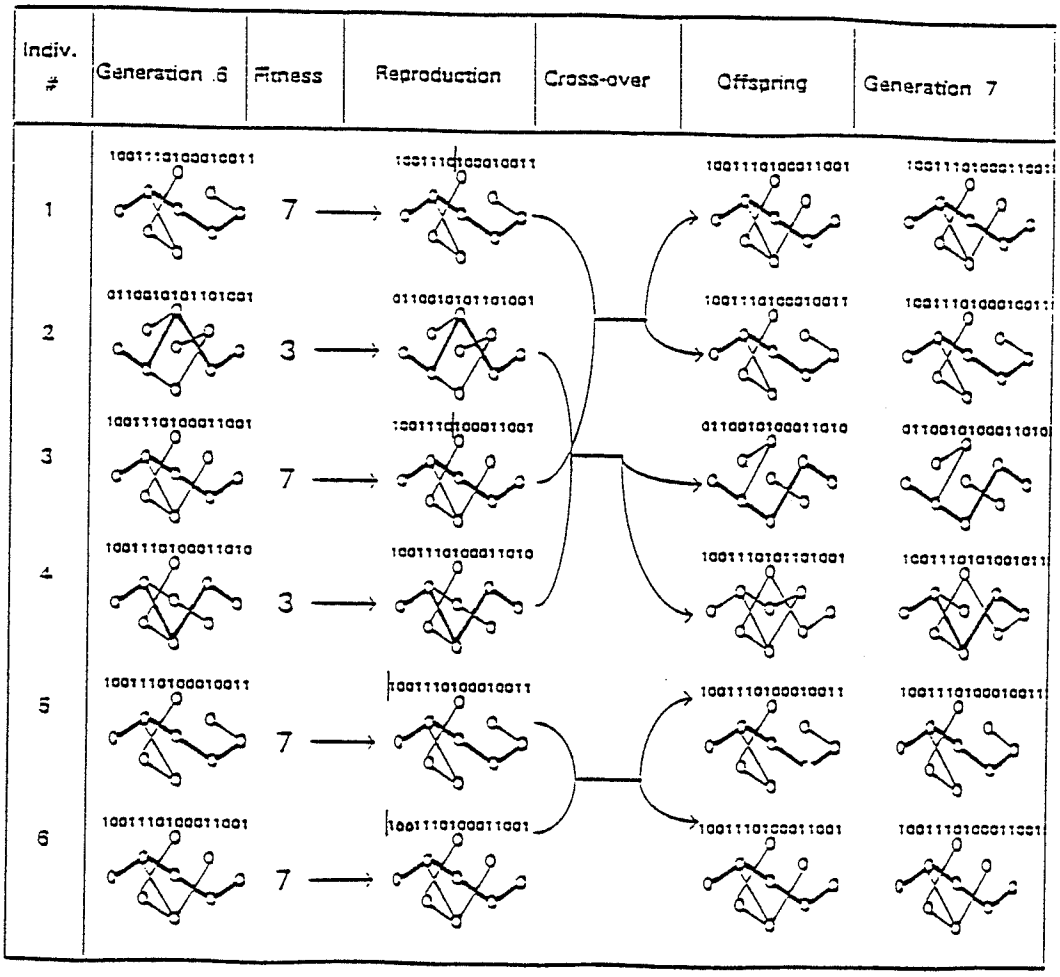


Figure C.6: Evolution of a generation treated by GA operators; generation no. 7

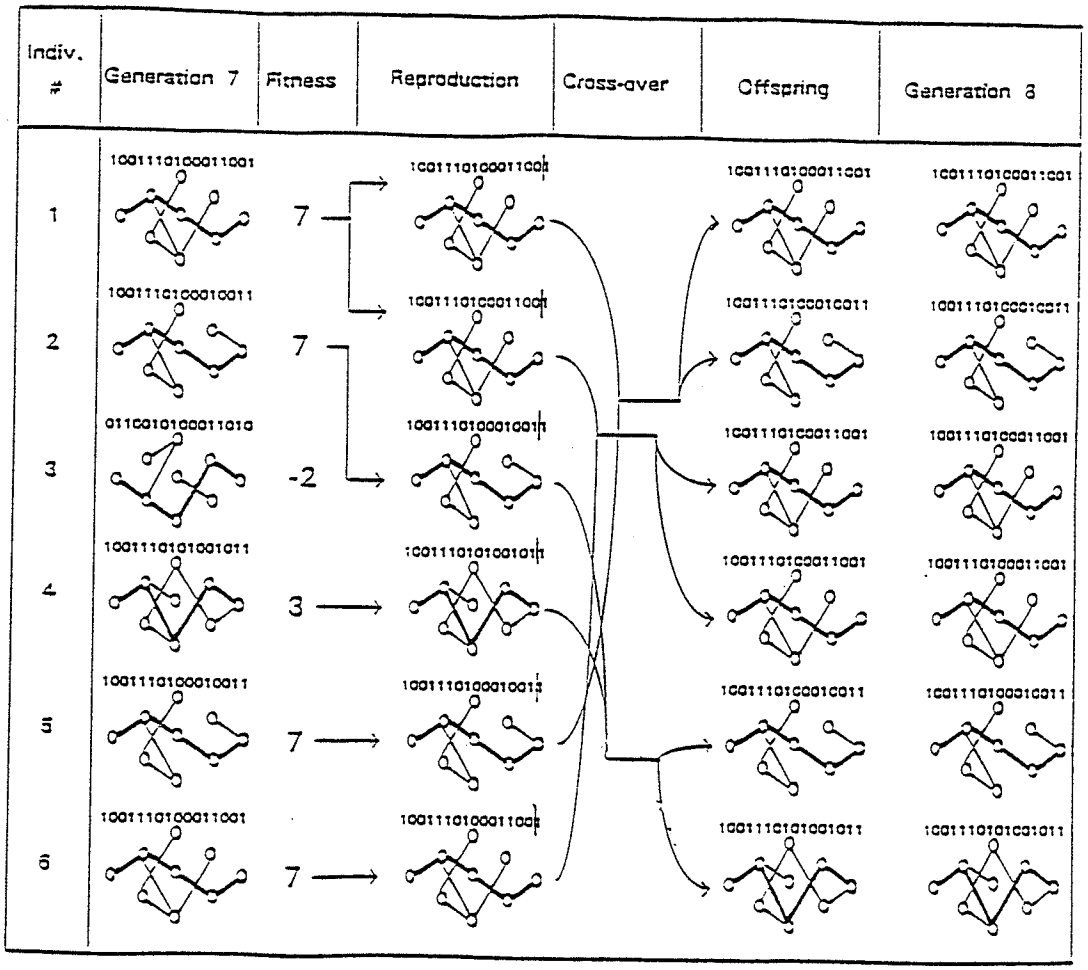


Figure C.7: Evolution of a generation treated by GA operators; generation no. 8

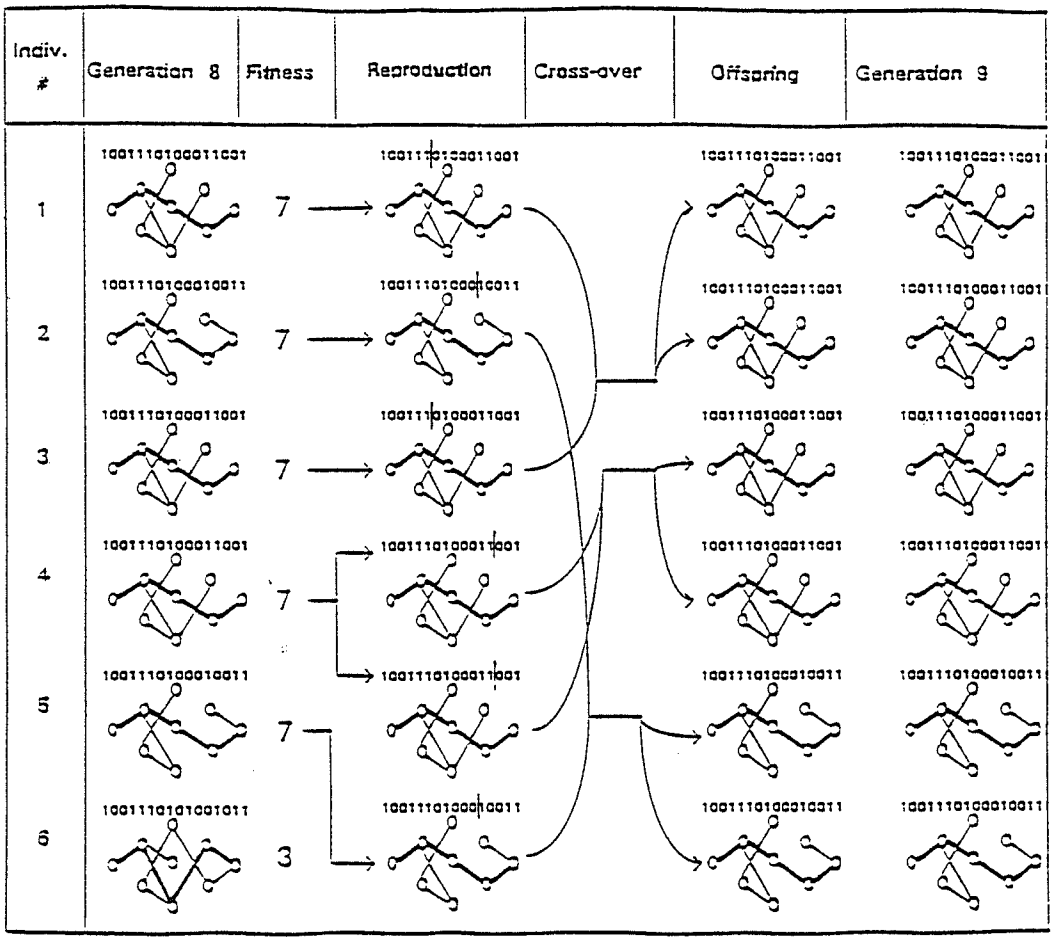


Figure C.8: Evolution of a generation treated by GA operators; generation no. 9

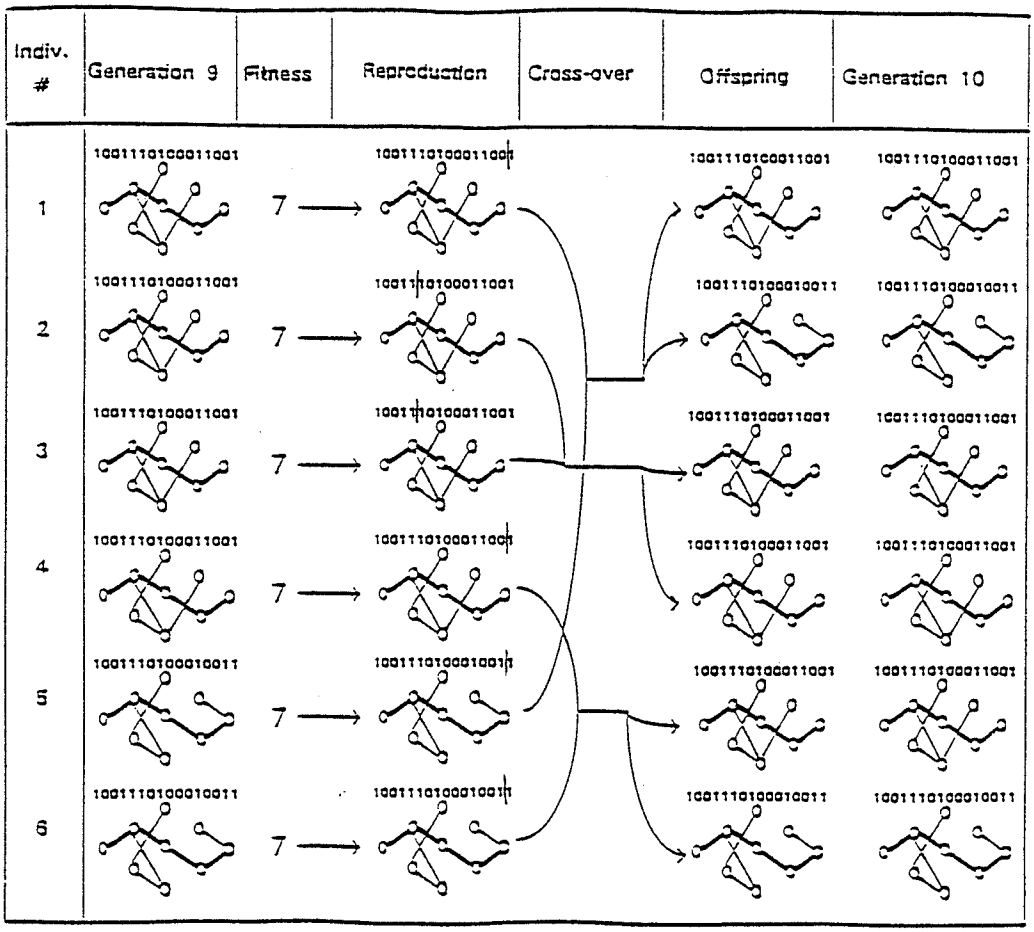


Figure C.9: Evolution of a generation treated by GA operators; generation no. 10

APPENDIX D

Compromise Solution Pool Input Parameters

D.1 Compromise Solution Pool

In all of the following examples, there is one set of two rows that represents an obtained compromise solution; the last example, however, has three rows because of the size of the binary solutions which caused a rounding off to the third row. The compromise solution values of the fitness function are depicted in the first element of the first row. The set of these values for all compromise solutions can be used to determine the order of the quality of these solutions (the ranking of these solutions). This, can be simply carried out by sorting these solutions in decreasing order where the best compromise solution is the one at the top of this order, the