PARAMETER SELECTION FOR GENETIC ALGORITHM (GA)-BASED SIMULATION OPTIMIZATION

A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Onur Boyabatlı

August, 2001

I certify that I have read this thesis and in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. İhsan Sabuncuoğlu (Principal Advisor)

I certify that I have read this thesis and in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. M. Selim Aktürk

I certify that I have read this thesis and in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Doğan Serel

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray

Director of Institute of Engineering and Sciences

ABSTRACT

PARAMETER SELECTION FOR GENETIC ALGORITHM (GA)-BASED SIMULATION OPTIMIZATION

Onur Boyabatlı
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. İhsan Sabuncuoğlu
August, 2001

Improvements on heuristic techniques with the availability of faster PC's increase the importance of simulation-optimization (sim/opt) applications. Sim/opt methodologies use computer simulation integrated with an optimization sub-routine to optimize the problems of interest. The main contribution of these methods is to make simulation as a prescriptive tool rather than a descriptive tool, which has been widely used as the descriptive tool for estimating the performance of complex stochastic systems. Sim/opt methodologies have been applied on various combinatorial optimization problems, and the current trend in sim/opt area is the use of meta-heuristic techniques. Genetic Algorithm (GA) is the well known meta-heuristic, which is a global search algorithm taking its inspiration from natural genetics. GA has several parameters affecting its performance. Even for the GA with same structural parameters (coding scheme, operator types, stopping criterion), the different combinations of numerical parameters (initial population type, population size, maximum generation number and the crossover and mutation probabilities) may lead to drastic changes in the performance of the algorithm. This study examines the effects of numerical parameters of GA on its performance in terms of both fitness and CPU time; and proposes guidelines for appropriate parameter selection. A test problem of a serial assembly line taken from the literature is used for the GA-based simulation-optimization application. A genetic algorithm coded in C is integrated with a simulation model developed using SIMAN simulation language. Modifications on the test problem are made to analyze the behavior of GA parameters under different experimental conditions. The computational results reveal that in the case of a dominant set of decision variables, for rapid convergent GA applications high mutation rates are more useful, whereas the crossover operator does not have any significant impact on GA performance.

Keywords: Simulation, Optimization, Genetic Algorithm, parameter selection

ÖZET

GENETIK ALGORITMALARA BAĞLI BENZETİM-ENİYİLEME UYGULAMALARINDA PARAMETRE SEÇİMİ

Onur Boyabatlı
Endüstri Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Dr. İhsan Sabuncuoğlu
Ağustos, 2001

Daha hızlı kişisel bilgisayarların varlığı, bulgusal tekniklerdeki ilerlemelerle birlikte benzetim-eniyileme uygulamalarının önemini arttırmıştır. Benzetim-eniyileme metodları ilgili problemleri eniyilemek için, bilgisayar benzetimini bir eniyileme alt-rutini ile tümleyerek kullanırlar. Bu metodların ana katkısı, kompleks rassal sistemlerin performanslarını kestirirken tasfir edici bir araç olarak yaygınca kullanılan benzetimi, hükmedici bir araç haline getirmektir. Benzetim-eniyileme metodları birçok birleşi eniyileme problemleri üzerinde uygulanmıştır, ve benzetim-eniyileme içindeki şu anki trend meta-bulgusal tekniklerin kullanılmasıdır. Genetik algoritma en yaygın meta-bulgusal teknik olup, ilhamını doğal genetikten alan tümel arama algoritmasıdır. Genetik algoritmaların performansını etkileyen birçok parametresi vardır. Aynı yapısal parametredeki (kodlama yapısı, işlemci çeşitleri, sonlanma kriteri)genetik algoritmalarda bile, sayısal parametrelerin (populasyon büyüklüğü, en yüksek cenerasyon sayısı, değişim ve eşeyleme olasılıkları, başlangıç popülasyonu çeşidi) değişik birleşimi, algoritmanın performansında şiddetli değişikliklere meydan verebilir. Bu çalışma sayısal parametrelerin genetik algoritmanın kabiliyet ve ana işlemci zamanına dayanan performansı üzerindeki etkilerini inceler. Uygun parametre seçimleri için rehberlik önerir. Literatürden alınan bir seri montaj hattı, test problemi olarak genetik algoritmalara dayanan benzetim-eniyileme uygulamasında kullanılıyor. C dilinde kodlanan genetik algoritma, SIMAN kullanılarak geliştirilen benzetim modeline tümlenmiştir. Test problemi üzerinde değişiklikler yapılarak genetic algoritmanın parametrelerinin değişik deneysel koşullrdaki davranışları incelenmiştir. Hesaplanan sonuçlar ortaya çıkarmıştır ki; karar değişkenlerinin üstün bir seti olduğu durumda, hızlı yakınsak genetik algoritmalar için yüksek değişim olasılığı daha kullanışlıyken, eşeyleme işlemcisinin genetik algoritmanın performansı üzerinde belirleyici bir etkisi olmamaktadır.

Anahtar Kelimeler: Benzetim, Eniyileme, Genetik Algoritma, parametre seçimi

To my family and my little lovely virus

ACKNOWLEDGEMENT

I would like to express my deep gratitude to Assoc. Prof. İhsan Sabuncuoğlu for his guidance, attention, motivation, suggestions, and patience throughout this work.

I am grateful to Assoc. Prof Selim Aktürk and Assist. Prof. Doğan Serel for their valuable comments and kindness.

I would like to express my special thanks to Güneş Erdoğan for his deep understanding, and his sincere help throughout the work.

Special thanks to Deniz Orhon, for her deep motivation and encouragement, and the reminding when I forgot myself. It is precious to know there is someone who really believes in me.

And my family, many thanks to you for your moral support and great encouragement all the time.

I cannot fully express my dear gratitude for my other colleagues, and professors for their both academic and moral support and encouragement. Many thanks, especially for your patience and understanding.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

When the systems under consideration are complex and contain stochastic features, in case of almost all real-life situations, analytical models to represent the system behavior become impossible. In these situations, simulation is usually recommended as the tool for the analysis of the system. The goal of the simulation is to measure the performance of the system under some set of input parameters (alternatives or decision variables). Input parameters are plugged in the simulation model, and outputs (performance measures) are measured at the end of the simulation.

Optimization is finding the parameter (decision variables) combination, which gives the best performance (best objective function value) among all feasible combinations of parameters. There are many optimization algorithms developed in the literature, and most of them require mathematical expression of model and objective function, which becomes an arduous task for complex systems. The main advantage of simulation appears here, that is, it does not require mathematical expression of model and even objective function.

Unfortunately simulation itself is not an optimization technique. The best thing that can be done with only simulation is, analyst can simulate the system under different combinations of decision variables and select the one with the best performance. When the solution space is large, it is impossible to process all the points, thus the conclusions may be misleading. This leads the use of simulation incorporated with the optimization algorithms, which corresponds the main frame of all simulation-optimization techniques. The optimization algorithm proceeds by going new solutions from current solutions, and the performance of solutions are measured by simulation. The procedure iteratively continues until some stopping criterion is met, and best result obtained is set to be the optimal solution. The flowchart in Figure 1 gives the general overview of the simulation-optimization techniques.

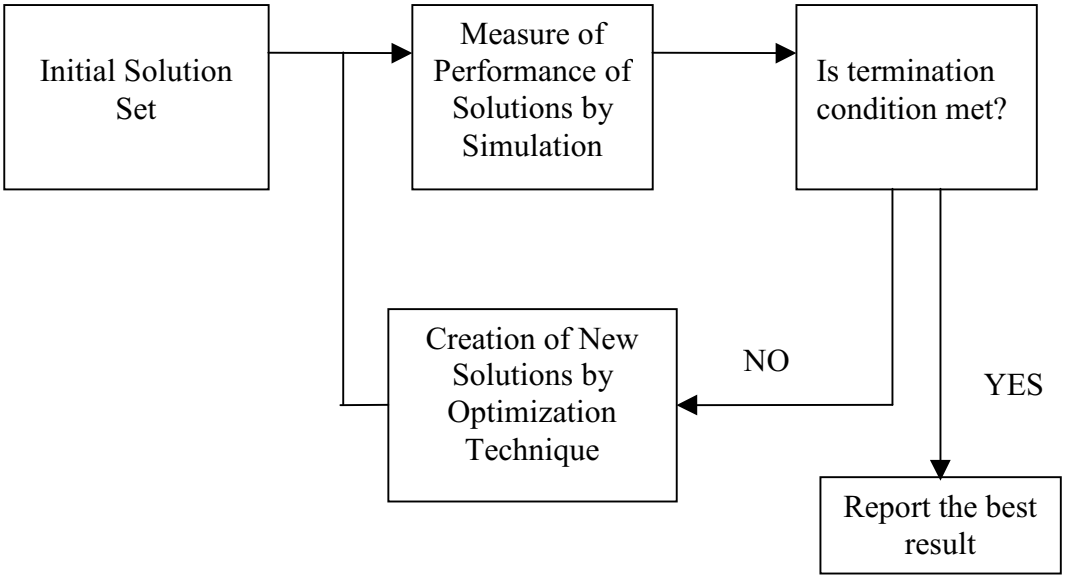


Figure 1: Flowchart of Simulation-Optimization Techniques

After the power of simulation-optimization techniques first observed, there have been various optimization methods that were integrated with the simulation. Earlier developments were generally local search algorithms, which benefits from the derivative or gradient information of the solution space. As the PC's become faster, and new heuristic techniques are developed, importance of simulation-optimization (sim/opt) is increased, and new methodologies are included in sim/opt literature. There became a shift from local search to global search algorithms, which generally do not require explicit mathematical representations.

The sim/opt literature has used the inferences gathered from other classes of science in the development of global search algorithms. Different processes in nature and other scientific branches have incorporated to the optimization literature by the excellent work of several authors. Meta-heuristics like simulated annealing and genetic algorithms are developed by this manner. There will be a detailed discussion in Chapter 2 about the simulation-optimization techniques.

Genetic Algorithm (GA) is a search algorithm based on the mechanics of natural selection and natural genetics. In natural genetics, genes in the chromosomes represent the physical features of individuals, and the presence/absence of these genes and their order determine the characteristics of individuals. Traits of individuals are passed through

generations by genetic operators, and the famous "survival of the fittest" principle is valid through generations, which means that the highly fit individuals, who can be accustomed to the environmental factors have more chance to deliver their genetic structure to future generations.

In GA, individuals represent the solutions to the problem under consideration. Starting from an initial population, GA proceeds population by population, each time creating new individuals from the current individuals by some genetic operators. Highly fit individuals have more chance to transfer their traits to future generations, like in the case of natural genetics. In theory every generation contains more fit individuals than the previous generation on the average, thus GA goes to better solutions as the algorithm proceeds. GA maintains iteratively and stops when the termination condition is met. The best solution found so far, is the solution to the optimization problem. A detailed explanation of GA is given in Chapter 3.

After Holland's work (Holland, 1975) and Goldberg's book (Goldberg, 1989), there have been a lot of implementations of GA in the optimization literature. General problems like traveling salesman problem, quadratic assignment problem and graph coloring, and more specific problems like assembly line balancing, layout optimization and job shop scheduling have been approached by GA. As the power of GA is observed, and applications to different problems increase, an extended literature about the theory and mathematical background of GA is also developed. These are given Chapter 2.

GA has several parameters that affect its performance. These parameters can be classified into two groups, structural and numerical parameters. *Structural parameters* are concerned with the structure of GA; that is, any alteration of these parameters requires re-coding of the algorithm, which also affects the performance of GA drastically. The coding scheme, operator types and stopping criterion are the main parameters of this group. For example, changing the stopping criterion needs further re-coding of GA. Consider the two different stopping criteria, maximum generation number and convergence of the generations. For maximum generation termination, GA proceeds until the maximum number of generation is achieved, thus it will be enough to count the number of generations processed in the algorithm. But for the convergence criteria, in which GA stops when there is no significant difference between the population's maximum and average, GA should control the average and maximum of each generation iteratively, which requires further coding in GA.

The second group, *numerical parameters*, is composed of population size, maximum generation number, initial population type and operator probabilities. Alterations of these parameters do not require, a significant re-coding of GA, only the numerical variables defined in the algorithm are altered. For instance in the proposed GA of this study, where the detailed explanation is given in Chapter 4, to change the population size parameter, it is enough to change the global variable's value defining the population size, from the current level to the desired level. Although change in levels of these parameters do not have a significant alteration on the coding of GA, even for the same structural parameter set, GA may show different performance under distinct set of numerical parameters. Detailed explanation about all these parameters is given in Chapter 4.

Different parameter settings usually lead to different results in GA applications, for that reason, a number of studies are conducted about determining the appropriate levels of parameters. These studies are discussed in Chapter 2 by reviewing the GA literature. There are several studies about the theoretical development of GA applications for different problem types, but these studies generally consider one parameter at a time. The drastic changes in GA performance according to structural parameters help in the construction of literature about selection of appropriate levels of them. For instance, the analyst can determine the adequate coding scheme for his GA depending on the problem under consideration.

Numerical parameters are analyzed thoroughly, but as mentioned above these studies generally include one or two parameters at a time. To the best of our knowledge, there is no study in the literature that examines the behavior of *all* numerical parameters and proposes guidelines for parameter selections.

This study investigates the effects of numerical parameters on GA performance, by using a test problem taken from the literature. The other set of parameters, the structural parameters of GA are set to convenient levels, and the analysis will be held only for the numerical parameters. As mentioned above, different settings of structural parameters also alter GA performance drastically, thus it is another difficult task to determine the appropriate settings of these parameters. In the selection process, we review the literature, and try to select the most common and the simplest and the most easy-to code structural parameters for proposed GA. The details will be covered in Chapter 4.

The main aspect of this study is to make general conclusions about numerical parameters of GA, in addition to finding the best combination of the parameters for obtaining

best performance *for the test problem considered*. In our opinion, this will be a valuable contribution to GA literature for setting criteria to determine the appropriate levels of these parameters and to present their interactions *regardless* of the problem of interest. The best numerical parameter settings will be presented for the test problem, and general conclusions will be drawn from the analysis, by also examining the extensions of the test problem. The results of this analysis may provide background to GA-based simulation-optimization applications in selecting the appropriate levels of GA parameters leading to best performance and understanding the interactions between the parameters. Also the terminology used in the discussion of computational results of the test problem will be useful for other researchers in stating rational explanations for performance of GA in their own applications under different parameter settings.

The rest of the thesis is organized as follows. Chapter 2 presents the related literature about simulation-optimization techniques and genetic algorithms. Chapter 3 gives a brief overview about genetic algorithms, thus the knowledgeable readers can skip this chapter. Chapter 4 illustrates the proposed study, and the GA-based sim/opt model used during the analysis with the necessary statistical tools. Chapter 5 interprets the experimental results on the original test problem. Chapter 6 presents the further analyses made on the problem domain, whereas finally Chapter 7 gives the concluding remarks; and suggestions for future research. Appendices provide the computer codes.

CHAPTER 2

LITERATURE REVIEW

This chapter is organized in two main sections. The first section presents the simulation-optimization (sim/opt) literature developed up to now, with the theoretical studies and applications to various combinatorial optimization problems. The second section is devoted to the genetic algorithm (GA) literature, where sim/opt applications of GA are presented here with the relevant theoretical studies together.

2.1. Simulation-Optimization Literature

After the realization of integrating the simulation with an optimization sub-routine, there are various methodologies developed in the literature for sim/opt purposes. The first applications start with the convolution of the present optimization algorithms with simulation, where simulation is used for the measurement of the performance of a solution. Many theoretical developments are achieved in sim/opt methodologies, some of which will be presented below.

Simulation-optimization techniques can be categorized into two branches, *local* optimization and *global* optimization techniques. The local optimization techniques mainly require some mathematical expression of the system or the solution space, like the derivative estimation, gradient information or construction of regression models. The well-known sim/opt procedure is the *random search*. The points are selected randomly from the whole solution space and the one with the best simulation result is set to be the optimal solution. Although it is easy to implement, since it slowly converges to optimum because of the lack of processing of previous information in each iteration, it may result with misleading conclusions; random search is not a reliable sim/opt procedure.

Nelder-Mead Simplex/Complex Search (Nelder and Mead, 1965) and Hooke-Jeeves Pattern Search (Hooke and Jeeves, 1961) are other local optimization methods. The former one requires construction of a simplex with number of vertices, and response of each vertex is calculated by simulation. A new point is added to the simplex by discarding the worst point of

current simplex and projection of that point to the centroid of the remaining points. The latter one, checks if any alteration in a variable results with improved performance while the other variables are held constant. The procedure is applied for each variable, and terminated when the incremental values do not improve the response.

Response Surface Methodology (RSM) is a popular sim/opt procedure requiring some statistical background of the implementer. As a local optimization motive, RSM merely depends on fitting regression equations in the solution space. Starting from an initial solution, once a regression model is fitted by experimental designs, a movement takes place on the regression line in the way of improved response. When there is no improvement occurring along the regression line, new regression model is fitted on the current point. When the first order design does not fit the sub-region or analyst is doubtful about the closeness of optimal point, high order regression models are fitted instead of simple linear models.

Perturbation Analysis (Ho *et al.*, 1979), Frequency Domain Analysis (Schruben and Cogliano, 1981), Stochastic Approximation (Robbins and Monro, 1951) are the other local optimization techniques used in sim/opt applications, whose main characteristic is the requirement of gradient information of the solution space. These methods depend on the derivative estimation of the solutions. The extended literature about the local optimum seeking sim/opt algorithms can be found in excellent review papers of Meketon (1987), Jacobsen and Schruben (1989), Safizadeh (1990), Fu (1994), Azadivar (1992) and Tekin and Sabuncuoglu (2000).

The methodologies discussed so far are classified in local optimization techniques, because they cannot guarantee to find the global optima of the optimization problem, which constitutes the major drawback of all the methods. Criticism is the fact that most of them requires mathematical expression of solution space or objective function, or some additional information about the problem like the gradient or derivative information. Also some of them require high level of statistical knowledge to be applied.

These pitfalls of the local optimization sim/opt procedures force people to develop new heuristics or methodologies. The optimization people get benefit from the other branches of science and nature itself in the development of new global search techniques. Genetic algorithms (GA) and simulated annealing (SA) are the main examples for algorithms developed in this manner.

GA is first developed by Holland (1975), and takes its inspiration from the natural selection and natural genetics. The second part of this chapter examines the GA literature, and Chapter 3 gives the major framework of GA, so the interpretation is left to these parts of this study.

SA is included in optimization literature by Kirkpatrick (1983). SA is a search technique drawing its inspiration form the chemistry, the process of physical annealing of solids. The process is the heating operation of a metal to a very high temperature and then cooling at a slow rate to a low temperature. Algorithm starts with an initial solution, and neighbor of solution is generated by a suitable mechanism. A solution is selected in the neighbor, if it is better in terms of objective value, then that solution is accepted as the current solution. If vice versa, again there is a probability of accepting that solution as the current solution. This probability depends on a parameter T, which is analogous to temperature in physical annealing. SA begins with high values of T and stays at the same temperature for some number of iterations, then the temperature is gradually decreased until final temperature is reached, thus at each T, the probability of accepting poor solutions changes.

SA has found a lot of implementation areas in combinatorial optimization, because the acceptance of poor solutions obscures the algorithm to stick on local optima. The sim/opt applications of SA to manufacturing systems and assembly line balancing problems are presented in below paragraphs.

Tabu search (TS) of Glover and Laguna (1997) is another search technique used in sim/opt literature. It can be defined as a constrained search procedure, because the algorithm is not allowed to search some subset of the solution space in each iteration, where this subset constitutes the tabu list of TS. Although TS is younger than the other algorithms, it has found a lot of implementation areas in combinatorial optimization and simulation-optimization literature.

GA, SA and TS can be defined as meta-heuristics. The power of these meta-heuristics in simulation-optimization studies takes the attention of commercial simulation software vendors. Historically, there are several simulation packages, used in real industrial applications, but none of them has the ability to optimize the system under consideration. The development of these algorithms, with the improvements in computer technology made the companies add “optimization” sub-routines to their simulation packages. Tabu search, genetic

algorithms and simulated annealing construct the main principles of the optimization sub-routines of the commercial simulation software (Law, 2000)

The theoretical development of simulation-optimization methodologies is presented, with brief discussions about the procedures and their major properties. The following paragraphs illustrate some simulation-optimization studies carried on different problems, which are summarized in Table 1.

Table 1: Summary of Simulation-Optimization Applications

Publication	System	Techniques
Bengu and Haddock (1986)	Inventory System	Exhaustive Search, Fibonacci Search, Pattern Search, Nelder-Mead Search
Azadivar and Lee (1988)	Robot Manufacturing Cell	Simplex/Complex Search
Haddock and Mittenthal (1992)	Automated Manufacturing System	Simulated Annealing
Shang and Tadikamalla (1993)	Automated Manufacturing System	Response Surface Methodology
Teleb and Azadivar (1994)	Flexible Manufacturing System	Complex Search
Suresh and Sahu (1994)	Assembly Line Balancing	Simulated Annealing
Weintraub *et al.* (1999)	Scheduling of Manufacturing System	Tabu Search

Bengu and Haddock (1986) apply simulation-optimization techniques to an inventory system to determine the optimal levels of reorder quantity and reorder level for a continuous review inventory model. Exhaustive search, Fibonacci search, Pattern search and Nelder-Meads search are used separately as optimization subroutines integrated with computer simulation.

Azadivar and Lee (1988) develop a heuristic simplex method as a sim/opt methodology. The heuristic is applied to a robot manufacturing cell with three serial workstations and several robot manipulators in charge of loading, unloading and transporting the products. A simplex consisting of number of vertices is constructed, and the response at each vertex is

simulated. The vertices are compared statistically, and the point whose lower confidence limit is lower than the upper confidence limits of the rest is deleted.

Haddock and Mittenthal (1992) use simulated annealing (SA) with integrated computer simulation to find the optimal levels of three distinct sets of decision variables for an automated manufacturing system. There are only 120 different combinations of decision variables, and SA is able to find the optimal solution.

Shang and Tadikamalla (1993) use response surface methodology (RSM) in the optimization of a computer integrated manufacturing system of an automated printed circuit board manufacturing plant via simulation. The fractional factorial designs are used to reduce both the number of experiments, and CPU time. RSM is successful in finding near-optimal solutions to the problem.

Teleb and Azadivar (1994) present a new approach for multi-criteria optimization of systems using computer simulation. The methodology suggests a new way of defining a compromise solution in terms of the maximum likelihood, which a solution will contain the optimal solution for all objectives. The methodology is applicable to the objective functions and constraints that are normally distributed. The complex search proposed in the study is applied to a flexible manufacturing system, to support the methodology with empirical work.

Suresh and Sahu (1994) propose a simulated annealing (SA) algorithm to solve an assembly line balancing problem. The study draws some conclusions about the parameters of SA depending on the computational results gathered. The main finding is that the lower the rate of cooling, better will be the quality of solution, but slower cooling schemes lead to exponential increase in the computational time.

Weintraub *et al.* (1999) apply tabu search incorporated with computer simulation on a scheduling problem of a large-scale manufacturing system. There are over 1000 jobs and 100 machines in the system, and the objective is the minimization of maximum lateness. The computational results reveal that the proposed algorithm manages to find optimal or near-optimal schedules for different industrial settings.

2.2. GA Literature

GA is first developed by Holland (1975) and his associates in University of Michigan. Holland's work is mainly about the theoretical development of GA, and his work presents the mathematical expressions lying behind the logic of the algorithm. Although GA seemed to be

a powerful global search algorithm, the complexity of the coding scheme and the CPU time it requires were the obstacles in front of GA applications. With the availability of faster PC's, Goldberg's book (1989) about GA opened a new period for GA literature.

Goldberg gives comprehensive information about GA in his book. Beside the mathematical underpinnings, the applicative features of GA with modified operators are presented in the book. The main important point of his study is the computer code of GA. Goldberg edited all the code required for an excellent GA algorithm, which is very beneficial in the implementation of GA for other optimization people. The modifications required in original GA, developed by Holland, for various combinatorial problems are also contained in the book.

Goldberg's study is not the only book published about GA in the literature. Various authors analyzes GA, and presents their findings with comprehensive discussion about GA and its parameters. The followings are some examples.

Michalewicz's (1992) book about GA presents the structural development of the algorithm. The working principles of GA, besides the effectiveness are presented in this study. There are also various extensions of GA operators to guide the analyst in different applications. There is a section about the comparison of binary and floating-point representations. Also coding structures of GA applied to different combinatorial problems like traveling salesman problem (TSP) and transportation problem are presented in the book.

Reeves (1993) collects meta-heuristic techniques like tabu search, simulated annealing and artificial neural networks in his book, where there is a chapter devoted to GA. Author presents detailed explanation about GA, and the working principles of GA by constructing theoretical work behind the algorithm. The operators and extensions of them from GA literature have found great attention in this book. Various modifications of GA operators and parameters are included, whereas author presents GA applications to combinatorial problems like, TSP, scheduling, graph coloring, bin packing, set covering and Knapsack problems finally.

Man *et al.* (1999) publish another book about GA, which mainly covers the signal processing features. The book includes introductory material about GA, with mathematical background and modified operators. Authors present GA applications in filtering, H-infinity control, computational intelligence, speech recognition systems, production planning and scheduling systems and communication systems, respectively.

After the power of GA is understood, and the comprehensive books ease the application of GA, people apply the algorithm on various optimization problems. Unfortunately GA has several parameters affecting its performance in finding near-optimal solutions, and inadequate settings of these parameters may lead to very poor performance of GA. Thus, theoretical studies are conducted for determining appropriate parameter levels for different applications. In fact, most theoretical studies aim to make general conclusions that are valid for all GA applications, but the problem-dependent characteristics of GA hinder these conclusions.

Another class of theoretical studies is conducted on the development of new GA operators. Holland's GA has had a lot of modifications since it was developed. The main reason behind these studies is the inapplicability of GA to some problems. Since the known operators are invalid for the structure of the problem under consideration, or give poor performance of GA, new kind of operators and different types of coding schemes are developed to harmonize the problem with GA and increase the performance. Table 2 summarizes the theoretical studies carried about GA, which are presented below paragraphs.

Syswerda (1989) proposes a new type of crossover "uniform crossover" to GA literature. The mathematical and theoretical underpinnings of this operator are presented whereas empirical comparisons of the operator with one-point and two-point crossovers are handled on some function optimization test problems. Uniform crossover leads to better performance in most of the empirical studies of the author.

Goldberg (1989) analyzes the selection of appropriate population sizes for serial and parallel GA implementations. The approximate rate of schema processing is the main response of the analysis, which is maximized. Exact and asymptotic formulas are also presented for the expected number of schemata in a population of strings. The computational results suggest that relatively small population size is adequate for serial implementations, whereas large populations are required for parallel implementations.

Schaffer *et al.* (1989) study the effects of some control parameters of GA on its performance. This is the closest study to ours. The control parameters selected are the population size, crossover and mutation probabilities and number of crossover points used in each mating. Experimental design is used for statistical analysis of the factors, with empirical studies on some function optimizations. The computational results reveal that the selection and mutation operators together is very significant in the behavior of GA. Mutation appears to be more effective than crossover on GA performance in this study.

Table 2: Summary of Theoretical Studies on Genetic Algorithms

Publication	Description	Additional Features
Syswerda (1989)	Development of new type of crossover "Uniform Crossover"	Comparison with one-point, two-point crossovers
Goldberg (1989)	Selection of appropriate population sizes	Distinct analyses for parallel and serial implementations
Schaffer *et al.* (1989)	Analysis of effects of some control parameters of GA on its performance	Population size, operator probabilities and number of cross sites are included
Richardson *et al.* (1989)	Guidelines for assigning penalty functions to infeasible solutions	
Fogarty (1989)	Analysis of effect of varying mutation probability on GA	Different levels of initial population types are considered
Back (1993)	Determination of optimal mutation rates	Separate analysis for uni-modal and multi-modal functions
Janikow and Michalewicz (1993)	Comparison of binary coding with floating point representation	Effect of dynamic mutation is measured for both cases
Starkweather *et al.* (1993)	Comparison of genetic sequencing operators	Empirical results are obtained on TSP and warehouse/shipping scheduler problem

Richardson *et al.* (1989) discuss some guidelines for GA with penalty functions. The concept of penalty functions is to penalize infeasible observations instead of discarding them, in order not to loose genetic information present in infeasible solutions. Generally for maximization problems, infeasible solutions are assigned to be low fitness values not to be selected frequently. Current thought is to penalize harshly, but the study presents some guidelines for the use of these penalty functions.

Fogarty (1989) discusses the effect of varying the mutation probability over time and its effect on GA performance. A minimization type test problem is used in the analysis. Two initial population types are used, one is a seeded population containing good solutions, whereas the other is randomly created. It is observed that varying the mutation rate significantly improves the performance of seeded population case, but not when the initial population is randomly generated.

Back (1993) studies optimal mutation rates in GA. The interaction of mutation rate with coding scheme of GA is also investigated. The results indicate that the simple binary coding scheme is appropriate for the test problems covered during the study. Optimal mutation rate is 1/l, where l is the length of string, for uni-modal function, and decreasing it towards the search process slightly accelerates the search. For multi-modal functions, 1/l is not appropriate for the analysis, but there is no mutation rate proposed, since different results are gathered from distinct test problems.

Janikow and Michalewicz (1993) compare the binary and floating point (FP) coding representations in GA, experimentally. The experimentations are conducted on a dynamic control problem, which is highly complex and quite difficult. For FP representation, new type of mutation operator is generated, also effect of dynamic mutation is measured for both cases. The CPU time performance beside the fitness (objective function value) performance is observed. The computational results suggest that; although there are advantages of low cardinality alphabets (binary), these advantages can be compensated for other type of coding scheme by designing new and appropriate operators, and this may lead to better performance of GA.

Starkweather *et al.* (1993) conduct a study on the comparison of several genetic sequencing operators, which are used in scheduling type of problems. The enhanced edge recombination operator, two types of order crossover, cycle crossover, partially matched crossover, and position-based crossover are the operators included in the analysis. To test the performance, a traveling salesman problem (TSP) and a warehouse/shipping scheduler problem are used. The experimental results reveal that the effectiveness of different operators depend on the problem domain; because operators, which perform poorly in TSP, work well in the second problem.

The theoretical studies generally enlighten the sim/opt people in the construction of appropriate GA for different problem types; and availability of faster PC's encourage them to apply this powerful global search algorithm integrated with computer simulation to various combinatorial optimization problems like facility layout optimization, assembly line balancing, quadratic assignment problem, job shop, manufacturing, queuing and inventory problems. There are also studies comparing GA with other optimization methodologies by empirical studies. The articles mentioned in the following paragraphs are summarized in Table 3.

Table 3: Summary of GA-based Simulation-Optimization Case Studies

Publication	System	Additional Features
Nakano (1993)	Job Shop System	3 famous job shop problems from literature are solved
Cruz and Haddock (1993)	Inventory and Queuing Models	Comparison with IPA (Infinitesimal Perturbation Analysis)
Yunker and Tew (1994)	University Computer System	Comparison with RSM and Hooke-Jeeves pattern search
Suresh *et al.* (1995)	Facility Layout Problem	Comparison with different GA application
Wellman and Gemmill (1995)	Asynchronous Automatic Assembly Line	Comparison with stochastic quasi-gradient methods
Pierreval and Tautou (1997)	Plastic Yogurt Manufacturer	Comparison with near-exhaustive search
Azadivar and Tompkins (1999)	Production System	Comparison with random sampling with same number of evaluations
Hamamoto *et al*. (1999)	Facility Layout of Pharmaceutical Plant	Comparison with human designers and layout optimization packages
Fontanili *et al*. (2000)	Free Modular Transfer Assembly Line	Comparison with analytical solutions
Lee *et al*. (2000)	Assembly Line	Relative fitness values are used

Nakano (1993) proposes a conventional GA for job shop problems. Job shop problem (JSP) is one of the most well known problems of combinatorial optimization, where there are N jobs to be processed on M machines. An operation sequence of a job is machine sequence, and operation sequence on a machine is job sequence. The full set of job sequences is called a representation, where a feasible representation is called a schedule. The objective is finding a schedule that minimizes the total elapsed time. A GA model for JSP is proposed, and 3 well-known JSP problems form optimization literature are solved with GA, and the experimental results show that solutions generated by GA is as good as those obtained by branch and bound methods.

Cruz and Haddock (1993) present GA-based simulation-optimization application on five different systems; three of which are queuing and the rest are inventory systems. The analysis of exponential propagation of certain schema over the generation of GA run is made. The results obtained are compared with the results gathered form IPA (infinitesimal perturbation analysis).

Yunker and Tew (1994) compare GA with pattern search of Hooke-Jeeves (1961) and response surface methodology via accuracy and stability by application on a test problem of university computer system. Accuracy is the measure of how close to the optimum, where stability is evaluated using the variance of the response function determined from 50 searches; lower the variance more stable the process. Maximum generation number of 7 is used in GA, where population size is 30. GA outperformed both methods for accuracy and stability, but the significantly large computation time of GA is noted to be a trade off.

Suresh *et al.* (1995) apply GA on a facility layout problem. The objective is the minimization of total workflow between departments, where 11 different extensions of the problem are analyzed, 2 of which have Euclidean distance, whereas the rest have rectilinear distance between stations. GA-based simulation model includes user-defined crossover and real value representation of solutions. Random selection is used as the reproduction operator ignoring the objective values of the solutions. 2000 generations are used as stopping criterion with population size of 100. The results obtained are compared with the results of Tate and Smith (1995), and except one case the proposed GA did better or same.

Wellman and Gemmill (1995) optimize an asynchronous automatic assembly line with GA. The system is a closed loop model with 10 stations in a single loop with unreliable stations. The buffer sizes between stations are the decision variables; the maximum throughput is used as the objective function. Infeasible strings are allowed in generations by assigning low throughput (penalty) to them. The results are compared with the results of stochastic quasi-gradient method (SQM), and GA cannot outperform SQM statistically.

Pierreval and Tautou (1997) use GA for the optimization of manufacturing system, which produces printed plastic yogurt containers. Decision variables are the silo capacity, which holds the initial inventory, warehouse capacity, which carries the finished goods inventory, and the process type applied during production. Total cost is minimized as the objective of the problem. Real chromosome representation is used, and a user-defined mutation is applied. When a mutation has to be performed on a numerical gene, either a

uniform selection is done in the interval (Min, Max) with probability 0.5, or one of the two interval bounds are selected randomly. Results are compared with near-exhaustive search, and same minimum is obtained in both applications.

Azadivar and Tompkins (1999) use GA with qualitative variables in the optimization of a production system. The system is a complex one, with several stages; a machine processed in each stage with different processing characteristic, and different routings for parts. The objective is the minimization of WIP (work-in-process) for a given production requirement, whereas the decision variables are the types of machines for each stage, routing of each part type, and layout plan for machines. During the GA implementation, simulation models are automatically generated through an object-oriented process and are evaluated for various candidate configurations of the system. Real chromosome representation is used with a population size of 70, and no mutation is allowed. Results of GA are compared with random sampling with same number of evaluations.

Hamamoto *et al.* (1999) apply GA-based simulation optimization to facility layout problem, presenting a case study on medium-size and small-size pharmaceutical plants separately. The multiple-objective analysis of maximizing the throughput rate and minimizing the traveling time per trip is conducted in the study. The results of GA are compared with the current real performance and the outputs of several layout optimization packages like Corelap, Craft and Blocplan. The comparison indicates that proposed GA outperforms human designers and other layout optimization algorithms in minimizing the traveling time per trip under the same throughput rate.

Fontanili *et al.* (2000) use GA in optimization of free modular transfer assembly line, where products move on the line according to the generalized flow shop, each product be realized in only one round without necessarily going on all the workstations. The routing of a product is constituted of “non-premature” and “non-redundant” phases. Simulation software incorporated with a Pascal code is used for the implementation. The inter-release times between 25 identical batches are determined to minimize the makespan. Real representation is used. Reproduction in each generation takes place from the best two members of the current population, 6 children are created. The results are compared with the analytical solutions and GA achieves near optimal solutions.

Lee *et al.* (2000) apply GA based simulation-optimization to an assembly line problem. The assembly line in the problem is a semi-automated serial compressor assembly line

comprising five workstations connected by accumulating belt conveyors, which also act as buffers. Decision variables are processing times of conveyors and stations, whereas there are three separate analysis done according to three distinct objective functions; maximizing throughput, minimizing tardiness and maximizing the utilization. Relative fitness values are used instead of the direct objective function values, with 25% of elitism. Different levels of crossover, mutation probabilities, and population size and elitism percent are investigated.

Considering the literature review here, it can be stated that GA has found applications on various combinatorial optimization problems. The power of search mechanism of GA makes the algorithm appropriate for optimization problems. As the applications have proceeded, theoretical studies are conducted in GA literature. These studies are mainly about the understanding of mathematical underpinnings of GA, development of new operators and structures for GA applications, and the determination of appropriate levels of the parameter settings of GA. The latter one is concerned with our study. This kind of studies generally contains analysis of only one parameter at a time, and ignores the interactions among the other parameters. To the best of our knowledge, there is no a study in the literature analyzing all the GA parameters, initial population type, population size, maximum generation number, and mutation and crossover probabilities. Our work will be a contribution to GA-based sim/opt literature by presenting the interactions between the numerical parameters, with the conclusions made for the best parameter settings of GA for the problem domain under consideration.

CHAPTER 3

GENETIC ALGORITHMS

3.1. Introduction

Genetic algorithms (GA) are search algorithms based on the mechanics of natural selection and natural genetics. In natural genetics, genes in the chromosomes act as a code for the physical features of each individual organism, and each organism is completely described by its gene values (its alleles). The presence/absence of genes and their order in the chromosome decide the characteristic features of individual species of a population. The different traits are passed on from one generation to the next through different biological processes, which operate on the genetic structure. By this process of genetic change and survival of the fittest, a population well adapted to the environment results.