second best follows, and so on. The samples, shown in this Appendix, are not sorted out according to their ranks, they can be easily sorted if needed. The second value, in the first row, is the actual compromise solution points where both values of the two criteria are depicted.

The second row of each solution point depicts the compromise solution (path) as described in the text (the set theoretic representaion).

A sample of Pareto (compromise) solutions for set theoretic and stage matrix approaches applied to all 10 examples, that are shown in Table 3.2 in the text, are displayed here.

Pareto Solutions for Example Number 1, Chapter 3 (Section 3.5.2)

0.723166 {42.000000; 122.000}

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

0.727386 {36.000000; 117.000}

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

0.669884 {63.000000; 130.000}

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$

0.727386 {36.000000; 117.000}

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

The rest of the solutions of this example and the other 9 examples are supplied in a computer disk.

D.2 Networks Input

In this section, the network flow parameters and configurations are displayed. We show here the first network example of the 10 examples that are tested in the text (Chapter 3, Section 3.5.2 and Chapter 4, Section 4.3.2) for both stage matrix and stage set methods.

The first value denotes the number of stages of the network. Next, the number of input nodes to each stage is shown, starting at the source node (1) and ending at the sink node (24). After that the total number of nodes is shown. A set of four values is displayed for every arc connection between two nodes. The first two values denote the arc connection between their respective nodes, e.g. $1 \rightarrow 2$, this arc carries a cost value of 7 and a quality (or profit) value of 1. The rest of the arcs are similarly depicted; a total of 80 arcs are shown.

The network comprises 24 nodes, 80 arcs, and 7 stages. The input nodes to each stage are as follows: 1,3,5,4, 6,2,2. The rest of the input data comprises the arc input (the first two entries of the rows, e.g. $2 \rightarrow 5$) and the corresponding two

objective values (e.g. 2 and 8 for that entry). The rest of these entries are included in the computer disc.

```
1 2 7 1
1 3 11 3
1 4 14 6
2 5 2 8
2 6 8 8
2 7 5 29
```

The rest of data of this example and the other 9 examples input is available in a separate computer disk.