Similarly, in a GA, a finite-length string coding is used to describe the necessary problem parameters of each solution for the search problem under consideration. Each string corresponds to an individual, and every individual presents its power in the survival process by terms of its *fitness* value. Higher the fitness values, better the individuals performance in the evolution process. There is a population of individuals with their strings and fitness values, which corresponds to a generation. GA is an iterative algorithm applied generation by generation. In every generation, first, parents are selected depending on their fitness values, and then by some genetic operators the strings of children are produced which contribute to the members of the new population. With their calculated fitness values, the new generation is obtained. And this procedure is repeated until some stopping criterion is met (generally depending on the number of generations). Like the natural genetics as the generations proceed, the fitness of the whole population (average fitness) increases, corresponding to better populations. The meaning is that, good properties of the individuals are carried to the further generations by GA, as in the evolution process. Figure 2 summarizes the logic of GA basically.

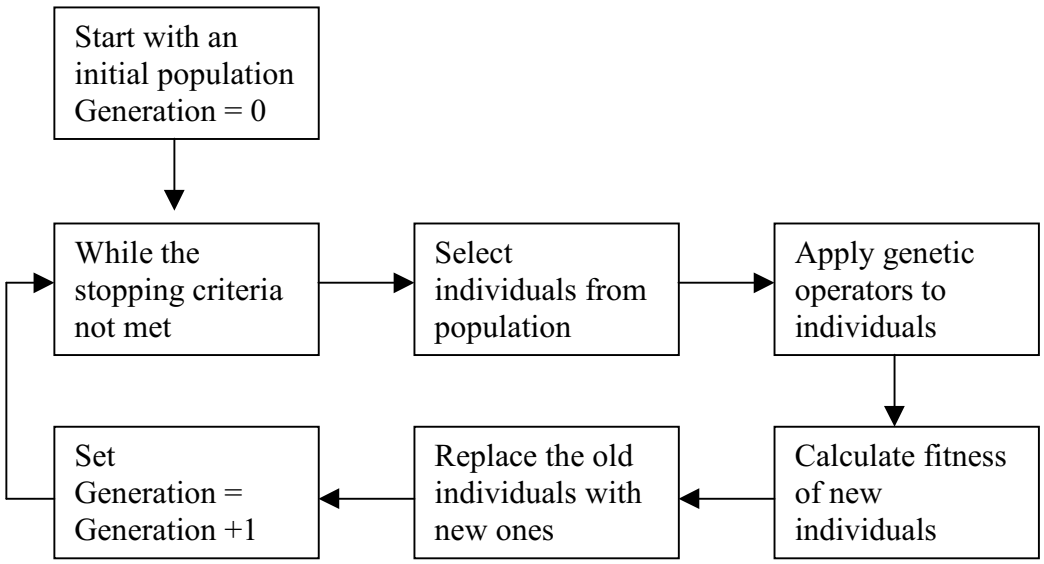


Figure 2: Flowchart of Simple Genetic Algorithm

3.2. Conceptual Framework

The GA starts with the coding of the parameters of the optimization problem to the strings. Although the real values of the parameters can be used in strings, generally the parameters are coded in strings over some user-defined alphabet (or alleles). Thus, GA generally works with the coding of the parameters instead of the parameters themselves. In most of the GA applications, binary coding of 1's and 0's are used, but also the other coding structures like Gray coding, sequence representation and real value coding is applicable. Since it is the most convenient and common one, binary coding is illustrated and used here.

The coding scheme must map the variable to a value between its maximum and minimum range. In a single parameter-coding structure, the string with all 0's should map to the minimum value of the variable, while the string with all 1's should correspond to the maximum value of the variable. The other values are mapped linearly in between them. To illustrate a typical coding, consider an optimization problem with one decision variable x, which ranges between 1 and 4. Let the parameter (designed length value) be coded in a two-bit integer. In this coding, the string "00" represents the real value 1, while "11" corresponds to 4. Similarly "01" represents 2, and "10" corresponds to 3.

A multi-parameter coding can be constructed by concatenating several parameters coding in a single string. Consider a case with two variables having the following binary representation:

Variable 1: 1111	String
Variable 2: 0000	11110000

Then the string is developed by the concatenation of each variable strings, resulting a eight-length string (11110000), where first four entries represent the first variable, while the last four belong to the second.

In GA's terminology, each string along with its decoded value is an *individual*, and the entity 1 or 0 in the string is an *allele*. A collection of such individuals in a user-defined solution space is called a *population*. A *generation* is the discrete time step required to complete creation of a new population, including the string processing and fitness calculation.

In nature, populations evolve by selective pressures, mating among the individuals, and some occasional events that alter the genetic structure, such as mutation. In GA's, similar effects are simulated by specific operators, and new population is created from the old population by recombination, duplication and the changing of strings of the individuals. In many GA applications there are three basic operators used:

- Reproduction
- Crossover
- Mutation

Reproduction (or selection) is normally the first operation employed on a population. It is the selection procedure by which highly productive individuals live and reproduce, and less productive ones die, where the productivity of an individual is defined as the individual's *fitness*. The selected individuals are the volunteers for the parents of the next generation, or they may be copied to the next generation directly. In each case, they have the ability to carry their good properties to the next generation.

Reproduction selects individuals based on their fitness values relative to that of the population. Although various selection schemes have been proposed and implemented, the most common selection procedure known is the *Roulette Wheel* selection.

In Roulette Wheel selection, the individual i can be selected with a probability $f_i / \Sigma f_i$ where f_i denotes the fitness value of i^{th} individual, and summation is taken over all the

individuals in the population. In this way, more highly fit strings have a higher probability of selection. Obviously, to apply this type of reproduction, the fitness values should be non-negative.

Fitness value is usually the objective function value of the optimization problems. However, there are some situations that the naïve choice of objective value becomes unusable or inadequate. A phenomenon that is often observed is, as the algorithm proceeds the population converges to set of similar solutions in terms of objective values, where it is very hard to discriminate the better solutions among the population. Another trivial situation is the presence of negative fitness values. To apply roulette wheel selection, negative fitness values should be avoided. In both cases *fitness scaling* is used. There are several types of scaling mechanisms in the literature like linear scaling, sigma truncation and power law scaling (Goldberg, 1989) to handle these problems. The main aspect of these scaling algorithms is to convert the naïve fitness value to a desired manner, which gathers the availability of discrimination and application of selection procedures.

Since selection is a stochastic process, there is no guarantee that the best individual (one with the maximum fitness value) will survive into the next generation. One way of dealing such a situation is *elitism*, in which the best one or more members of the population is directly copied to the next generation.

The second operation, crossover proceeds in two steps. First, two individuals are picked at random from the mating pool generated by the reproduction operator as the potential parents. Then the exchange of genetic material (alleles) occurs between the individuals with probability p_c. The uncrossed parents are directly copied to the next generation.

The exchange operation has various types depending on the type of the crossover used. In the literature, there are one-point, multiple-point, uniform, linear-order, partially-matched and cycle crossover types, which are applicable to various type of problems (Reeves, 1993). The simple one-point crossover is the most common operator used, so it will be presented here.

Next, a cross site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals. For example, consider the following binary parents with the cross site as shown:

Parent 1: 00|0000

Parent 2: 11|1111

Exchanging the bits to the left of the cross site, two new individuals are created following crossover:

Child 1: 001111

Child 2: 110000

These become members of the new population. This process continues until the population is filled with new individuals. Thus, the crossover operator creates new individuals by mixing the genetic information of the exiting individuals. Crossover plays a primary role with the reproduction operator in GA. After reproduction emphasizes the highly fit strings, crossover recombines these selected strings, to produce better individuals. They are the main operators, which avoid the inheritance of the good properties of individuals to the further generations.

Mutation is another operator used in GA. It is the occasional alteration of an allele's value with some specified probability p_m. Even tough there are other kinds of mutation seemed in the GA literature, the most common used is the bit mutation. In bit mutation, every bit of the string is mutated with probability p_m. For binary coded strings, mutation is defined as the conversion of 1's to 0, and vice versa. For example, consider the following string and mutation scheme:

String: 111111

Then after mutation of 2^{nd} and 4^{th} bits, the string becomes:

New String: 101011

Mutation is a very important operator of the GA, since it gathers the diversity to the population. Also it is a guard for premature convergence, that is, mutation helps GA not to stick to local optima by the diversification it provides. Without mutation GA may converge to a local optima, since every generation similar strings are processed. But mutation has the ability to directly change the parameter's value, contributing to a change in the searched space.

There are other operators like inversion and dominance developed in GA literature (Reeves, 1993), but the basic algorithm establishes from these three main operators. After all

the operators are applied to the current population, we have a set of individuals of the new population. With the calculated fitness values, these new individuals correspond to the next generation. GA proceeds iteratively until some termination condition is met. The most common used stopping criterion is the maximum generation number. After some predefined generation is reached, algorithm is terminated. Another application is that, the algorithm is terminated if there is no significant increase in the average fitness or the maximum fitness of the populations among last few generations. Also the convergence of the average fitness to the maximum fitness of the population with a precision level is another stopping condition used in the literature.

At the end of the algorithm, best result found so far is the solution of the optimization problem under consideration.

3.3. Mathematical Background

GA is a powerful search algorithm, which takes its basic principles from the natural evolution and genetics. Beside the algorithm itself, substantial amount of effort is spent on understanding the mathematical underpinnings behind GA's. A more rigorous understanding of the operators may be obtained by considering the processing of similarities among the strings. To describe the mathematical interpretation of GA, some terminologies should be defined. A *schema* (*schemata*, plural), as defined by Holland (1975), is a similarity template describing a subset of strings with similarities at certain string positions. Consider the following two strings:

String 1: 10111

String 2: 00110

These two strings have several similarity templates in common. If we define a * as a "wild card" to represent either a 1 or a 0, then the following are four schemata:

H_1: ***1*

H_2: *0*1*

H_3: 1***1

H_4: 00**0

H_1 and H_2 contain both strings, whereas only first string is contained in H_3 and the second string in H_4. In general, for alphabets of cardinality k and length l, there are $(k+1)^l$ schemata.

Two terms used to describe a schema H are its order $o(H)$ and its defining length $\delta(H)$. The order is defined as the number of defined positions (with 1's and 0's) in H, and the defining length is defined as the distance between the outermost defined positions in H. In the above example,

$$o(H_1) = 1 \; o(H_2) = 2 \; o(H_3) = 2 \; o(H_4) = 3$$
$$\delta(H_1) = 1 - 1 = 0 \; \delta(H_2) = 4 - 2 = 2$$
$$\delta(H_3) = 5 - 1 = 4 \; \delta(H_4) = 5 - 1 = 4$$

With these terminologies, Holland managed to derive a lower bound on the expected number of a schema H at generation t+1, which is computed from the expected number of schema of H in generation t (m(H,t)):

$$m(H,t+1) \geq m(H,t)\, f(H)/f_{avg} \, [\, 1- p_c \, \delta(H)/l\text{-}1 - p_m \, o(H)]$$

where l is the string length, f_{avg} is the average fitness of the population, p_c is the crossover probability, p_m is the mutation probability and f(H) is the fitness of the schema H, defined as the ratio of total fitness of all strings contained in schema H to expected number of schema H:

$$f(H) = \Sigma_{s_i \varepsilon H} \, f(s_i) \, / \, m(H,t)$$

According to this bound short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations. This is known as the fundamental theorem of GA or *schema theorem*. Highly fit schemata of small defining length and order have a crucial role in the mechanism of GA. Since as the length of the schemata decreases the possibility that crossover can disturb the schemata decreases. Also as the order decreases, the possibility of disturbance of mutation decreases. For that reason, these short, low order schemata are called for *building blocks*.

Another important feature of GA is so called *intrinsic parallelism*. Holland postulated that, although there are n structures in each generation processed in GA, at each generation n^3 number of schemata is processed, in other words the effective schemata processed is of the order $O(n^3)$. This is the basic principle where the power of GA comes from. Although the computational time to process the population is proportional to sample size n, there are actually n^3 schemata processed in parallel.

To sum up, GA seeks near optimal performance operating on building blocks, and the allocation of exponentially increasing number of trials to these building blocks in parallel is the essence of genetic search, where the power of GA comes from. As the generations progress, the trials become less random as the number of desirable sub-strings increases in the population. Because of the multiplicity of solutions, the search proceeds in parallel in the neighborhoods of the good solutions, which enhances the ability of GA to reach optimal or near-optimal solutions.

CHAPTER 4

PROPOSED STUDY

The study is about the analysis of numerical parameters affecting GA performance. The scope of the study is to determine best combination of parameters for GA based simulation/optimization application and draw some general conclusions about the parameters.

The objective of this thesis is given in Section 4.1. Section 4.2 defines the test problem and; Sections 4.3 and 4.4 summarize the simulation and genetic algorithm models proposed, respectively. The validation of the proposed models is presented in Section 4.6, and finally the statistical tools used during the analysis are explained in Section 4.7.

4.1 Study Objective

Genetic Algorithms (GA) are very powerful search mechanisms, working on the basic principles of natural selection and natural genetics. The power and simplicity of GA make it popular for even large-scale optimization problems. The main advantage of GA is that it does not require neither mathematical expression of response surfaces nor any derivative or gradient information. After Holland (1975) has proposed GA as a search mechanism, and especially following the Goldberg's (1989) book on GA, many applications of GA on various problem types are conducted. Beside the case studies, a lot of effort has spent on the theoretical development of GA up to now.

Although GA seems to be a robust algorithm, which contains same operators and has the same algorithmic logic for different applications, in fact the algorithm itself is significantly different for distinct problems. The main reason is that, GA has several parameters, and any combination of these parameters has unpredictable impacts on the performance on GA. To visualize the situation, we can classify the parameters into two groups, structural parameters and numerical parameters. These are explained in the following sections.

4.1.1. Structural Parameters

Structural parameters are the main factors affecting the GA performance, and the hardest set of parameters to be dealt with in a GA application. As understood from its

categorical name, they are concerned with the structure of GA. The change in any parameter value requires significant alterations in the coding pattern of GA's. The *coding scheme*, *operator types* and *stopping criterion* are the main parameters included in this group.

As stated before, GA starts with the *coding* of the problem to the strings, and decision maker should decide which coding pattern is appropriate for the problem under consideration.

The same aspect is relevant for the *operator selections*. Although there are main operators included in every GA application, since all of them have various types, it is a hard task to select the appropriate combination. The operators are the main tools presenting the power of GA on the optimization problem. To carry the good properties of the individuals to the further generations, reproduction and crossover operators should be selected conveniently. Also the choice of mutation type is effective on GA for not sticking on local optimums (or preventing the premature convergence). The other operators can be used in GA for the problem depending on the decision-maker's choice.

Another important structural parameter is *stopping criterion*. Different termination conditions generally result different performance of GA. As the evolution process requires a long period of time, GA has to be processed for a relevant duration, in order to present and apply its logic coming from the natural genetics. Terminating GA earlier disturbs the power of the algorithm, but the longer runs may have the inefficient use of CPU time. Thus, it is an arduous task to select the best termination condition.

However, since the publication of Goldberg (1989) and Davis's (1991) books on GA, a number of applications of GA have observed on various optimization problems. As a result, a huge literature has been developed on GA applications, presenting which combination of structural parameters is adequate for different types of problems. Thus, the performers can find the necessary insight required for the selection of structural parameters from the early experiences of others. To give an example, it is known that the sequence representation of coding schemes is better for scheduling and sequencing problems.

Moreover, the applicability of the structural parameters sets a constraint in front of decision-maker, and forces to eliminate some parameter values. For instance, the simple one-point crossover cannot be applied to the problems having sequence representation, because one-point crossover is exchange of the tails of strings after the cross point. This is irrelevant for sequence representations, since the tail of the second individual and head of the first

individual both may contain same variables in sequence. As one variable cannot be present in a single sequence, basic one-point crossover is inapplicable for this type of representations.

To sum up, although the structural parameters are critical for the effective use of GA, the theoretical studies conducted upon the structural parameters, guide the analyst in selecting the appropriate levels of structural parameters.

4.1.2. Numerical Parameters

The other parameters of GA can be classified as the *numerical parameters*. Even though they are in the second place in terms of the effectiveness on GA performance, even under same structural parameter values, different combination of numerical parameters alters GA performance drastically. One good thing about these parameters is that, it is easy to cope with them; they can be changed in the algorithm without any significant effort spent on coding. The *initial population type, population size*, *maximum generation number*, *crossover* and *mutation probabilities* are the main factors considered in this category.

The population size (n) of a genetic algorithm influences the rate of convergence and number of schemata that will be processed. Small populations run the risk of under-covering the solution space, while large population size is not cost-effective in terms of its large computation time. At each generation GA searches for n individual points, but closely n^3 schemata. If n is not large enough, GA cannot search the solution space adequately. On the contrary, large n will require a higher CPU time, which inversely influences the performance of GA in terms of cost. Another danger of small n is the high risk of premature convergence. If small populations are used, there may be some dominating individuals, which are always selected by the reproduction operator; thus, convergence to a local optima can occur, because search mechanism goes around the patterns of these individuals. If you have more individuals, the probability of having good individuals from different parts of solution space increases, so possibility of premature convergence will decline.

As stated above, stopping criteria is a very important structural parameter of GA. Generally some number of iterations is specified previously, and algorithm proceeds until this number is reached. This is the most common termination condition used in GA applications. Once the stopping criterion is set in this manner, the selection of the maximum generation number (m) becomes a competitive task. Like the population size (n), large value of m increases the CPU time drastically, while small values have the risk of improper GA

performance. As the evolution process requires a long period of time in nature, GA also needs some time (generations) to present his power coming from the natural genetics. When m is not large enough, GA may not be able to have the required time to show its ability. But in the opposite, large m gathers the availability of searching more individuals, which generally conducts to better performance in best fitness value found; unfortunately, CPU time elapsed increases severely. Generally after some generations CPU time shows an exponential behavior as the generation number increases.

The probabilities of the genetic operators are other influential numeric parameter on GA performance. As the crossover (mutation) probability, p_c (p_m), increases; obviously possibility to cross (mutate) increases. Low mutation and high crossover rates bring the risk of premature convergence, while high mutation and low crossover rates decrease the GA performance in terms of carrying the better solutions to future generations.

After selecting two good individuals as the potential parents, p_c determines whether to exchange the genetic material between the individuals or directly copy them to the next generation. Low values of p_c acts like DeJong's elitist strategy (1975), which guarantees the presence of good individuals in the next generation, but this time individuals do not have the opportunity to recombine their good patterns, which may lead to loss of a good combination. High values of p_c conduct to more exchange of genetic material (solution pattern in optimization literature), but may lead the absence of a good pattern in following generations, since crossover directly crashes the schemata.

Low values of p_m, brings the risk of premature convergence. Mutation operator is the guard for local optimums, since it gathers the diversity to the search space. If mutation occurs rarely, after some transient generations, similar individuals could dominate the populations, and same patterns are carried to the generations. In other words, search may occur around a local optima. High values of p_m damage the algorithm's pattern, since once a good schema is obtained, mutation can alter the bits of this schema, meaning the destruction of that schema. Thus, the probability of carrying the good patterns of good individuals will decrease as p_m increases.

Initial population type can be considered as the numerical parameter of GA, since it does not have any influence on structure of the algorithm. GA starts with an initial population and proceeds until a termination condition is met. Generally random populations are used as the starting points. But there are some empirical studies in the GA literature, presenting the

effectiveness of starting with a good population compared with a random one (Reeves, 1992). The initial population can be constructed from the results of some preliminary search heuristic like random search, pattern search etc. Seeding GA with good individuals speeds up the convergence to better solutions, but seeded initial population raises the risk of premature convergence. By seeding the algorithm, GA is forced to start the search around some good points (may be local optimums). If high domination of these points among the population occur, GA may converge to a local optima, that is premature convergence may occur.

From these explanations, it is clear that even under same structural parameter settings, the different combinations of numerical parameters may lead to drastic changes in GA's performance. Thus, finding the best numerical parameter setting for GA is a valuable result for GA literature. Throughout this study, the numerical parameters of GA are investigated, and the best combination of them is found for the test problems, and the effects of them under different problem settings is analyzed by the use of *experimental designs*.

The parameters under consideration are:

1. Initial Population Type
2. Population Size
3. Maximum Number Of Iterations
4. Crossover Probability
5. Mutation Probability

A test problem is taken for GA application, and significance of different values of these parameters on GA performance is analysed by the help of factorial designs and ANOVA (Analysis of Variance). The main aspect of this study is not to state the best combination of parameter values on this specific problem, but rather to make *general* conclusions about the effects of these parameters.

Two types of performance measure are considered throughout the analysis:

- Best fitness value obtained
- CPU Time elapsed

Generally GA performance is measured in terms of the best point it finds. But for us, this is a very naïve choice, because a good GA implementation should have the ability to find the maximum efficient point in a minimum time. So not only the objective value of the best point found, but also the CPU time elapsed should be considered while measuring the

performance of GA. Because it is trivial that large populations and more iterations increase the search space, so raise the possibility of finding better solutions, but CPU time generally shows exponential behavior with respect to generation number after some initial iterations, which is very cost-ineffective. The analysis has the objective of maximizing the fitness value and minimizing the CPU time together.

In the first part of the study, the effects of these parameters on an optimization problem are analyzed, then in the second phase the behavior they present under different objective function parameters is investigated. Finally the performance of the parameter settings is tested under different solution space of the same problem, by adding linear constraints to the optimization problem.

4.2. Test Problem

The problem is taken from the 2000 WSC Proceedings of Law and McComas (Law and McComas, 2000). The reasons for selecting this problem can be summarized as the following four points:

First of all, the study, where the problem is taken, is about the simulation/optimization applications. This study is conducted to compare optimization packages of different simulation software companies. Since GA is directly used in one of the sim/opt packages, we are certain about the applicability of GA to the problem.

Another reason is that, numerical results of the simulation model are available, hence we have the opportunity to validate our GA based simulation/optimization model against their results.

Also the problem represents a realistic manufacturing problem. The system is a basic assembly line with very realistic cost parameters, which can have a direct application in real-life.

Finally, the system is a basic one, and easy to handle in both simulation modeling and GA application. To some extent, we have the control over the system, and can discriminate the good solutions, which is very beneficial in testing the GA performance. Also by intuition we can predict the results when we make some modifications to the problem or the algorithm itself.

The system under consideration is a manufacturing system consisting of four workstations and three buffers, which are located between the stations starting from the first station (Figure 3).

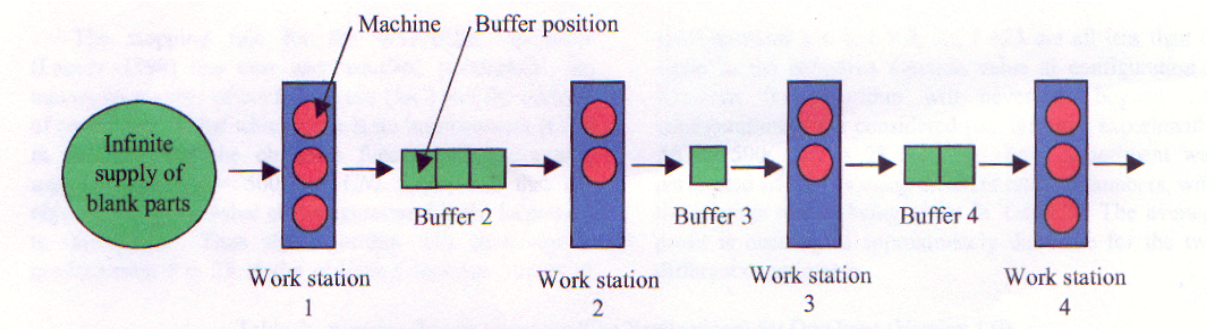


Figure 3: The serial assembly line modeled in test problem (Law *et al.*, 2000)

There is an infinite supply of parts before the first workstation, i.e. whenever it is idle; it pulls a new part immediately. A machine cannot discharge a part if the succeeding buffer is full. At the end of the fourth workstation, there is no buffer located, so that each part discharging from the workstation immediately leaves the system. The processing times have an exponential distribution with a mean given in Table 4:

Table 4: Processing Times of Machines in Workstations

Work Station	Mean Processing Time for a machine (hours)
1	0.33333
2	0.50000
3	0.20000
4	0.25000

Let,

M_i = Number of machines in work station i (i = 1,..,4) and

B_i = Number of buffer positions in buffer that is located at the end of station i (i = 1,2,3)

Every machine has a total operating cost of 25000 units, and every buffer position has a total operating cost of 1000 units. Besides, every throughput has a profit of 200 units. The objective of the problem is finding the optimal combination of number of machines and buffer positions that maximizes the profit of the system.

If we define the objective function random variable f, then the system in Figure 3 corresponds to f (3,2,2,3,3,1,2);

$M_1 = 3$, $M_2 = 2$, $M_3 = 2$, $M_4 = 3$, $B_1 = 3$, $B_2 = 1$, $B_3 = 2$ with an objective function value of:

$$f(3,2,2,3,3,1,2) = 200 \times \text{throughput} - 25000 \times (3+2+2+3) - 1000 \times (3+1+2)$$

where throughput is computed from simulating the model for a 30 day period. Also additional warm- up period of 10 days is used for eliminating the effect of the initial transient state.

Increasing the machine numbers and buffer locations increase the production rate, but also the operating cost, thus extra units of each variable are added to the system if the profit gathered from the increase in throughput can compensate the cost of extra units.

In the original problem, the number of machines in work station i ranges from 1 to 3, while buffer positions range from 1 to 10, so that there are $3^4 \times 10^3 = 81{,}000$ different combinations of decision variables. But we change the range of the decision variables as; machines' range from 1 to 4, and buffer positions' range from 1 to 16; thus there are $4^4 \times 16^3 = 1{,}048{,}576$ combinations of decision variables.

There are two reasons behind the modifications of the original problem's parameters. First of all, we want to measure the performance of GA on a more complex problem. The original problem has only 81,000 combinations, which can be stated as a small-scale optimization problem. But now we have over one million combinations that the modified problem can be classified as the large-scale problem.

Another reason is discarding the infeasible solutions generated during GA implementation. In the original decision variable settings, our GA model produces infeasible solutions, and these infeasible solutions have nearly half percentage in all solutions generated, which makes the GA inefficient. Moreover, the number of infeasible solutions generated by the algorithm differs among distinct parameter settings, thus comparing CPU time of the algorithm under these circumstances may lead us to false conclusions, since the algorithms spend different duration for dealing with the infeasible solutions. By the modification of the ranges of the variables, the feasibility of strings (points) generated during implementation is guaranteed.

4.3. Simulation Model

The simulation model is developed using SIMAN V simulation language in Unix environment. Model is a generic one, and there is no need for modification for the simulation of the system with different values of decision variables of the problem. This is provided by the use of data files. Simulation model starts with the reading of data files, and the values in these data files are set to the values of machine numbers in workstations and buffer locations between stations (decision variables) of that model. At the end of the simulation run, the total profit of the system is written to an output file, for GA to process from that file. 5 replications of each system are taken, and the output file contains the total profits (objective function value) for each of the 5 replications. Each simulation run lasts for 57600 seconds (40 days), in 14400 (10 days) of which no statistics are collected, because of the warm-up period. On the average for a loaded system, where high throughput rate is achieved, simulation takes 0.3 minutes, while a less loaded one with less production rate lasts for 0.18 minutes including all 5 replications. The sample code of the simulation model can be seen in Appendix A.

4.4. GA Model:

The main concern of this study is to analyze the effects of numerical parameters of GA on its performance in terms of best fitness value, and CPU time. To apply such an analysis requires selection of structural parameters of GA. GA is coded with respect to a specific set of structural variables, and experimentation is conducted at each combination of numerical values.

The basic GA model is selected for the analysis. The binary coding scheme, roulette wheel selection, raw objective value as fitness with offsetting scaling if necessary, one-point crossover, bit mutation and maximum iteration numbers as stopping criterion are used in developing the algorithm. Neither other operators nor combined strategies like elitism are applied. It is one of the first developed GA models in the literature (very similar to Holland's), and the most common kind of GA used in applications. Since it is basic, it is simple and easy to handle the coding of GA. Thus, GA with this set of structural parameters is selected for the analysis. Table 5 summarizes the structural parameters.

Table 5: Structural Parameters of GA Covered in the Analysis

Type	Definition
Coding Scheme	Binary Coding
Reproduction Operator	Roulette Wheel Selection
Fitness Calculation	Objective Function Value
Crossover Operator	One Point Crossover
Mutation Operator	Bit Mutation
Stopping Criterion	Maximum Number of Iterations

There are seven decision variables in the test problem, four belong to number of machines to each workstation and three are for the number of buffer positions between the stations. Machines range from 1 to 4, and buffers 1 to 16. In the binary coding representation, machine numbers have 2-bit scheme, where buffers have 4-bit scheme. For number of machines the binary coding and the corresponding real values are as follows:

$$00 \longrightarrow 1 \quad 01 \longrightarrow 2$$
$$10 \longrightarrow 3 \quad 11 \longrightarrow 4$$

For the buffer positions the coding scheme and the real representations are stated as below:

$$0000 \longrightarrow 1 \quad 0001 \longrightarrow 2 \quad 0010 \longrightarrow 3 \quad 0011 \longrightarrow 4$$
$$0100 \longrightarrow 5 \quad 0101 \longrightarrow 6 \quad 0110 \longrightarrow 7 \quad 0111 \longrightarrow 8$$
$$1000 \longrightarrow 9 \quad 1001 \longrightarrow 10 \quad 1010 \longrightarrow 11 \quad 1011 \longrightarrow 12$$
$$1100 \longrightarrow 13 \quad 1101 \longrightarrow 14 \quad 1110 \longrightarrow 15 \quad 1111 \longrightarrow 16$$

The strings representing the individuals are produced by the concatenation of individual strings, starting from the number of machines in first workstation, and continuing with the buffer positions in the same order. 4 two bit strings, and 3 four bit strings lead to a binary representation of individuals with string length 20. For instance a string corresponding to the decision variables' combination;

$$M_1 = 1 \quad M_2 = 2 \quad M_3 = 3 \quad M_4 = 4 \quad B_1 = 10 \quad B_2 = 11 \quad B_3 = 12$$

is:

00 01 10 11 1001 1010 1011

1 2 3 4 10 11 12

The fitness values of strings are the objective function values of that variable setting. The average of five replication results gathered from the simulation of the system with the real values of the variables in the string corresponds to the fitness value of the string. The simulation of the system with the above variable settings results with the total profit of 147,800, 148,800, 144,200, 156,200, and 135,400 for each replication, respectively. Then the fitness value of the string is calculated as the average of these numbers; 146,480.

The roulette wheel selection procedure is used for the reproduction process. The individuals are selected with a probability, which is equal to the ratio of individual's fitness to the total fitness of all individuals in the population. If the individual in the example is in a population with total fitness of 1000000 then the probability of selection of that individual is 0.14684. (146,480 / 1000000)

Since roulette wheel selection requires non-negative fitness values, scaling of the fitness is applied when necessary. The simulation duration is 30 days, which is reasonably long for producing the throughput that compensates the cost of machines and buffers; but for some rare combinations of machine and buffer variables the cost may be larger than the profit, thus negative objective values can be achieved. When there is a negative fitness value in the population, offsetting technique is used for fitness scaling. The minimum fitness value of the population is determined, and this minimum is carried to 0 by adding the absolute value of its own. Similarly the same amount is added to the other individuals in the population. For example, when there is a minimum fitness of –30,000 in the population, 30,000 is added to the fitness value of every individual in the population.

The one-point crossover is applied with a probability of p_c. After the reproduction mechanism drags two potential parents among the population, the crossover is applied with this probability. During crossover, a crossover site is selected randomly between 1 and (string length-1). Then the exchange of the tails occurs between the parents. Consider the following example, with parents and their real variable settings:

10111001|0100010101111 3-4-3-2-5-6-7

01001101|1000100110101 2-1-4-2-9-10-11

If the line represents the cross site, then the following two children are produced with the corresponding real values:

$$1011100100001001101 0 \quad 3\text{-}4\text{-}3\text{-}2\text{-}1\text{-}10\text{-}11$$

$$0100110111000101011 1 \quad 2\text{-}1\text{-}4\text{-}2\text{-}11\text{-}6\text{-}8$$

The mutation is applied by random changes in the bit values of strings. After the crossover operator, if no crossover has applied then the selected parents, if crossover has applied then the children, go under mutation. For every bit of the string mutation occurs with probability p_m. The following string is a solution to the test problem;

$$\underline{1}111111111\underline{1}1111111\underline{1} \quad 4\text{-}4\text{-}4\text{-}4\text{-}16\text{-}16\text{-}16$$

If the string is mutated form the underlined positions, the following string is obtained:

$$0111111110111111101 \quad 2\text{-}4\text{-}4\text{-}4\text{-}12\text{-}16\text{-}14$$

After the crossover and mutation are completed, the strings of the following generation's individuals have obtained. If there are infeasible solutions reached after the operations, then these strings are discarded, and new ones are generated from the beginning instead. The strings are generated twice at a time; if only one of them is infeasible, both are discarded and new ones are generated instead. The form the next generation, the fitness values of new constructed strings are required. These values are supplied from the simulation results of the strings. The real parameter values of each string is computed in the algorithm and written to output files. Then the simulation model is called in GA, and simulation model takes these files as input files, and calculate the objective function value of the parameter setting via simulation. The result is written to another file by the simulation model, and GA retrieves the fitness value of each string from that file. This algorithm proceeds for every individual in the population.

This procedure is started with an initial population and repeated until the maximum number of generations is reached. At the end of each GA run, the best fitness obtained up to now, and the CPU time elapsed are collected as the performance measures. Figure 4 resembles the flowchart of GA constructed in this analysis. A sample GA code can be found in Appendix B.

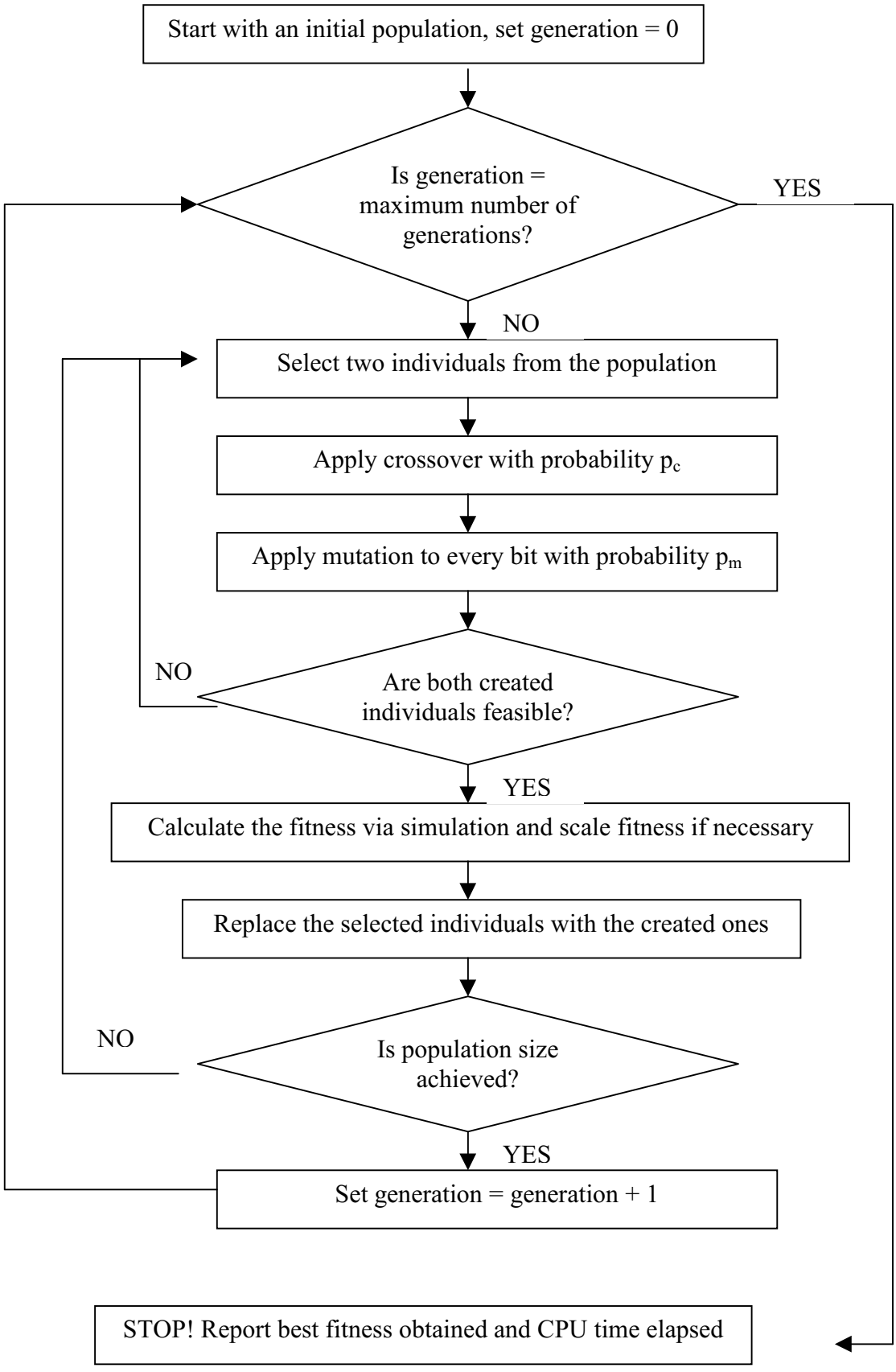


Figure 4: Flowchart of GA Model

4.5. Validation of Models

To validate the simulation model, Law's findings in the original problem are used (Law and McComas, 2000). In the original model the combination 3-3-2-2-7-8-4 has an estimated profit of 591,588 and 3-3-2-2-7-7-4 has an estimated profit of 591,512 monetary units, which are the average of 50 replications' results. To validate our proposed simulation model, we run the proposed model under same conditions and estimate the total profit of the system. Then 95% confidence interval is constructed for the expected profit to see whether the intervals contain the mean performance of original problem. Table 6 summarizes the results:

Table 6: Comparison of Simulation Results of Original Model and Proposed Model

Solution Combination	Law's Model	Proposed Model	Confidence Interval (%95)
3-3-2-2-7-8-4	591588	589872	(586562, 592582)
3-3-2-2-7-7-4	591512	588880	(585349, 592411)

For each point (solution combination), the Law's model's expected profit is included in our confidence intervals, thus we validate our proposed simulation model statistically. Unless the mean performance of the Law's model is included in the confidence limit, we cannot argue that our model is similar to the model developed in the original problem statistically.