APPENDIX E

Computational Complexity Analysis

E.1 Complexity Analysis of Stage Matrix Method

The following outline shows a series of matrix multiplications of all stage matrices which represent the various stages of flow networks (depicted here for clarity).

$$\underbrace{\left(1 \times n_1\right)\left(n_1 \times n_2\right)}\left(n_2 \times n_3\right) \ \ldots\ldots\ldots \left(n_{k-1} \times n_k\right)\left(n_k \times 1\right)$$

$$\underbrace{(1 \times n_2)} \times \quad \uparrow$$

$$\underbrace{(1 \times n_3)}$$

.....

$$\underbrace{(1 \times n_{k-1})} \times \quad \uparrow$$

$$(1 \times 1)$$

These multiplications are carried out one pair at a time, where the first matrix represents the product matrix that is obtained by multiplying the previous stage matrix (or the product matrix) with the following stage matrix. Obviously, the very first pair of matrices are the first and second stage matrices of the network. These multiplications are carried out in that fashion until the final outcome matrix of "one" dimension is reached. If the value of this matrix is greater than or less than "one" then the modification procedure is applied, followed by the application of GA operations. Otherwise, for the single path outcome case, the path is processed by

GA operations alone. The matrices, shown above, describe the input and output nodes of each stage (e.g. n_1 and n_2 in stage 2) in a network of $k+1$ stages. We discuss next the computational complexity of the modification procedure and GA operations that are employed to find the best path in the flow networks. These networks are represented as stage matrices as shown in the first line of the outline of matrices.

In order to carry out a computational complexity analysis of the stage matrix procedure, the following terms are first described in reference to Figure E.1. Figure E.1 shows a network of $k+1$ stages, n_i nodes per-stage, each input node is connected to every output node of all stages, source and sink nodes.

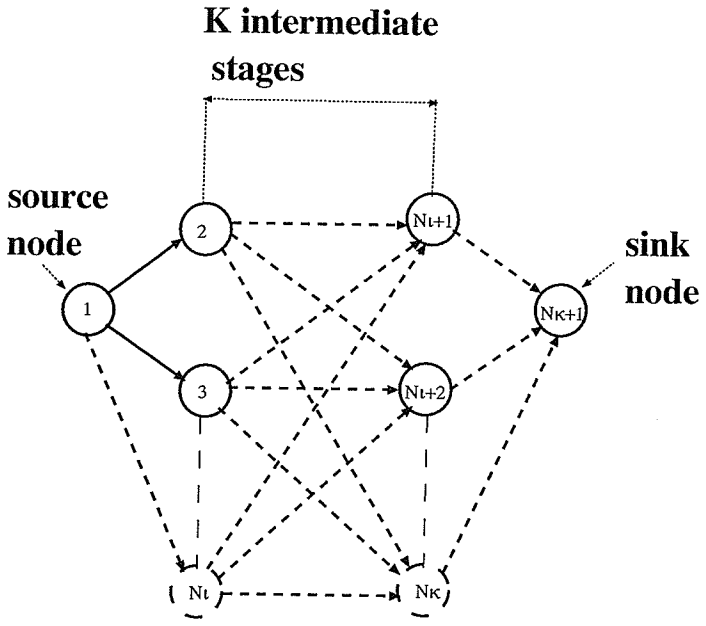


Figure E.1: Stage-based network representation.

The maximum required operations in a given network problem is directly related to the number of nodes of that network. In order to find an estimate of the maximum number of nodes in these networks the following derivations are carried out. The maximum number of nodes is an important parameter in regard to estimating the

computational complexity of any algorithm that is used to solve any given network problems.

The maximum distribution of nodes at each stage is calculated as follows:

$$Maximize \qquad (n_i \quad n_{i+1}) \tag{E.1}$$

Now suppose $M = n_i + n_{i+1}$, i.e. the total number of nodes in every stage is equal to M.

$n_{i+1} = M - n_i,$

$n_i n_{i+1} = n_i(M - n_i),$

$n_i n_{i+1} = (-n_i)^2 + M \times n_i,$

by differentiating for n_i

$\frac{d}{dn_i}(n_i \quad n_{i+1}) = -2n_i + M,$

The maximum occur when the differential is equal to zero:

$M = 2n_i,$

Therefore,

$n_i = M/2$ and $n_{i+1} = M/2,$ It can be shown that the maximum distribution of nodes across the entire network (in terms of stages), occurs when the output nodes are equal to the input nodes at all stages:

$$n_i = n_{i+1} = M/2 \tag{E.2}$$

The computational complexity of the matrix multiplications step and the rest of the modification procedure, described in chapter 3, is arrived at as follows:

$1 + n_1 + n_2 + n_3 + ... + n_k + 1 = n,$ where $k = number$ of stages -1 that must be greater than 2. The above formula can be easily proven by substituting $n_{1...k}$ by $(n-2)/k$.

Therefore the maximum number of expected computational operations can be

found from:

$$Maximize\ (n_1n_2 + n_2n_3.... + n_{k-1}n_k + n_k), and \tag{E.3}$$

$$n_1 + n_2 + n_3 + ... + nk + 2 = n \tag{E.4}$$

where n is the total number of nodes.

As shown earlier, in Equation (E.2), Equations (E.3 and E.4) are maximum when:

$$n_i \ = \ n_{i+1} \ = \ M/2 \ = \ (n-2)/k, \tag{E.5}$$

or, the maximum value occurs after substituting for n_i by $(n-2)/k$ as follows:

$$(k-1)((n-2)/k)^2 + 2 * ((n-2)/k) = (n-2)^2(k-1)/(k)^2 + (n-2)(2/k)$$
$$\simeq (n-2)^2/k = C(n-2)^2 \tag{E.6}$$

Equation (E.6) gives the maximum number of expected computational operations for any type of modification operations described in the previous chapter or for the identification procedure where all stage matrices are multiplied together.

A brief description of each procedure of the modification algorithm is presented next.

E.1.1 Complexity Analysis of the Path Corrector

The modification procedure is described, in detail, in chapter 3 (Section 3.3.2. Referring to the path identifier procedure in that section, the various steps described there are analyzed as follows:

The first step takes at most $O(Cn^2)$ time units, as proved earlier.