It is known that for an appropriate genetic algorithm application, although there may be some erratic behavior of fitness values depending on the solution space, the average fitness of the populations should increase as the generations proceed. To validate our GA model, pilot run of proposed GA is taken with some settings of numerical parameters. Figure 5 presents the best fitness and average fitness behavior versus the generation number.

Even though there are sometimes some sharp declines in the average fitness of populations, generally as the generations proceed the average fitness presents an increasing behavior, and at large generations the average fitness becomes very close to best fitness value. Since the pilot run shows a predictable behavior, there is no reason for being doubtful about the incorrectness of our GA model.

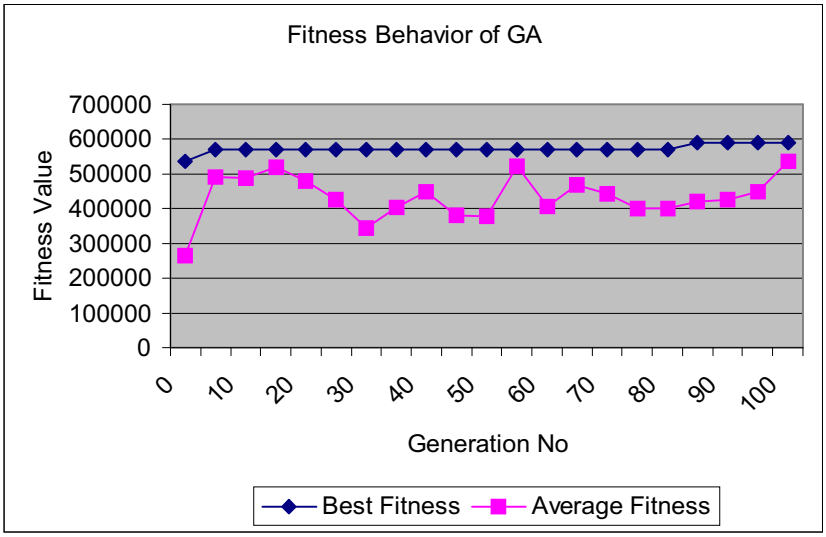


Figure 5: Behavior of Fitness According to Generation Number in Proposed GA

In addition to that, the proposed GA model is validated by investigating its behavior under some limiting conditions. One example is setting the mutation and crossover probabilities to 0. In that case all populations should contain the same individuals, so it is in our model. Other predictable parameter settings are also used, and the proposed GA model is validated.

The final and the most important validation procedure is measuring the performance of GA in terms of the best point it finds. From Law's original problem, the best solution to the problem is 3-3-2-2-7-8-4. Our algorithm proposes 3-3-2-2-5-8-3 combination as the best combination found, which is very close to the Law's solution of the test problem. Machine numbers are same for each solution; only slight differences are present in two separate buffer levels. The objective function values are also very close to each other.

In conclusion, we are convinced that our GA-based model is a valid model of the system and hence, we continue with the model for further investigation of the problem.

4.6. Design of Experiments

To illustrate the effects of different numerical parameters on GA performance in terms of best fitness values and CPU time, the main aspects of experimental design are used, since the principal goal of the experimental designs is to estimate how changes in input factors affect the responses of the experiment under consideration like in our case.

One of the main tools of experimental design studies is the factorial design. Factorial designs are widely used in experiments including several factors, where not only the main effects but also the joint effects of the factors are significantly effective on experiments' response. In factorial designs, different levels of each factor are specified and the analysis is conducted in every combination of these factors. At each factor combination, the response of the system is measured. At the end, by statistical techniques, the significant factors affecting the response, and the way they affect are determined.

The effect of a factor is defined to be the change in response produced by a change in the level of the factor. This is called the *main effect*, because it refers to the primary factors of interest in the experiment. It is also usual that, the difference in response between the levels of one factor is not the same at all levels of the other factors. This is explained by the interaction between the factors, and the *interaction effects* are very important for experimental designs.

The most common type of factorial designs is 2^k factorial designs. There are k factors each having only two levels, so that there are 2^k different combinations of the factors. Although there is no general rule about the selection of two levels of each factor, it is more convenient to set each level in opposite with each other. By definition we can define one level as the *low level*, and the other as the *high level*. The reason is that in the analysis of the effects, strong and significant conclusions can be made about the opposite values. But while selecting the two opposite levels (low, high), more concern should be given in order not to set unrealistic values for these levels.

Even though it is better to select opposite values in nature, another important point to be taken care of is the range of the two levels. Because in 2^k factorial designs, it is assumed that the response of the experiment shows linear behavior between high and low levels. If these levels are very far from each other, this linearity assumption will loose its strength. Under the light of these findings, 2^k factorial design is applied in this study.

4.6.1. 2^k Factorial Design:

The effect of numerical parameters of GA on performance measures is analyzed in this study. The objective is maximizing the performance of GA, i.e. maximizing the best fitness value found and minimizing the CPU time covered together. The numerical parameters are:

- Initial Population Type
- Population Size

- Maximum Generation Number
- Crossover Probability
- Mutation Probability

Since there are 5 parameters 2^5 factorial design is used in this study. Obviously there are 32 distinct combinations of parameters leading to 32 design points. For each point five independent GA runs are taken. This is provided by assigning different random number streams to the simulation model by changing the seeds of SIMAN V.

To apply the 2^k factorial design to our model, two levels of each parameter should be determined by considering the pinpoints given throughout the above paragraphs. The main source that we can trust in the correct parameter selection is GA literature, because there are a lot of theoretical studies carried out about the optimal settings of these parameters individually. Also some of these work present ranges for parameters for a good GA application. The following paragraphs state the levels of the parameters and the reasons behind them.

Initial Population:

Some reports have found that better initial populations may result in better GA performance. Generally the initial population is generated randomly, but some studies have reported that seeding a population with high-quality solutions, which can be obtained from another heuristic technique like random search, may help GA find better solutions more quickly than a random start. But seeding the initial population increases the risk of premature convergence.

For this purpose, we applied a random search of 250 points. The best 20 or 40 points depending on the population size are selected for the seeded population. Random population is generated by random selection among these 250 points. We take the random population as the low level of the initial population parameter, while the population gathered after random search heuristic (seeded population) as the high level.

Population Size:

Population size is one of the most influential factors on the performance of GA. Small populations run the risk of under-covering the solution space, while large population size is not cost-effective in terms of its large computation time. Goldberg (1989) indicated that the

optimal size for binary-coded strings grows exponentially with the length of the string n. There are also empirical studies suggesting that population sizes larger than 30 could be adequate in many cases. Alander (1992) reports that a value between n and 2n is optimal for the problem type considered. Considering these results, the levels of population size are taken as 20 (n) and 40 (2n), where 20 equals to the string length of our GA model.

Crossover and Mutation Probabilities:

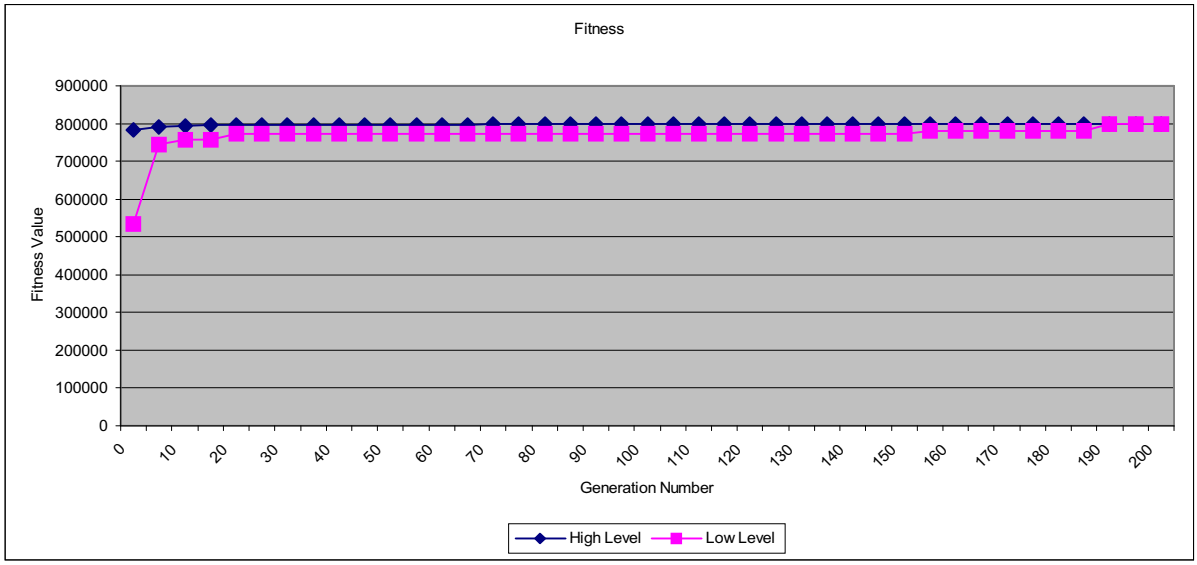
The selection of the crossover and mutation probabilities may change the performance of GA drastically. Low mutation and high crossover rates bring the risk of premature convergence, while high mutation and low crossover rates decrease the GA performance in terms of carrying the better solutions to future generations. Although it is very problem-dependent, generally mutation is applied with a low probability (typically less than 0.01), while crossover is applied with high probability (often 1). De Jong (1975) suggests that the bit-mutation rate should be n^{-1} where n is the string length. Michalewicz (1992) reports that crossover rates between 0.65 and 1, and mutation rates between 0.001 and 0,01 are useful in GA applications. In our study the crossover probability levels are set as 0.5 and 1,and the mutation levels are 0.01 and 0.05, where 0.05 is equal to 1/n, n is 20 in our model.

Maximum Generation Number:

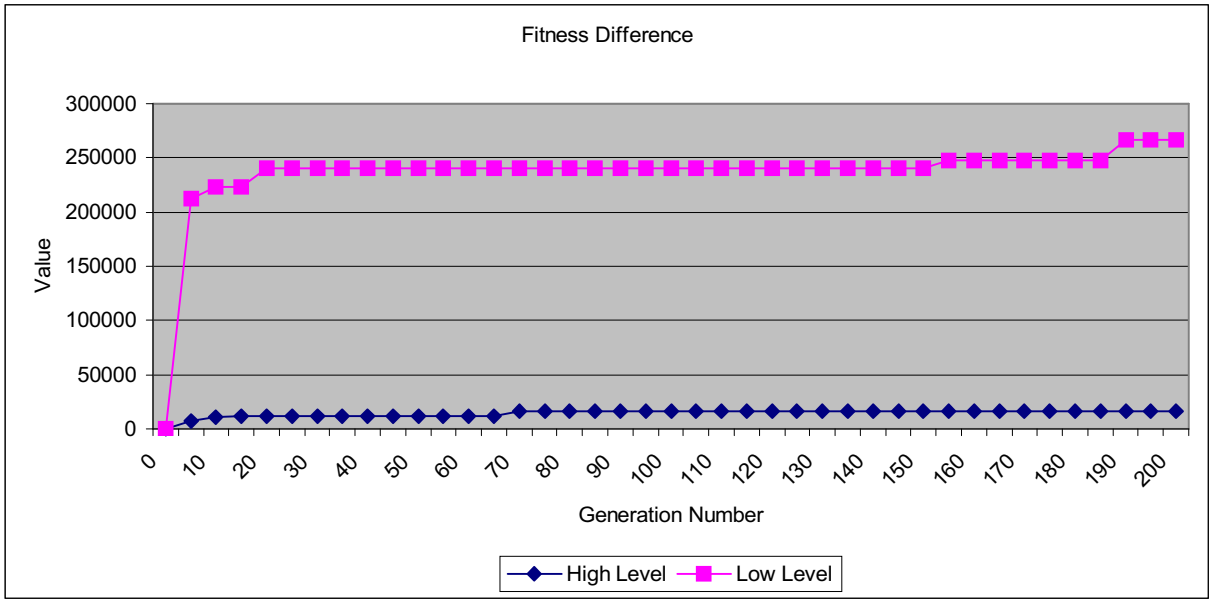
The maximum generation number is usually a very influential factor in GA performance. Its high value increases the computation time, while low level has the risk of poor performance of GA, since it narrows the searched space of the real solution space of the system considered. Although it is a very important factor, there is not a literature developed on it that can guide us in the selection of its' levels. The most important reason is that, the parameter itself is very problem-specific, and also it depends on the implementer's choice. To determine the high and low levels of the parameter, we make experimentation, thus we take some pilot runs by fixing the other parameters and change the maximum generation number.

We fix the other parameters at their low levels and high levels, and take the runs under these two separate conditions. The parameters at their high levels contribute to design 1, while the low level one contributes to design 2. Figure 6.a shows the best fitness value obtained under different values of maximum generation number of GA.

Since the points are not visible to scale, the fitness increments to maximum generation number are re-graphed (Figure 6.b). From Figure 6.b, it is visible that no increment in fitness is obtained after 75th generation at design 1, while little improvement occurs after 15th generation. For the second design point, little increment occurs after 15th generation and 190th generation. Beside the fitness value, we have to consider the CPU time as the performance measure. Figure 6.c represents the CPU time versus different levels of maximum generation number parameter.

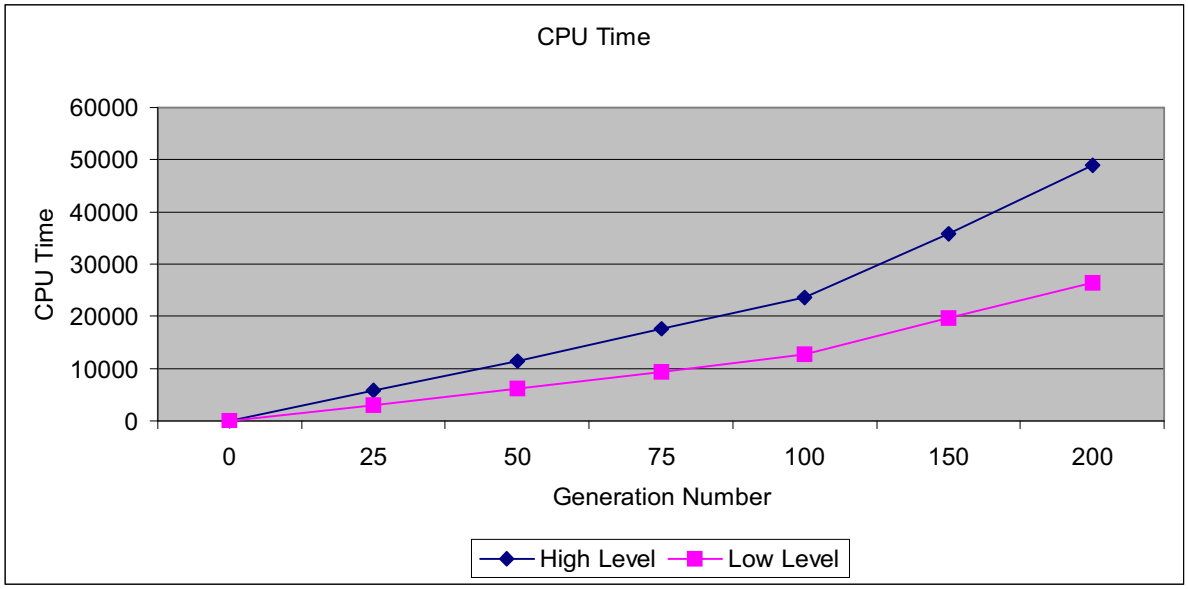


a) Best fitness obtained versus maximum generation number



b) Difference of Fitness versus Maximum Generation Number

Figure 6: Analysis of Maximum Generation Number



c) CPU Time versus Maximum Generation Number Graph

Figure 6: Analysis of Maximum Generation Number (cont'd)

The CPU time versus maximum generation number shows nearly exponential behavior, and at both designs CPU time increases drastically after 100th generation. Under these circumstances, as both the maximum fitness and CPU time considered, the levels of 50 and 100 seemed to be logical as the levels of the parameter.

In conclusion, as a result of the literature review and pilot experimentations, the two levels of each factor are determined as given in Table 7.

Table 7: Factors and Levels for Factorial Design

Parameter	Low Level	High Level
Initial Population	Random	Seeded
Population Size	20	40
Maximum Generation	50	100
Crossover Probability	0,5	1
Mutation Probability	0,01	0,05

The analysis is continued by taking 5 independent GA replications for each 32 design points. Then ANOVA (analysis of variance) is used for determining the significance of each effect, and the appropriate levels of each parameter for better GA performance in terms of

fitness and CPU time. We benefit from the SPSS statistical software package during the ANOVA.

4.6.2. Diagnostic Checking

The 2^5 factorial design is applied under some assumptions. The effects are assumed to be fixed, designs are completely randomized so that replications are independent of each other, and usual normality assumptions are considered to be satisfied. Before implementing the factorial design, these assumptions have to be checked.

The parameter levels are not selected randomly from a set of possible values, thus it is not a random-effect model. The levels are specified by the literature search and further experimentation, so first assumption is satisfied automatically.

The independence of replications, and the design points is provided by the use of independent distinct seeds of SIMAN V in simulation model. Since independent seeds are used, the designs are completely randomized.

Normality assumption requires more effort for adequacy checking. Although moderate departures from the normality assumption are of little concern in the fixed effects ANOVA, that is robust to normality assumption, since F test is only slightly affected (Montgomery, 1991); this assumption still has to be checked. The residual examination is beneficial in the diagnosis of normality violations. Residual plot of the model is constructed by the use of regression model, which is plotted regarding to the effect sizes estimated during factorial analysis. If residual plots of both CPU time and fitness responses show a linear behavior especially in the middle portion of the points, then there is no reason to suspect about the non-normality of the data.

After significant factors are obtained by using ANOVA, the normal probability of effects is used for the diagnostic checking of the adequacy of significant parameters. The plot of effect sizes versus corresponding normal probabilities has to show linear behavior for all insignificant effects, and significant effects should present an outlier, disturbing the linearity.

Finally, ANOVA assumes the homogeneity of variances between independent replications, thus homogeneity assumption has to be satisfied. There are a lot of ways to search the relevance of this assumption. In this study, Bartlett test (Montgomery, 1991) is used.

CHAPTER 5

EXPERIMENTAL RESULTS

The aim of this study is to examine the numerical factors, which affect GA performance. The factors investigated are the initial population type, population size, maximum generation number, and mutation and crossover probabilities. A test problem is used for the analysis, and 2^5 factorial design is implemented to determine the significance of each factors, and factor interactions. Also modifications of the problem are used for examining the behavior of the factors under different conditions. Table 7 summarizes the factors and factor levels of the analysis.

Table 7: Factors and Levels for Factorial Design

Parameter	Low Level	High Level
Initial Population	Random	Seeded
Population Size	20	40
Maximum Generation	50	100
Crossover Probability	0,5	1
Mutation Probability	0,01	0,05

All the computer applications are carried on Sun HPC Server 4500 with 12 400mhz CPU, 3 GB memory, 45 GB disk and 20/40 GB 8mm tape unit.