The second step takes O(1) time units , for each "if" statement.

The total expected number of time units is therefore, $O(Cn^2)$ for the entire procedure.

The Multiple Path Corrector procedure is analyzed as follows:

Step a: the matrix multiplications will be less than or equal $O(n^2)$; if the final product of matrices multiplication is one, then the expected time is equal to $O(n^2)$ time unit.

Step b: The identification of the matrix that caused an outcome of greater than one takes $O(1)$ time unit. Finding the column responsible for the multiplicity, and fixing that condition, take $O(1)$ and $O((n-2/k)^2)$ time unit respectively. The expected time is then found as follows:

Maximum $(O(1), O(1), O(((n-2)/k)^2)$, and the outcome is $O(((n-2)/k)^2)$ time units; or as before $O(Cn^2)$, where $C \simeq (1/k)$.

Step a is repeated at most k times, therefore, the worst time is $O(n^2)$ per call. Another constant number that affects the value of steps a and b is the maximum number of multiplicity of paths within each network. This number is a percentage of the total number of alternative paths that exist in the network. Since network paths are, initially, generated randomly, we expect the multiplicity of paths per single individual to be much lower than 50% of the total number of possible paths. This percentage can be substantially smaller in the succeeding generations. The expected growth rate of this procedure is thus $O(qn^2)$, where q is the percentage described above.

Finally, the Disconnected Path Procedure will require approximately the same amount of computational time as the previous Multiple Path Corrector procedure.

E.1.2 GA Procedure Complexity Analysis

The complexity analysis of the various modification procedures, described above, is done for a single individual network. Obviously, some factors must be taken into account in order to accommodate for the number of individuals per generation, i.e. the population size and the number of generations required by the GA. The population size, p, will give an upper bound on the previous growth rate calculation. The new growth rate is $O(qpn^2)$ time unit. Similarly, the number of generations (G) required to converge to an optimal solution gives another upper bound on the above growth rate; the new growth rate is $O(qpGn^2)$ time units. If we assume that qpG are some variable m, then the growth rate is $O(mn^2)$ time units.

The symbol m, in the above growth rate, can have a wide range of values depending on the size of network, the size of the selected population, the number of generations and so on. Also, during the application of the disconnected path procedure, it is possible that the final outcome (product matrix) may have a value greater than one. In this case, an application of the multiple path procedure is inevitable. This can occur, at worst, (k-2) times.

Since these and other GA parameters, such as the probability of cross-over, are stochastic in nature, a study of the CPU time for every problem must be conducted. CPU is the final metric for measuring the growth rate (computational complexity) of the GA procedures that are presented in this work. The total growth rate of the entire procedure is approximately $O((k-2)n^4)$, assuming that m would reach the n^2 in some rare occasions. Finally, the generation of a network, from a zero/one binary string, takes at most $O(n^2)$.

It can be easily shown that no other operation within the modification or GA operations will have a growth rate larger than $O((k-2)n^4)$, therefore, this growth rate is the final expected value that describes the computational complexity of the

entire computer package. The growth rate obtained above is also arrived at by a rigorous counting of every computer program operation of the actual implementation of the modification procedure along with the GA operations.

E.2 Complexity Analysis of Stage Set Method

The growth rate, or the computational complexity, of the modification algorithm is carried out first. An upper bound, on the application of this procedure, is then inferred from the expected GA parameters. In order to conduct a computational complexity analysis of the stage set procedure, the same terms that were used for the stage matrix (wherever applicable) are borrowed here. The maximum distribution of nodes across the entire network, occurs when the output nodes (sets) are equal to the input set of nodes for all stages (sets). This is proved in this Appendix (Section E.1) by Equation (E.5). In order to analyze the GA based modification procedure, we apply the same logic adopted in Chapter 3 and earlier in this Appendix. We start by calculating the number of operations required for the modification procedure, shown in step (2) of the algorithm chapter 4 (Section 4.2). In this step, the checking operation of a single set member (node) per-stage (set) takes $O(K)$. The sets n_1 through n_k time step calculation is shown below:

$$1 + \sum_{i=1}^{k} n_i + 1 \;=\; 1 + (n-2)_1/k + (n-2)_2/k + ... + (n-2)_k/k + \;1, \qquad \text{(E.7)}$$

or

$$2 + (n-2)/k \times k = n. \qquad \text{(E.8)}$$

Therefore, the growth rate of the checking operation, step (2), takes at most $O(n)$ time units. This is a linear relation of growth rate with the number of network nodes.

The first step of the algorithm gives an upper bound on the number of operations that are required to converge to an optimal solution. This bound is p, representing the population size. Another upper bound is the number of generations required for converging to an optimal solution; is the bound G.

The final growth rate of our computational analysis of the procedure is $O(npG)$ time units. This growth rate can approach n^3 in some rare occasions, when the population size and the generation number approaches n^2. However, it can be maintained at a much lower rate, some fraction over the square of n, by stopping at a near optimal solution, e.g. stopping at an earlier generation.

APPENDIX F

Solution Pool and Input Parameters

F.1 Compromise Solution Pool

In all of the following examples, there is one set of three rows that represents an obtained compromise solution; the examples have three rows because of the size of the binary solutions which caused a rounding off to the third row. The compromise solution values of the fitness function are depicted in the first element of the first row. The set of these values for all compromise solutions can be used to determine the order of the quality of these solutions (the ranking of these solutions). This, can be simply carried out by sorting these solutions in decreasing order where the best compromise solution is the one at the top of this order, the second best follows, and so on. The samples, shown in this Appendix, are not sorted according to their ranks; they can be easily sorted if needed. The second value, in the first row, is the actual compromise solution points where all values of criteria (2, 3 and 4 respectively) are depicted.