This chapter presents the experimental results for the original test problem. Model adequacy checking, the analysis results with respect to fitness response, and CPU time response are illustrated respectively. Finally the consequences of each separate analysis are combined.

Throughout this chapter during the factorial design of experiments, abbreviations are used instead of the factors' main titles. The following terminology is printed when necessary:

A: Initial Population Type

B: Population Size

C: Maximum Generation No

D: Crossover Probability

E: Mutation Probability

Also in factorial design tables "-1" denotes the low level of the factor, while "1" represents the high level. For example a design point of (1, -1, 1, -1, 1) corresponds to the GA parameters of seeded initial population, population size 20, 100 maximum generations, 0.5 probability of crossover, and 0.05 probability of mutation.

5.1. Original Test Problem

The test problem is taken from WSC 2000 Proceedings (Law and McComas, 2000). The system is a serial assembly line with four workstations, and three buffers located between the stations, starting from the first station. There are seven decision variables, four of which are the machine numbers in each workstation, whereas the rest are number of buffer locations. Objective function is maximizing the total profit, which depends on the throughput level, and machine and buffer operating costs.

5.1.1. Model Adequacy Checking

The GA model is processed 5 times at each of 32 design points. The results of 5 independent replications of the GA model are given in Table 8.a and Table 8.b for the fitness value and CPU time, respectively.

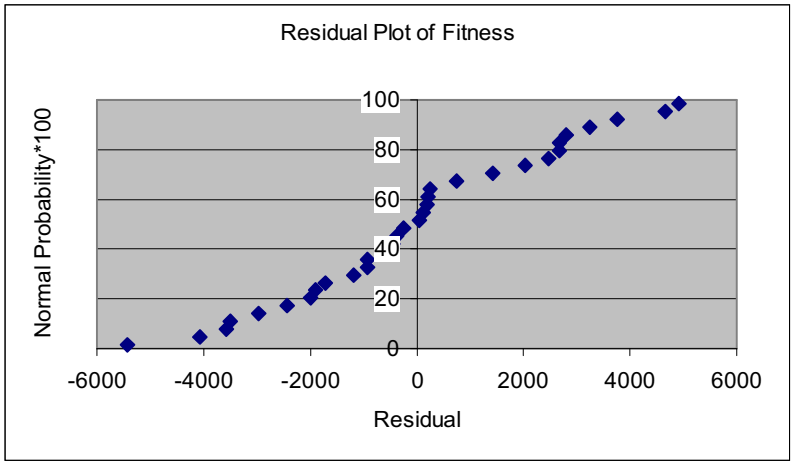
After having the GA results for the analysis, the assumptions of ANOVA are checked. The independence of the replications is provided by the use of independent seeds. The normality assumption is controlled by the residual plots of both responses separately. The residuals are calculated from the difference between the observed values and predicted values, where predicted values are estimated from the regression model developed from the data. Figure 7.a and 7.b presents the residual plot of fitness and CPU time, respectively.

Table 8.a) GA Results for Fitness Response

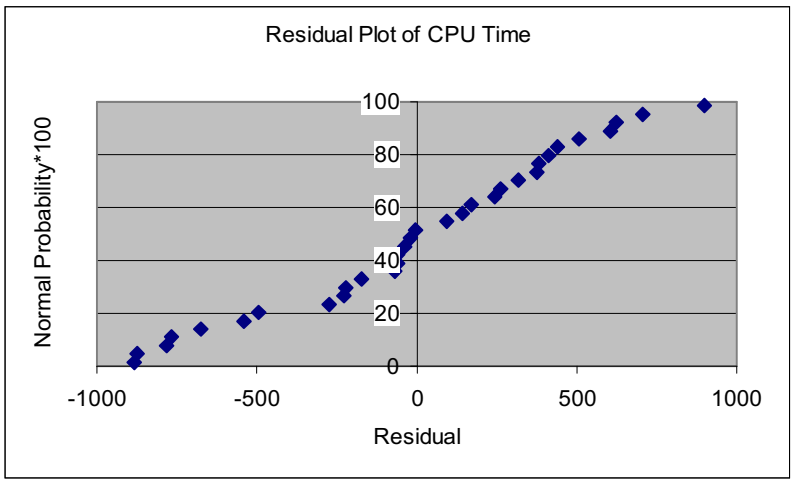
A	B	C	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	1	1	805160	800840	810080	807080	802520
-1	1	1	1	1	810160	805080	812760	798640	808080
1	-1	1	1	1	804960	801800	805040	796240	800400
-1	-1	1	1	1	805880	801320	802760	792600	784840
1	1	-1	1	1	799680	801800	803720	800960	799120
-1	1	-1	1	1	803440	799920	803360	797440	797040
1	-1	-1	1	1	788760	795680	809840	800480	787160
-1	-1	-1	1	1	781240	787720	796680	791400	791280
1	1	1	-1	1	802160	801320	805240	802880	798320
-1	1	1	-1	1	805000	800760	806320	804880	802520
1	-1	1	-1	1	805040	801880	799960	792880	786720
-1	-1	1	-1	1	801840	800680	804480	802320	793000
1	1	-1	-1	1	810240	803120	809320	806600	794680
-1	1	-1	-1	1	804960	799840	798000	805160	790800
1	-1	-1	-1	1	802000	793840	802640	792920	787040
-1	-1	-1	-1	1	800040	792920	795040	786120	782200
1	1	1	1	-1	796880	802080	797320	808400	803880
-1	1	1	1	-1	797240	809400	804400	791560	797200
1	-1	1	1	-1	794480	805000	790120	800760	793880
-1	-1	1	1	-1	780160	793240	778120	783120	790560
1	1	-1	1	-1	808760	792880	802280	797960	799600
-1	1	-1	1	-1	793680	792520	808160	794040	785320
1	-1	-1	1	-1	792520	794880	802000	786520	786720
-1	-1	-1	1	-1	769160	773920	778840	794120	785840
1	1	1	-1	-1	806480	799760	803480	802680	798320
-1	1	1	-1	-1	798480	784520	806440	802040	793920
1	-1	1	-1	-1	799680	801640	793080	799480	792200
-1	-1	1	-1	-1	785560	794920	780920	793200	779560
1	1	-1	-1	-1	805880	793080	800560	806600	795200
-1	1	-1	-1	-1	778280	785320	777280	791960	794960
1	-1	-1	-1	-1	802040	795480	806600	798640	785760
-1	-1	-1	-1	-1	789880	787880	789040	783080	782200

Table 8.b) GA Results for CPU Time Response

A	B	C	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	1	1	23571,85	23930,33	23609,02	24471,98	29239,03
-1	1	1	1	1	23746,6	23367,13	23613,39	23734,84	29224,82
1	-1	1	1	1	12132,16	11839,36	11958,2	11866,09	14365,33
-1	-1	1	1	1	11210,63	11962,43	11711,09	11638,96	14723,69
1	1	-1	1	1	11795,13	12001,66	11813,39	11487,05	14424,27
-1	1	-1	1	1	11662,21	11828,58	11884,52	11640,66	13939,17
1	-1	-1	1	1	5943,06	5957,47	6019,61	6186,27	7447,59
-1	-1	-1	1	1	5776,7	5900,05	5784,55	6034,81	6403,7
1	1	1	-1	1	23993,68	24100,61	24185,34	24298,4	29568,31
-1	1	1	-1	1	23326,59	23944,99	23442,24	23639,27	28751,36
1	-1	1	-1	1	12219,79	11495,9	12110,21	11954,38	14510,3
-1	-1	1	-1	1	11231,22	11550,88	11955,79	11951,51	14455,35
1	1	-1	-1	1	11663,92	12139,83	12176,73	11842,86	14454,5
-1	1	-1	-1	1	11771,46	11465,31	11595,13	11528,75	14680,65
1	-1	-1	-1	1	5971,68	6091,54	6081,27	5593,62	7316,23
-1	-1	-1	-1	1	5472,82	5417,85	5886,28	5970,7	6584,37
1	1	1	1	-1	26756,9	26121,11	26938,52	27263,42	31178,36
-1	1	1	1	-1	25897,58	26752,44	26862,2	26429,21	31991
1	-1	1	1	-1	13712,24	13577,94	14189,74	13696,03	16538,9
-1	-1	1	1	-1	13387,95	12696,09	13221,52	13112,78	16879,29
1	1	-1	1	-1	13319,93	13662,64	13887,64	13632,37	16675,64
-1	1	-1	1	-1	13020,46	12142,05	12552,56	12176,01	15465,34
1	-1	-1	1	-1	6966,53	6653,03	7093,68	6694,74	7955,79
-1	-1	-1	1	-1	5570,98	5942,28	6238,06	6271,27	7835,56
1	1	1	-1	-1	26279,09	26403,68	27164,6	27528,76	32842,94
-1	1	1	-1	-1	26773,8	27384,94	25011,61	27416,63	32254,6
1	-1	1	-1	-1	13425,7	12795,56	13845,61	13334,28	15948,22
-1	-1	1	-1	-1	13194,73	12591,47	12765,38	12756,62	15860,45
1	1	-1	-1	-1	13334,17	13690,92	13506,55	13711,33	16614,54
-1	1	-1	-1	-1	13281,85	12969,23	12176,99	13093,08	16596,13
1	-1	-1	-1	-1	6762,8	6439,06	6858,3	6645,11	8423,61
-1	-1	-1	-1	-1	6422,12	6307,06	6322,59	6021,14	7784,33



a) Residual Plot of Fitness Response



b) Residual Plot of CPU Time Response

Figure 7: Residual Plot versus Responses

Both graphs show linear behaviors, and there are not any significant deviations from a straight line, so both data sets satisfy the normality assumption.

To investigate the homogeneity of variance, Bartlett's test (Montgomery, 1991) is applied on both data sets. Two data sets pass from the test, and since each set satisfies the normality and independence assumptions, and variances among the replicates are homogeneous, ANOVA can be applied comfortably.

5.1.2. Fitness Response

ANOVA table is obtained by using the statistical software package SPSS. The last column indicates the significance of effects, and the previous column includes the F statistics used for the significance test. Table 9 is the ANOVA table for fitness response.

Table 9: ANOVA Results for Fitness Response

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	6476247704	31	208911216	5,40	0,000
Intercept	101576446552361	1	101576446552361	2627600,02	0,000
POPTYPE	1227220840	1	1227220840	31,75	0,000
POPSIZE	1927654560	1	1927654560	49,86	0,000
MAXPOP	627897760	1	627897760	16,24	0,000
CROSSP	3240	1	3240	0,00	0,993
MUTP	1193774760	1	1193774760	30,88	0,000
POPTYPE * POPSIZE	113973760	1	113973760	2,95	0,088
POPTYPE * MAXPOP	167772160	1	167772160	4,34	0,039
POPSIZE * MAXPOP	817960	1	817960	0,02	0,885
POPTYPE * POPSIZE * MAXPOP	15825640	1	15825640	0,41	0,523
POPTYPE * CROSSP	1428840	1	1428840	0,04	0,848
POPSIZE * CROSSP	104716960	1	104716960	2,71	0,102
POPTYPE * POPSIZE * CROSSP	113434240	1	113434240	2,93	0,089
MAXPOP * CROSSP	24211360	1	24211360	0,63	0,430
POPTYPE * MAXPOP * CROSSP	47349760	1	47349760	1,22	0,270
POPSIZE * MAXPOP * CROSSP	16537960	1	16537960	0,43	0,514
POPTYPE * POPSIZE * MAXPOP * CROSSP	5867560	1	5867560	0,15	0,697
POPTYPE * MUTP	503816040	1	503816040	13,03	0,000
POPSIZE * MUTP	6658560	1	6658560	0,17	0,679
POPTYPE * POPSIZE * MUTP	15976960	1	15976960	0,41	0,521
MAXPOP * MUTP	4733440	1	4733440	0,12	0,727
POPTYPE * MAXPOP * MUTP	1797760	1	1797760	0,05	0,830
POPSIZE * MAXPOP * MUTP	94003560	1	94003560	2,43	0,121
POPTYPE * POPSIZE * MAXPOP * MUTP	32041000	1	32041000	0,83	0,364
CROSSP * MUTP	1989160	1	1989160	0,05	0,821
POPTYPE * CROSSP * MUTP	46915560	1	46915560	1,21	0,273
POPSIZE * CROSSP * MUTP	91445760	1	91445760	2,37	0,127
POPTYPE * POPSIZE * CROSSP * MUTP	1024000	1	1024000	0,03	0,871
MAXPOP * CROSSP * MUTP	9682560	1	9682560	0,25	0,618
POPTYPE * MAXPOP * CROSSP * MUTP	3317760	1	3317760	0,09	0,770
POPSIZE * MAXPOP * CROSSP * MUTP	71289000	1	71289000	1,84	0,177
POPTYPE * POPSIZE * MAXPOP * CROSSP * MUTP	3069160	1	3069160	0,08	0,779
Error	4948159936	128	38657500		
Total	101587870960000	160			
Corrected Total	11424407640	159			

The 95% precision level is applied during the analysis. The corresponding F statistic is 3.915147 for 0.05 significance level. The factors, which are found significant are; initial population type, population size, maximum generation number, mutation probability and the interaction effect of population type and mutation probability.

The normal probability plot of effects is graphed in order to make the diagnostic checking of the ANOVA results gathered. Figure 8 resembles the normal probability plot of effects, estimated from the data set. The graph shows linear behavior except five outlier points, four of which are in the positive side, and the remaining in the negative side. These points correspond to the significant effects found from the analysis of variance, thus the correctness of the analysis is validated.

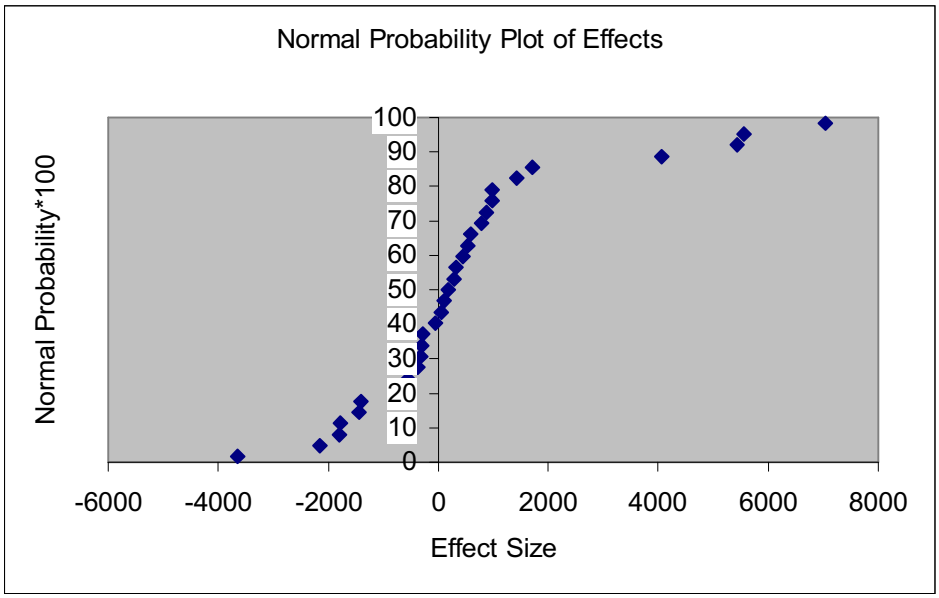


Figure 8: Normal Probability Plot of Effects for Fitness Response

Before giving a detailed explanation on the factors, it is beneficial to present some preliminary information about the solution space of the test problem under consideration. In the test problem we used, there are two main types of decision variables; M_i, number of machines in workstation i and B_i, number of buffer positions between station i and i +1. Our computational experiments with the model indicate that the number of machines in each workstation (M_i) is the dominating decision variable set, meaning that, the total profit obtained merely depends on the level of this variable. There are two reasons; one is the machine operating cost is significantly larger than the buffer cost (25 times larger), so any alteration of M_i has more significant impact on total cost of the system than the buffer. Also the unit cost of machine is high enough to affect the profit of the system. Another reason is even a unit increment or decrement on one of M_i 's affect the throughput of the system drastically compared to B_i. For these reasons M_i's constitute the dominating set of variables in the system considered.

Moreover, only some specific combinations of M_i's lead to good solutions (high profit). 160 independent GA applications with various parameter settings are carried out in the analysis, and the following scheme of the machine combinations is obtained in the best solutions found by GA:

Solution	# Times Encountered
3 – 4 – 2 – 3	81 times
4 – 4 – 2 – 3	49 times
3 – 4 – 3 – 3	12 times
3 – 4 – 2 – 2	8 times
4 – 4 – 3 – 3	5 times
3 – 4 – 3 – 2	3 times
4 – 4 – 4 – 3	1 time
3 – 4 – 2 – 4	1 time

As seen from the combinations, good solutions have some specific pattern in terms of machine numbers. More than 80% of trials, GA leads to a best solution with machine combination of 3 – 4 – 2 – 3 or 4 – 4 – 2 – 3. (It is a predictable result, since the first and the second workstations have the largest processing times, so they require more resources (machines) compared to the other workstations). Although low levels of buffer position lead to significant decrease in throughput so the profit of the system; for these good combinations of machines, buffer levels do not have very significant impact on the system performance. The following solutions and the corresponding objective values present an example to the situation:

Solution		Fitness Value (monetary units)
3 – 4 – 2 – 3 – 12 – 16 – 6	→	800680
3 – 4 – 2 – 3 – 16 – 15 – 9	→	794920
3 – 4 – 2 – 3 – 15 – 16 – 2	→	809400
3 – 4 – 2 – 3 – 16 – 12 – 8	→	800760
3 – 4 – 2 – 3 – 16 – 13 – 5	→	801320
3 – 4 – 2 – 3 – 13 – 16 – 10	→	795480
3 – 4 – 2 – 3 – 16 – 14 – 4	→	800840
3 – 4 – 2 – 3 – 12 – 13 – 2	→	803120

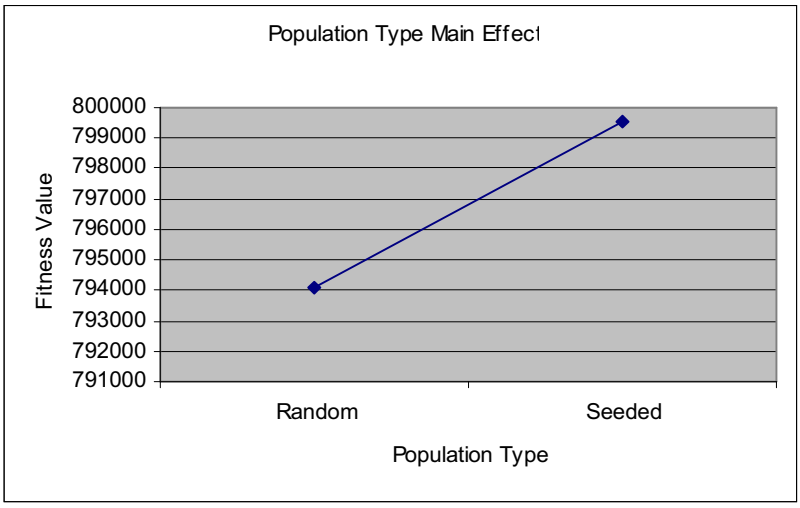
3 – 4 – 2 – 3 – 14 – 11 – 4 ⟶ 799760

The same combination of machine numbers, but different buffer levels does not present significant alterations in total profit. Because the cost of buffer position added is almost nearly compensated by the increase in the throughput level. But somehow there is also some discrimination in buffer levels among the good solution patterns. For example, for 3 – 4 – 2 – 3, third buffer level is smaller that the other two, which is an expected result. Because, third buffer is located between third and fourth station, and third station has only 2 machines, there is no need for a lot of buffer positions, which carry the WIP coming from the third station. As the first two stations are bottlenecks and have biggest number of machines, first two buffer levels are greater in some extent.