The second row (and the continuation to the third row) of each solution point depicts the the compromise solution (path) as described in the text (the set theoretic representaion).

A sample of Pareto (compromise) solutions for set theoretic approach applied to all 4 objective criteria examples, that are shown in Table 4.3 in the text (Chapter 4), are depicted here. A complete account of all the pareto solutions of all examlpes is given on the computer disc.

Pareto Solutions for the two objectives case, Chapter 4 (Section 4.4.2)

0.786834 {67.000000; 372.000000}

$1 \rightarrow 2 \rightarrow 6 \rightarrow 20 \rightarrow 35 \rightarrow 50 \rightarrow 62 \rightarrow 69 \rightarrow 75 \rightarrow 93 \rightarrow 102 \rightarrow 114 \rightarrow 120 \rightarrow 126 \rightarrow 128$

0.786655{57.000000; 356.000000}

$1 \rightarrow 2 \rightarrow 6 \rightarrow 20 \rightarrow 35 \rightarrow 47 \rightarrow 61 \rightarrow 68 \rightarrow 73 \rightarrow 93 \rightarrow 102 \rightarrow 114 \rightarrow 120 \rightarrow 126 \rightarrow 128$

0.764134{68.000000; 358.000000}

$1 \rightarrow 2 \rightarrow 5 \rightarrow 23 \rightarrow 35 \rightarrow 50 \rightarrow 62 \rightarrow 69 \rightarrow 75 \rightarrow 93 \rightarrow 99 \rightarrow 116 \rightarrow 120 \rightarrow 126 \rightarrow 128$

The rest of the solutions of this example and the rest of the examples are given on the computer disc.

F.2 Multi-Objective Networks Input

In the following section, the network flow parameters and configurations are displayed. These networks are the four examples that were tested in the text (Chapter 4) with stage set method. The input of the four objectives case is shown here. Since the same example is used for the other two cases (2 and 3 objectives), only one input file is shown here (for the sake of brevity). For the other two cases, the objectives, i.e. the last one and two objectives,respectively, of this input file are not introduced in their respective input files The constraint shortest path input was the same as for the two objective case. Finally, for the single objective case, only the first ojective values of this input file is used. The first value, in the following, denotes the number of stages of the network. Next, the number of input nodes to each stage is shown, starting at the source node (1) and ending at the sink node (128). After that the

total number of nodes is shown.

$$14, 1, 2, 12, 10, 4, 20, 4, 18, 6, 6, 2, 1, 128$$

A set of six values is displayed, below, for every arc connection between two nodes. The first two values denote the arc connection between their respective nodes, e.g. $2 \rightarrow 4$, this arc carries a cost value of 5, a quality value of 4, a time value of 8 and a distance value of 22. The first two objectives are the input parameters for the 2 objectives case. The first three objectives are the input parameters for the three objectives case. Finally, the all four objectives shown below are the input parameters of the four objective case; naturally the first objective is used for the single objective case. The rest of the arcs are similarly depicted; a total of 1160 arcs are shown.

```
1 2 0 30 0 0
1 3 15 1 15 30
2 4 5 4 8 22
2 5 2 29 2 17
2 6 4 30 5 18
2 7 11 27 12 28
2 8 12 17 11 15
2 9 6 16 20 4
2 10 15 1 20 30
2 11 15 1 20 30
2 12 15 1 20 30
2 13 15 1 20 30
2 14 15 1 20 30
```

2 15 15 1 20 30

3 4 15 1 20 30

3 5 15 1 20 30

3 6 15 1 20 30

3 7 15 1 20 30

3 8 15 1 20 30

3 9 15 1 20 30

3 10 15 1 20 30

3 11 15 1 20 30

3 12 15 1 20 30

3 13 15 1 20 30

3 14 15 1 20 30

3 15 15 1 20 30

4 16 1 12 10 1

4 17 6 8 18 17

4 18 1 17 4 22

4 19 5 8 3 18

4 20 15 15 9 12

4 21 13 25 2 18

4 22 15 28 4 23

4 23 3 27 10 5

4 24 15 1 20 30

4 25 15 1 20 30

4 26 15 1 20 30

4 27 15 1 20 30

4 28 15 1 20 30

```
4 29 15 1 20 30
4 30 15 1 20 30
4 31 15 1 20 30
5 16 10 16 13 10
5 17 6 6 17 28
5 18 10 2 13 13
5 19 4 16 4 2
5 20 9 8 2 26
5 21 4 12 19 27
5 22 10 2 4 16
5 23 15 26 2 18
```

The rest of this input file is shown in a computer disc.