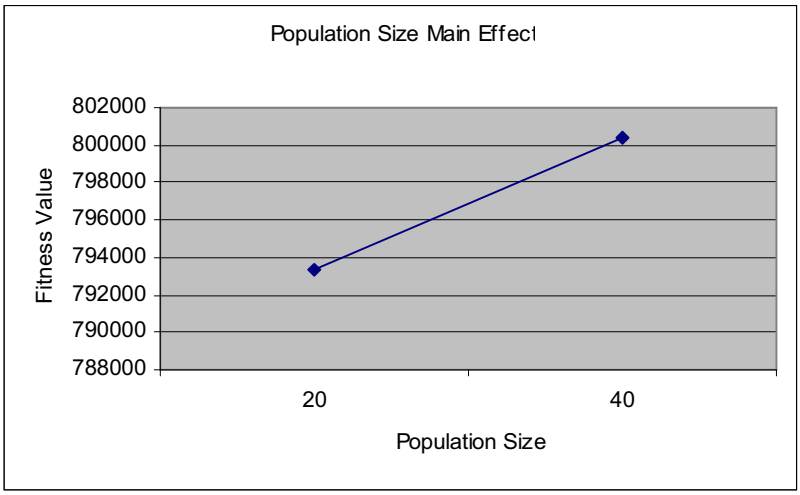
Thus, it can be concluded that good solutions have specific patterns in terms of machine combinations, and different buffer levels at these combinations create the diversity among the solutions with little improvements on objective value.

Another interpretation is that, good solutions are highly dominant in the solution space. In other words the solutions with the pre-specified combination of machines generally have significantly more fitness value than the other solutions. On the average the total profit of the system with good patterns can be estimated as nearly 780000, while no other combination of machines can reach such a profit value. The importance of this property reveals in GA application. Since some solutions, some good solutions, are very dominating in terms of fitness values, the selection procedure usually selects strings with these patterns for the potential parents of the following generation. Thus by the crossover operator, genetic material of these solutions is carried to the next generations, where a rapid convergence to these good solutions occur in GA. Since every time children are produced from the parents with same machine number patterns. After some initial generations, almost all individuals have these specific patterns in terms of machine numbers. Hence, this leads to the rapid convergence of GA to good solutions in few generations.

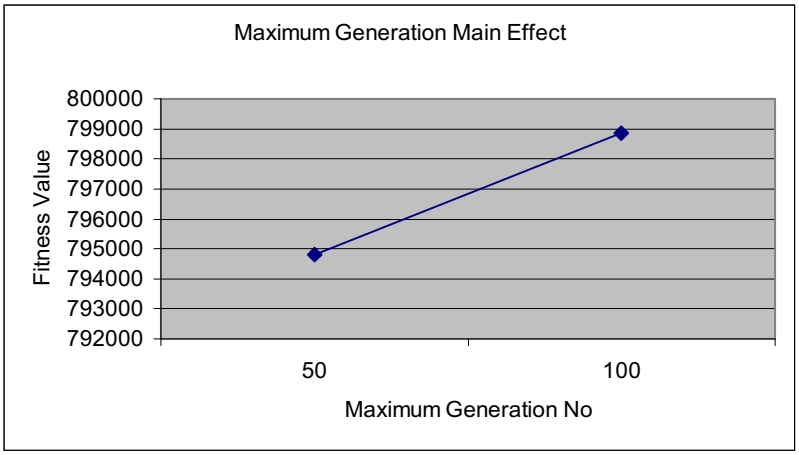
After these preliminary analyses about the solution space, the results of ANOVA can be presented. Figure 9.a-d resembles the significant main effects of the analysis, showing the change in fitness value going form the low level of a factor the high level. Figure 9.e presents all the main effects together, while Figure 9.f is the graph of only significant interaction effect, initial population type-mutation probability interaction.



a) Population Type Main Effect Graph on Fitness

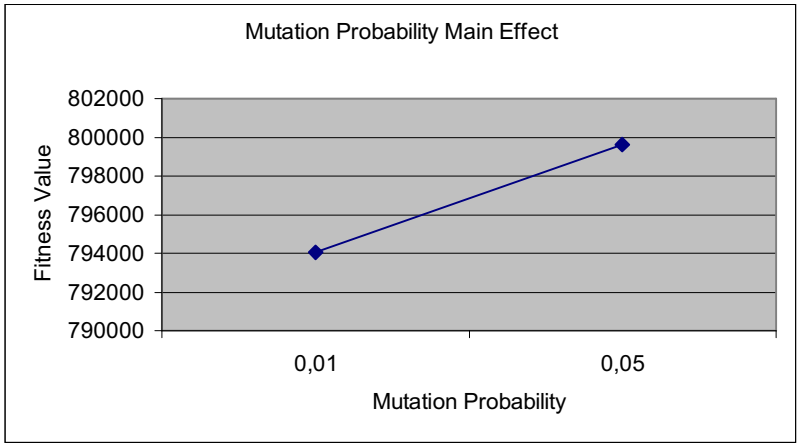


b) Population Size Main Effect Graph on Fitness

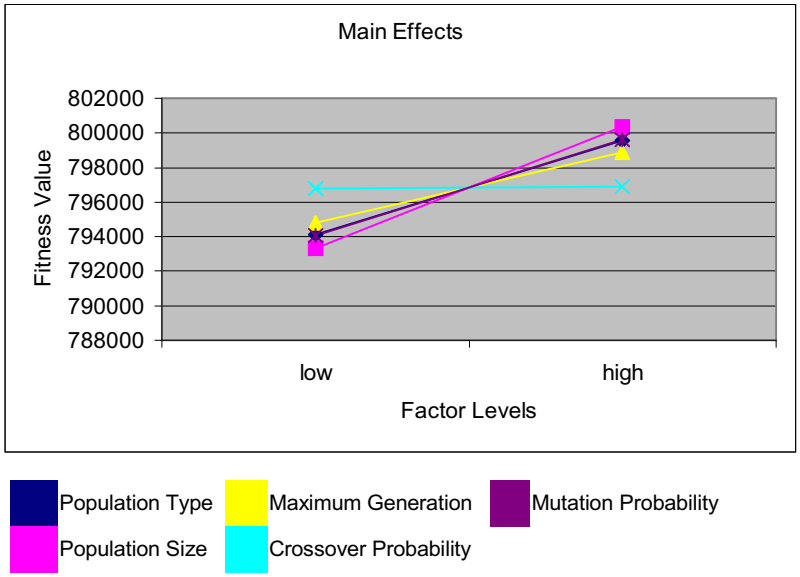


c) Maximum Generation Number Main Effect on Fitness

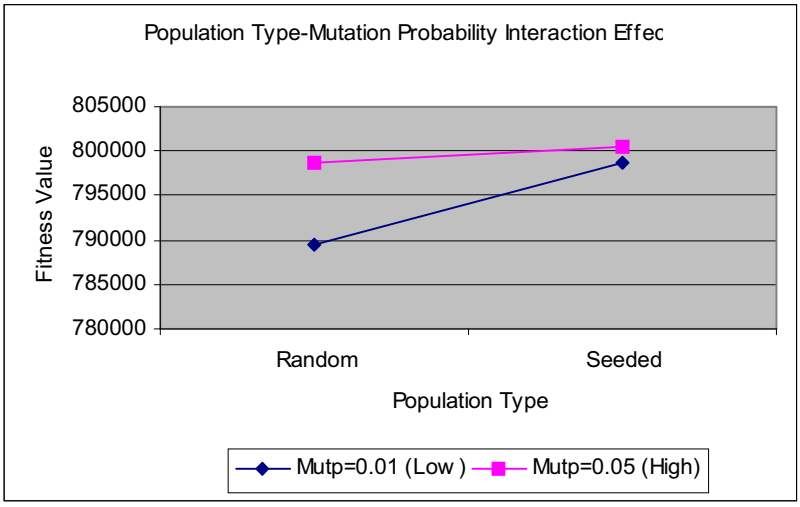
Figure 9. Graphs of Significant Effects with respect to Fitness Response



d) Mutation Probability Main Effect on Fitness



e) All Factors Main Effects on Fitness



f) Population Type-Mutation Probability Interaction Effect on Fitness

Figure 9. Graphs of Significant Effects with respect to Fitness Response (cont'd)

The initial population type is the first significant main effect. Starting with a seeded population increases the best fitness value found significantly compared with a random start, thus the initial population type has a positive effect on fitness response. (Figure 9.a) Although it is not a usual case for GA applications, there are empirical studies in the literature supporting such a conclusion (Reeves, 1992). In fact, since the good solutions have some specific pattern in terms of machine numbers, the goodness of the solution merely depends on the levels of the buffer positions after some generations. High dominance of good solutions leads to convergence to these good solution patterns, and GA starts to browse better solutions by changing the buffer levels. It is certain that, the machine numbers are also altered during genetic operations, but any alteration results with a poor fitness, thus selection mechanism ignores such individuals, and they do not have chance to survive for the following generations.

The difference between seeded population, and random population is, seeded one contains individuals with good machine combinations. Thus, because these solutions are highly dominant, the convergence of individuals to solutions with these patterns occurs very rapidly. For the random population case, GA spends some generations in reaching these good patterns, that is converge occurs more slowly. For seeded initial populations case, because of the more rapid convergence, GA has more time to try different combination of buffer levels, this increases the probability of hitting a better solution. GA uses its time more efficiently compared with the random case. For that reason initial population type has a significant impact on fitness value.

Population size and maximum generation number have also significant effects on best fitness value found. (Figure 9.b and 9.c) As you increase the levels of these factors, GA manages to find better solutions, thus they both have positive effects. This is a predictable, and also obvious result, since increasing the population size (n), or generation number (m) enlarges the search space. More individuals are processed, so probability of reaching better solutions increases. Hence, increase in levels of n and m, increases the performance of GA.

Crossover probability is the only factor, which has an insignificant main effect. From Figure 9.e, it seems that all the lines except crossover line show increasing or decreasing behavior, but crossover line is very straight between low and high values. The reason behind this depends on the nature of solution space. Good solutions have specific patterns in terms of

machine numbers, and they are highly dominant, so selection mechanism always selects these individuals, and convergence occurs after some starting generations. Low level of crossover probability means low chance to cross, in other words selected potential parents are copied to next generation directly. High level of crossover forces these potential parents to change genetic material with each other. If the effect of crossover is insignificant, then to cross or not to cross should not make any difference in fitness values. This is true for the nature of the solution and search space together.

After some initial generations, the individuals generally have 3 – 4 – 2 – 3 or 4 – 4 – 3 – 3 as machine combinations, corresponding to strings starting with:

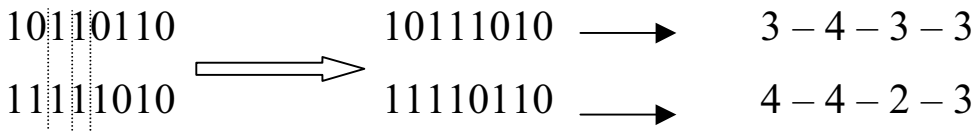
10110110……

11111010……

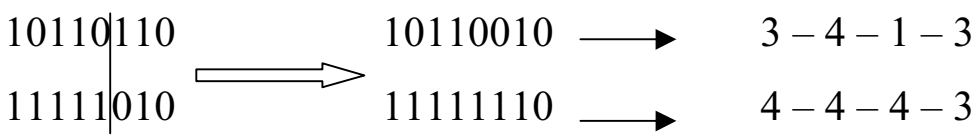
Although there are more patterns present in the solution space, and each pattern are in different amounts, since these two are the most common ones having largest fitness values, for a rough estimate, both strings have 50% occurrences in a population. Then for 2/3 of trails, reproduction operator selects two parents with same starting strings.

If selected two potential parents both have on of these patterns, then the crossover before the 8^{th} bit produce same individuals. Any cross-site after that point, generates individuals with different buffer levels with same machine numbers. If somehow an inappropriate level (very low) of buffer is generated, then the fitness decreases drastically, so dominance of good solutions make this individual disappear in the next generation, thus on the average, crossover cannot produce poor individuals. In the other case crossover generates individuals with some pattern of buffer levels, which are not poor. From the preliminary analysis, we know that different buffer levels with good machine patterns do not have very significantly different fitness values. Thus crossover cannot make significant discrepancy on fitness.

If selection mechanism selects both strings as potential parents (1/3 of trials), then crossover after 1^{st}, 6^{th}, 7^{th}, and 8^{th} bit does not produce distinct children. The crossover form the 2^{nd}, 3^{rd} and 4^{th} site generates the following pattern:



The cross-site of 5th bit produces the following strings:



From the generated strings all except 3 – 4 – 1 – 3 combination are present in good solutions' patterns, so they have close fitness values. The 3 – 4 – 1 – 3 combination results with a poor fitness value because of the high decline in throughput reasoning from the third workstation's bottleneck situation, and will disappear in the following generations because of high dominance of good solutions.

To sum up, there is no significant difference between the low level and high level of the crossover probability in terms of fitness, since it generates similar individuals having close fitness values. Even for most cases to cross or not do not make any difference, because the generated children are same with their parents. This is the main reason of insignificance of crossover operator on fitness value.

As Figure 9.e resembles, the mutation probability has a significant positive effect on fitness value. The power of mutation comes from its ability to search for various combinations of buffer levels. After the convergence to good solutions with specific machine number patterns, the improvements on fitness value is gathered by searching for different levels of buffer positions. Since mutation alters the value of a bit form 1 to 0, or vice versa, it easily provides diversity in solutions. It is obvious that, mutation can also take place in the first eight digits of the strings, but such a situation results with a poor performance of the system, because of deviation from the good machining pattern, and the dominance of the solutions distinguish the mutated individual in the next generation, so on the average mutation cannot produce poor individuals. Any mutation after 8th bit, leads to a different combination of buffers, which might have a better profit. While more combinations are searched, the probability of hitting a better solution increases automatically.

A question appears in our minds; although mutation is powerful in producing diverse solutions in terms of buffer levels, why does the crossover operator have insignificant impact?

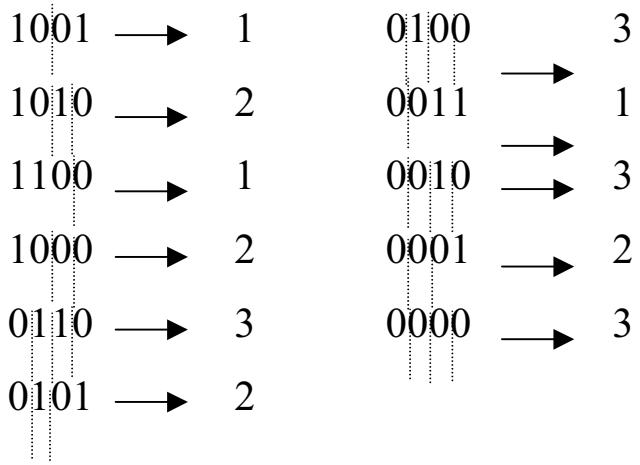
The main reason is that, the crossover operator generally produces same children with their parents as stated in the above paragraphs. We use the following example to explain this behavior:

Consider the first buffer position, and the high level of it (16) corresponding to "1111" binary representation. By mutation, all levels of this variable can be reached by bit mutations occurring at the same time, but since p_m is too small, it is rational to assume that only one mutation occurs at a time. Thus 1111 can be changed to the following strings and real values with probability p_m separately;

0111 ⟶ 8 1011 ⟶ 12 1101 ⟶ 14 1110 ⟶ 15

Hence, the probability to obtain a different buffer combination is $4 \times p_m$. (In fact the original probability of obtaining different buffer combination is $1 - (1 + p_m)^4$, which is larger than $4 \times p_m$. Thus using the original probability will lead a stronger conclusion).

To obtain different values of this variable, several conditions must hold for crossover operation. First the other individual should have a different representation than "1111". In fact the other parent should have binary representation of this buffer level other than "1111, 0111, 1011, 1101, 1110", since these representations cannot create buffer level different from the parents. The second condition is the cross-site should be selected in between this string; other cross-sites do not change the value of this variable, since it is copied to the child. To clarify the situation the following scheme presents the possible strings with possible crossover sites, where crossover can generate different value. The dashed lines are the possible crossover sites, and the numbers indicate the possible amount of crossover sites.



Although every string is found in different proportions in the individuals of GA, for a rough estimate we can assume that each string has 1/16 probability of occurrence, since there are 16 different string representations. Then for a single string, the probability of crossing with "1111" and producing different children can be calculated as the probability of selection of that string (1/16) × crossover probability × number of possible cross-sites /20, where 20 is the string length. For string "0000" this probability equals $1/16 \times p_c \times 3/20$, where same probability is $1/16 \times p_c \times 1/20$ for string "0011".

Since first buffer is located between first and the second stations, and first station is one of the bottlenecks, good solutions include large amount of buffer positions, at least the number of buffer positions is greater than 5, the strings on the right side can rarely be selected by the reproduction operator, so we can discard them in calculation. Hence, the probability to cross with "1111" and produce different children equals to the sum of distinct probabilities, which is equal to $1/16 \times p_c \times 11 / 20$.

For the best case (high crossover probability) this number contributes to 11 /320. For the worst case (low level of mutation), the probability of same operation with mutation is equal to $4 \times p_m = 4 /100$. Even for the worst case mutation has a higher chance to generate diverse individuals in buffer levels, since 4 /100 > 11 /320. Thus mutation is significantly effective on fitness, although crossover is not.

To sum up, once the good solutions are achieved, the population members are similar to each other in terms of machine numbers. The increase in mutation probability, allows GA search more combination of buffer levels, which automatically increases the GA performance.

The only significant interaction effect is the interaction between mutation probability and initial population type. For the low level of mutation probability, the improvement in the fitness value going from random initial population to seeded population is more significant than the high level of p_m (Figure 9.f). The initial population type has a significant main effect on fitness, which is stated above. The mutation operator is the relevant tool for improving the solutions, by searching different levels of buffer levels. When p_m is low, less mutation occurs, so alterations of strings decrease, and only the strings generated by crossover mechanism becomes effective on the fitness response, thus the power of seeded population becomes visible. Starting with good solutions lead to very rapid convergence, and GA has more time to search for different levels of buffers compared to random population. But for the high mutation rates, more mutation occurs, so the difference between initial population types is

compensated. As more mutation proceeds, the diversification of solutions occur, so random solutions can also hit better solutions in terms of buffer combinations.