APPENDIX G

More Case Studies

G.1 Multi-Optima Case Study

The example shown in Figure 4.3 has been altered so that two paths of optimal value equal to 7 are present. These paths (individuals) are allowed to mate during the course of applying the genetic algorithm. The convergence behavior is shown in Figure G.1. Another case was also attempted in this work where the two optimal paths were not allowed to mate during the application of crossover. The convergence behavior of this case is shown in Figure G.2. The results of applying the stage set method to find the best solution(s) for both of these cases are shown in Table G.1. The frequency of finding both of these optima for both cases is shown in that table. The two optima described in Table G.1 are:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 14 \rightarrow 20 \rightarrow 22 \rightarrow 24$

$1 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 14 \rightarrow 20 \rightarrow 22 \rightarrow 24$

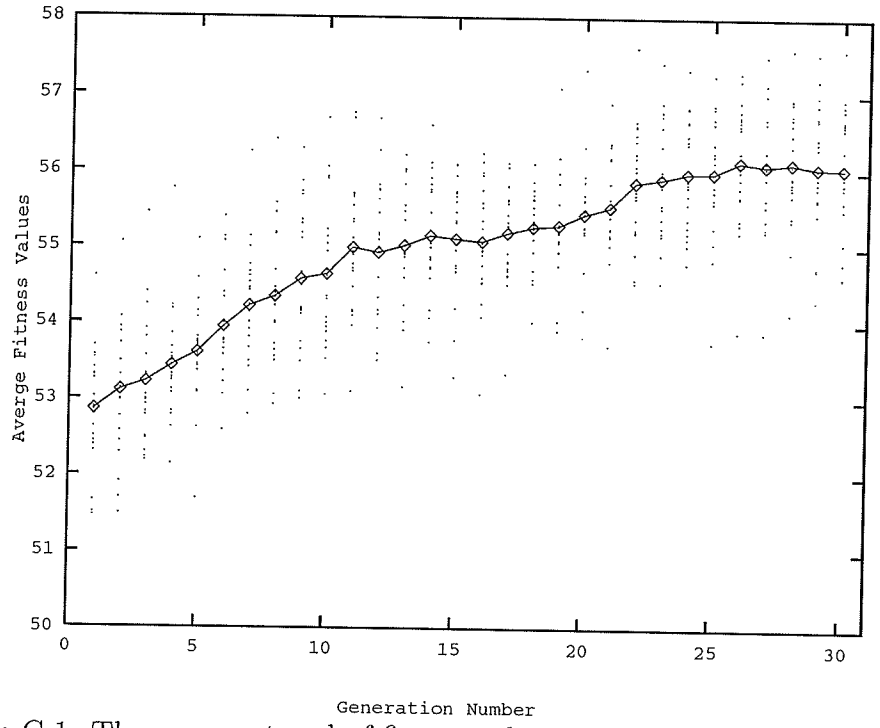


Figure G.1: The average trend of fitness values ; with mating of optima.

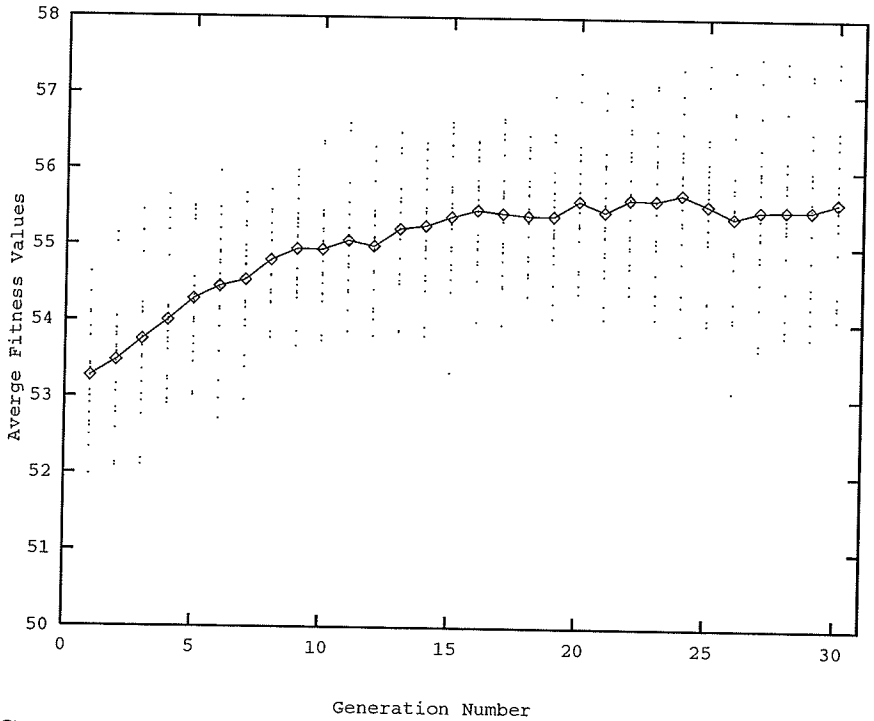


Figure G.2: The average trend of fitness values ; without mating of optima.

Table G.1: Results of the multi-optima case.

Generation #	No. of optima Mated		No. of optima Un-Mated	
	Opt. No. 1	*Opt. No. 2*	*Opt. No. 1*	*Opt. No. 2*
0	0	0	1	0
5	3	1	1	1
10	5	5	1	1
15	3	7	1	3
20	1	12	2	3
25	2	12	1	1
30	2	14	2	3

Table G.2: Multi-objective results; subject to various weights

Case #	Best Path	Weights $\{w_{cost};\ w_{quality}\}$	Fitness Values	Compromise Solutions $\{Cost;\ Quality\}$
1	$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$	{0.1;0.9}	0.889	{84;144}
2	$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.2;0.8}	0.813	{57;135}
3	$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.3;0.7}	0.764	{57;135}
4	$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.4;0.6}	0.727	{36;117}
5	$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.5;0.5}	0.711	{36;117}
6	$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.6;0.4}	0.701	{32;110}
7	$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.7;0.3}	0.694	{32;110}
8	$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.8;0.2}	0.687	{32;110}
9	$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$	{0.9;0.1}	0.700	{28;85}

G.2 Multi-objective Case Study with Various Weights

The first example shown in Table 4.2 is further treated here for several cases where various cost and quality weights are used. The results of applying the stage set model is shown in Table G.2.

The compromise solution pool for the first case of Table G.2 is shown below where the first row shows the fitness value and the compromise solution of each path. the second row shows the actual path:

$0.889961\{84.000000; 144.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$

$0.889961\{84.000000; 144.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$

$0.889961\{84.000000; 144.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$

$0.855640\{64.000000; 135.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$

$$0.855640\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.855640\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.855640\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.855640\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.799035\{42.000000; 122.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.806950\{70.000000; 128.000000\}$$

$$1 \rightarrow 4 \rightarrow 9 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

The compromise solution pool for the second case of Table G.2 is shown below:

$$0.813403\{57.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.813403\{57.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.799118\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.799118\{64.000000; 135.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.790844\{76.000000; 138.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.773635\{50.000000; 125.000000\}$$

$1 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.762934\{42.000000; 120.000000\}$

$1 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.762934\{42.000000; 120.000000\}$

$1 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.752124\{42.000000; 118.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.749531\{83.000000; 133.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 22 \rightarrow 24$

The compromise solution pool for the third case of Table G.2 is shown below:

$0.764024\{57.000000; 135.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.764024\{57.000000; 135.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.764024\{57.000000; 135.000000\}$

$1 \rightarrow 4 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.742333\{44.000000; 122.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.742333\{44.000000; 122.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.748456\{42.000000; 122.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.748456\{42.000000; 122.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$$0.744264\{65.000000; 136.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.742333\{44.000000; 122.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.745105\{57.000000; 131.000000\}$$

$$1 \rightarrow 4 \rightarrow 8 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

The compromise solution pool for the fourth case of Table G.2 is shown below:

$$0.727386\{36.000000; 117.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.727386\{36.000000; 117.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.723166\{42.000000; 122.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.669884\{63.000000; 130.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$$

$$0.682846\{34.000000; 104.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 18 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.662493\{37.000000; 102.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.662493\{37.000000; 102.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.694567\{50.000000; 123.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.669884\{63.000000; 130.000000\}$$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 19 \rightarrow 20 \rightarrow 23 \rightarrow 24$

$0.702675\{50.000000; 125.000000\}$

$1 \rightarrow 4 \rightarrow 6 \rightarrow 12 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

The compromise solution pool for the fifth case of Table G.2 is shown below:

$0.711597\{36.000000; 117.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.711597\{36.000000; 117.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.711597\{36.000000; 117.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.708356\{32.000000; 110.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.687948\{36.000000; 110.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.687948\{36.000000; 110.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.697876\{42.000000; 122.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.697876\{42.000000; 122.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.708356\{32.000000; 110.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.697876\{42.000000; 122.000000\}$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

The compromise solution pool for the sixth case of Table G.2 is shown below:

$$0.701379\{32.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.701379\{32.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.701379\{32.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.630392\{33.000000; 86.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.630392\{33.000000; 86.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.630392\{33.000000; 86.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.672918\{34.000000; 104.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 18 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.672918\{34.000000; 104.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 18 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.676889\{36.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.592940\{40.000000; 88.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 11 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

The compromise solution pool for the seventh case of Table G.2 is shown below:

$$0.694402\{32.000000; 110.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.694402\{32.000000; 110.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.694402\{32.000000; 110.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.677317\{29.000000; 91.000000\}$$

$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.677317\{29.000000; 91.000000\}$$

$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.672297\{28.000000; 85.000000\}$$

$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.680019\{36.000000; 117.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.680019\{36.000000; 117.000000\}$$

$$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.665830\{36.000000; 110.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.672297\{28.000000; 85.000000\}$$

$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

The compromise solution pool for the eighth case of Table G.2 is shown below:

$$0.687424\{32.000000; 110.000000\}$$

$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$

$$0.687424\{32.000000; 110.000000\}$$

$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.686293\{28.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.686293\{28.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.686293\{28.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.664231\{36.000000; 117.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.661804\{31.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 18 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.652179\{34.000000; 96.000000\}$

$1 \rightarrow 2 \rightarrow 7 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.637314\{34.000000; 85.000000\}$

$1 \rightarrow 4 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.620822\{39.000000; 103.000000\}$

$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 15 \rightarrow 21 \rightarrow 22 \rightarrow 24$

The compromise solution pool for the nineth case of Table G.2 is shown below:

$0.700290\{28.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

$0.700290\{28.000000; 85.000000\}$

$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$

The compromise solution pool for the nineth case of Table G.2 is shown below:

$$0.700290\{28.000000; 85.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.700290\{28.000000; 85.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.700290\{28.000000; 85.000000\}$$
$$1 \rightarrow 2 \rightarrow 5 \rightarrow 10 \rightarrow 16 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.695160\{29.000000; 91.000000\}$$
$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.695160\{29.000000; 91.000000\}$$
$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.672739\{31.000000; 85.000000\}$$
$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 18 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.601848\{41.000000; 116.000000\}$$
$$1 \rightarrow 3 \rightarrow 7 \rightarrow 12 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.658425\{33.000000; 91.000000\}$$
$$1 \rightarrow 2 \rightarrow 8 \rightarrow 13 \rightarrow 14 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.680447\{32.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$
$$0.680447\{32.000000; 110.000000\}$$
$$1 \rightarrow 3 \rightarrow 8 \rightarrow 13 \rightarrow 17 \rightarrow 21 \rightarrow 22 \rightarrow 24$$