As a conclusion, according to the factorial analysis considering the fitness response, the initial population type, population size, maximum generation number and mutation probability presents significant behavior, and all of them have positive effect. Thus, it is better to set these variables to their high levels to maximize the fitness value. Since no conclusion is gathered for crossover probability any level can be used. The only interaction effect claims in parallel with the main effects, so it is convenient to set all parameters except crossover probability to high level.

5.1.3. CPU Time Response

The ANOVA table for the CPU time response is presented as Table 10. According to %95 precision level, the following factors become significant as a result of F test: population size, maximum generation number, mutation probability, and interaction effect of population size and maximum generation number.

Table 10: ANOVA Results for CPU Time Response

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	8500190197,60	31	274199683,79	109,67	0,000
Intercept	34684395129,85	1	34684395129,85	13872,25	0,000
POPTYPE	6423020,66	1	6423020,66	2,57	0,111
POPSIZE	3929259559,70	1	3929259559,70	1571,53	0,000
MAXPOP	3990321062,21	1	3990321062,21	1595,96	0,000
CROSSP	28197,16	1	28197,16	0,01	0,916
MUTP	107683013,17	1	107683013,17	43,07	0,000
POPTYPE * POPSIZE	1175,81	1	1175,81	0,00	0,983
POPTYPE * MAXPOP	304493,78	1	304493,78	0,12	0,728
POPSIZE * MAXPOP	436856224,40	1	436856224,40	174,72	0,000
POPTYPE * POPSIZE * MAXPOP	24715,32	1	24715,32	0,01	0,921
POPTYPE * CROSSP	2435,00	1	2435,00	0,00	0,975
POPSIZE * CROSSP	850404,75	1	850404,75	0,34	0,561
POPTYPE * POPSIZE * CROSSP	18437,15	1	18437,15	0,01	0,932
MAXPOP * CROSSP	91908,65	1	91908,65	0,04	0,848
POPTYPE * MAXPOP * CROSSP	266530,11	1	266530,11	0,11	0,745
POPSIZE * MAXPOP * CROSSP	126162,32	1	126162,32	0,05	0,823
POPTYPE * POPSIZE * MAXPOP * CROSSP	110703,01	1	110703,01	0,04	0,834
POPTYPE * MUTP	531729,01	1	531729,01	0,21	0,645
POPSIZE * MUTP	11637101,93	1	11637101,93	4,65	0,033
POPTYPE * POPSIZE * MUTP	2667,83	1	2667,83	0,00	0,974

Table 10: ANOVA Results for CPU Time Response (cont'd)

MAXPOP * MUTP	12711051,69	1	12711051,69	5,08	0,026
POPTYPE * MAXPOP * MUTP	517114,42	1	517114,42	0,21	0,650
POPSIZE * MAXPOP * MUTP	683639,46	1	683639,46	0,27	0,602
POPTYPE * POPSIZE * MAXPOP * MUTP	548742,34	1	548742,34	0,22	0,640
CROSSP * MUTP	6702,40	1	6702,40	0,00	0,959
POPTYPE * CROSSP * MUTP	254139,39	1	254139,39	0,10	0,750
POPSIZE * CROSSP * MUTP	257278,39	1	257278,39	0,10	0,749
POPTYPE * POPSIZE * CROSSP * MUTP	50285,83	1	50285,83	0,02	0,887
MAXPOP * CROSSP * MUTP	211923,72	1	211923,72	0,08	0,771
POPTYPE * MAXPOP * CROSSP * MUTP	87003,19	1	87003,19	0,03	0,852
POPSIZE * MAXPOP * CROSSP * MUTP	296180,66	1	296180,66	0,12	0,731
POPTYPE * POPSIZE * MAXPOP * CROSSP * MUTP	26594,13	1	26594,13	0,01	0,918
Error	320034691,42	128	2500271,03		
Total	43504620018,87	160			
Corrected Total	8820224889,02	159			

The normal probability plot of effects is taken for the diagnostic checking purposes. Figure 10 illustrates the graph. The graph shows linear behavior except four outlier points, three of which are in the positive side, and the remaining in the negative side. These points correspond to the significant effects found from the analysis of variance, thus the correctness of the analysis is validated.

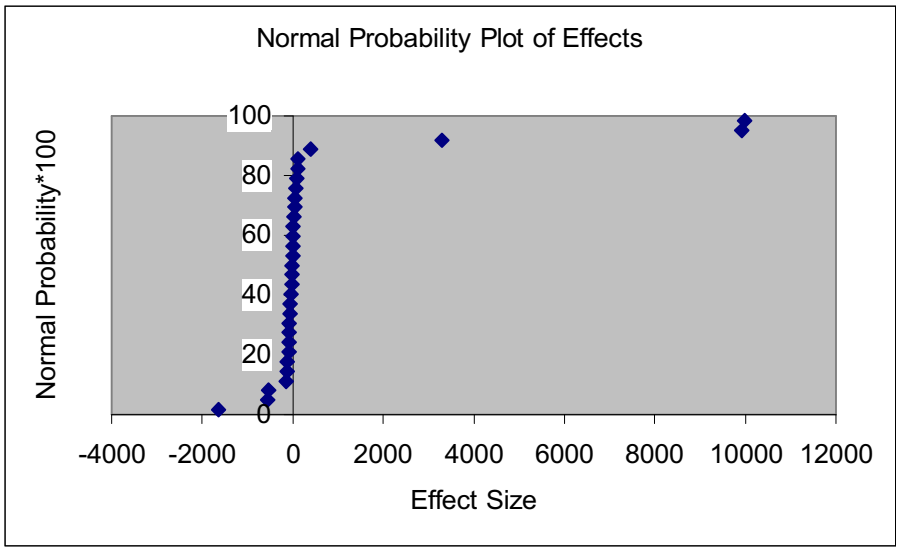


Figure 10: Normal Probability of Effects for CPU Time Response

Again, before any interpretation of the results, it is beneficial to make some preliminary analysis about the solution space, with respect to CPU time. First of all, the CPU time depends on the simulation time of individual points. Although the processing of GA requires some time, the simulation time of the solutions generated by GA constitute to almost all CPU

time (more than 95% of CPU time). Thus, the factors effecting the CPU time, in fact, alter the simulation time.

The simulation time is dependent on the load of the system. As the machine numbers and buffer levels increase, the simulation time increases in parallel, since more production occurs in the model. Like the fitness case, machine numbers are also dominating set of decision variables according to the CPU time. The main reason is, the production rate merely depends on the machine combinations, and since the simulation time is proportional with the production rate, it is also associated with the machine combinations. Buffer levels do not have significant effect on simulation time, except some extreme conditions (very low levels of buffer affecting the production process).

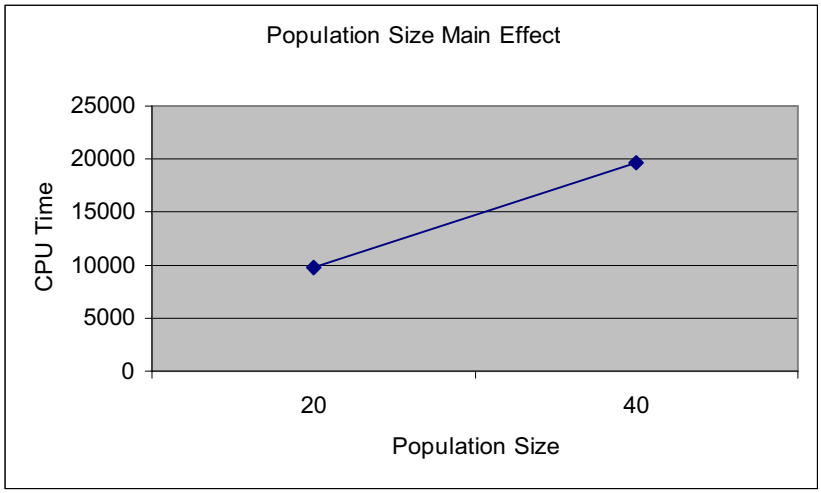
As stated in the fitness analysis, GA converges to some good solutions having specific patterns of machine combinations rapidly. These good solutions correspond to loaded systems, having more number of machines from the average, hence their simulation runs generally requires more time compared to other solutions, because more production occurs compared with the system having average values of variables. Going from these good patterns to a higher level of machine combination increases the simulation time, but however any negative deviation from the good patterns leads to significant decrease in simulation time, because the throughput of the system decreases significantly. The following example illustrates the situation:

Solution		CPU Time
3 – 4 – 2 – 3 – 15 – 13 – 2	⟶	0.23 minutes
2 – 4 – 2 – 3 – 15 – 13 – 2	⟶	0.16 minutes
4 – 4 – 2 – 3 – 15 – 13 – 2	⟶	0.24 minutes

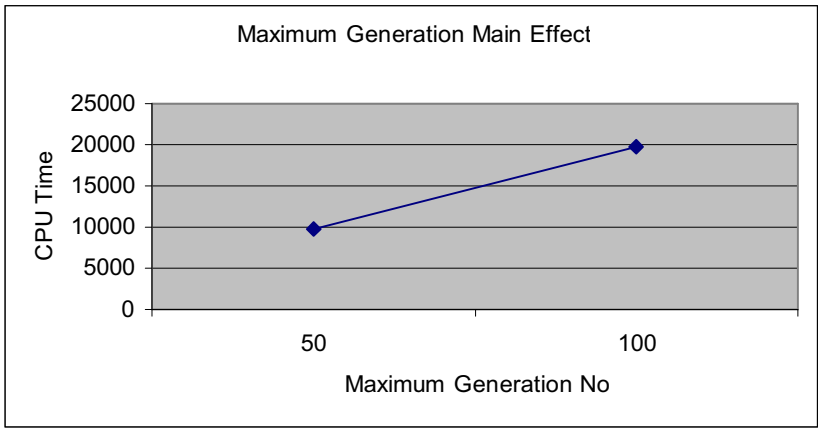
First combination is a good solution, and has a simulation time of 0.23 minutes, which is the average of five replications. Decreasing the number of machines in first workstation by one unit, results with a significantly less-loaded system having 0.16 minutes of simulation time; while a positive alteration of same variable increases the simulation time to only 0.24 minutes.

After these preliminary explanations, the results of ANOVA are presented below. Figure 11.a represents the change in CPU time with respect to change in population size. Figure 11.b shows the effect of maximum generation number, and Figure 11.c represents the mutation probability effect. Figure 11.d is the graph of all main factors, and finally Figure

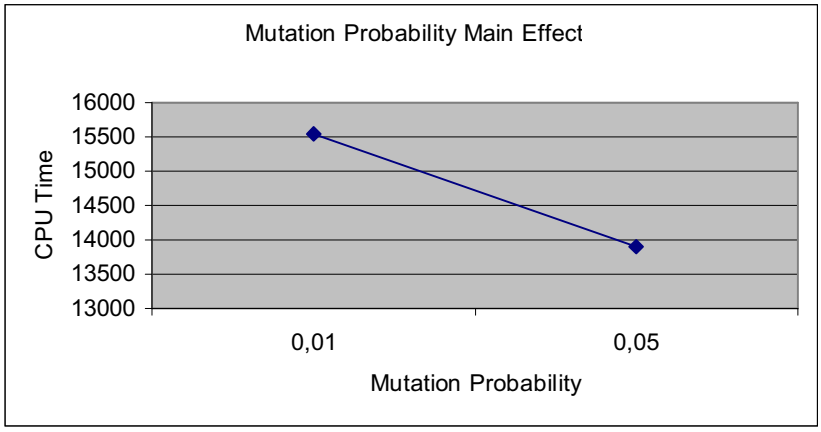
11.e resembles the only significant interaction effect, population size-maximum population interaction.



a) Population Size Main Effect on CPU Time

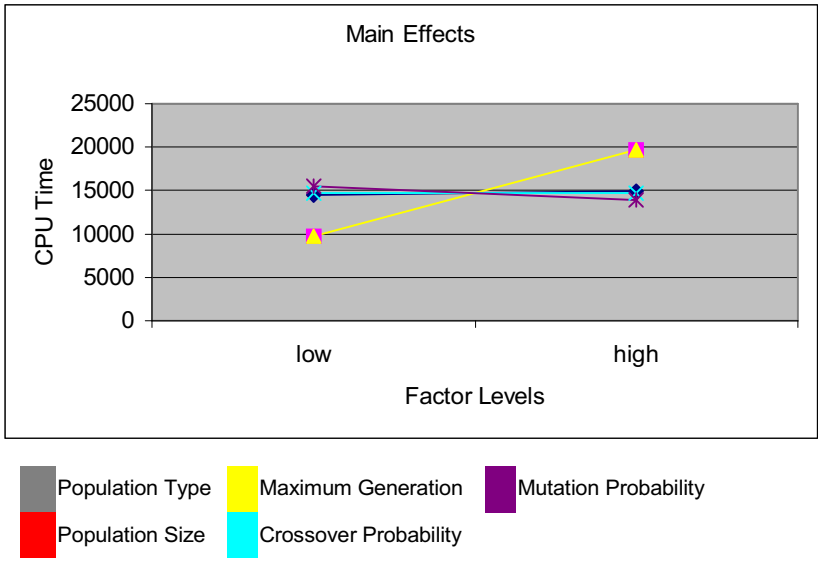


b) Maximum Generation Number Main Effect on CPU Time

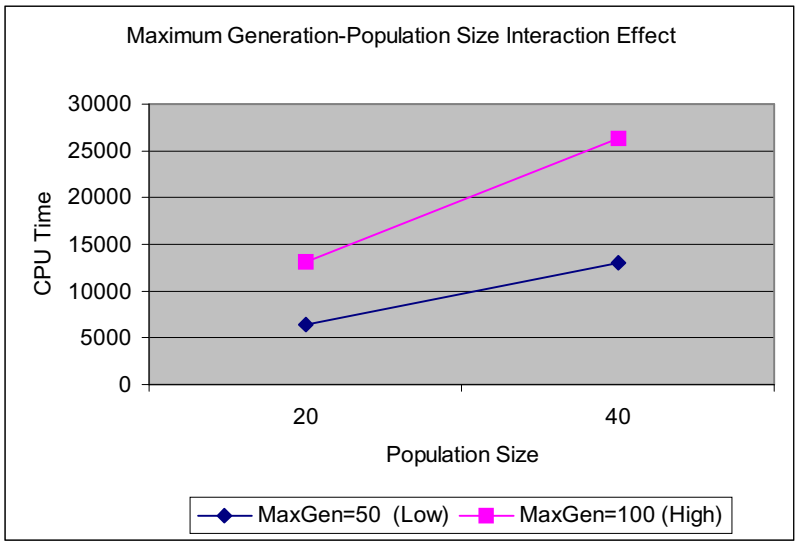


c) Mutation Probability Main Effect on CPU Time

Figure 11: Graphs of Significant Effects with respect to CPU Time Response



d) All Factors' Main Effects on CPU Time



e) Maximum Generation Number-Population Size Interaction Effect on CPU Time

Figure 11: Graphs of Significant Effects with respect to CPU Time Response (cont'd)

The initial population type is insignificant with respect to CPU time, which can be observed from the Figure 11.d. It has a straight-line representation, meaning that no alteration in CPU time occurs according to the change of level from low to high value. The reason behind this situation is the rapid convergence of the solutions to good patterns. Good solutions are loaded systems, and regardless of the buffer combinations good patterns have nearly equal simulation times. Although seeded populations converge to good solutions more rapidly, a random start looses only a few generations time to converge to the same good set of solutions, and on the average these few generations' individuals' simulation time does not

make significant impact on CPU time. At some initial generations, average CPU time covered is more at seeded populations compared with the random ones, since random populations have more poor solutions corresponding to less-loaded systems, with less simulation time. But when all the generations are compared, this discrepancy cannot show significant changes on CPU time.

The first significant effect is the population size. It has a positive effect on CPU time, which is an obvious result, because an increment in population size, directly increases the number of individuals processed, so the number of simulation runs taken. Thus increase in level of population size increases the CPU time (Figure 11.a).

The maximum generation number factor has same impact with the population size parameter. Like the population size, high level of maximum generation number doubles the number of points processed in GA application, obviously the CPU time increases substantially, going from low level to high level (Figure 11.b).

The crossover probability is insignificant according to CPU time (Figure 11.d), as in the case of fitness response. The reasons for the insignificance of the fitness response are also valid for the CPU time case. Generally to cross or not does not make any sense, because in each case the produced children are same, or carry same structural properties. Crossover always produces individuals with good specific pattern of machine numbers, because any individual objected with this is eliminated by the selection procedure in the following generations, as explained in fitness part. The only difference between the solutions is their buffer levels. However, except for some extreme cases (very low levels of buffers), which are rarely or never experienced, all the buffer combinations lead to a semi-loaded simulation model, thus the simulation time, so the CPU time are very close to each other. The combination examples used in the fitness analysis can clarify the situation:

Solution		CPU Time
3 – 4 – 2 – 3 – 12 – 16 – 6	⟶	0.22 minutes
3 – 4 – 2 – 3 – 16 – 15 – 9	⟶	0.23 minutes
3 – 4 – 2 – 3 – 15 – 16 – 2	⟶	0.23 minutes
3 – 4 – 2 – 3 – 16 – 12 – 8	⟶	0.23 minutes
3 – 4 – 2 – 3 – 16 – 13 – 5	⟶	0.23 minutes
3 – 4 – 2 – 3 – 13 – 16 – 10	⟶	0.23 minutes
3 – 4 – 2 – 3 – 16 – 14 – 4	⟶	0.23 minutes

3 – 4 – 2 – 3 – 12 – 13 – 2 ⟶ 0.23 minutes

3 – 4 – 2 – 3 – 14 – 11 – 4 ⟶ 0.23 minutes

All combinations except one has a simulation time of 0.23 minutes, which shows the similarity between the solutions having same machine number pattern in CPU time.

The mutation probability is the last significant factor on CPU time. Different from the other factors, mutation probability has a negative effect on CPU time (Figure 11.c). Increasing the mutation probability p_m decreases the CPU time, which is a desired result. Mutation gathers the diversity to the search space by the alterations it makes on the bits of individual strings. Because of dominant good solutions, rapid convergence occurs to some specific patterns of machine numbers. The preliminary analysis on the solution space reveals that negative deviations from the good machine combinations lead to more significant decrease in simulation time, when compared with the increase in simulation time in the case of a positive alteration of machine numbers, which carry the model to a more loaded one. Another interpretation about the algorithm is that after some starting generations all the populations are composed of individuals having combinations of 3 – 4 – 2 – 3 or 4 – 4 – 2 – 3 as machine numbers leading to strings starting with:

String 1: 10110110

String 2: 11110110

Mutation operator alters the value of bit from 1 to 0, or vice versa. A change from 1 to 0 leads a decrease in real value of corresponding variable, which corresponds a less-loaded system and the opposite results with an increased real value, which is a more loaded system. For string 1, the probability to go a more loaded system is 3/8 in terms of machine numbers, because there are 3 "0"'s present in the string. On the other hand a less-loaded system is achieved in 5 of 8 trials because of 5 "1"'s in the string. These probabilities are even more discriminating in string 2, because of six 1's and two 0's, corresponding to 6/8 and 2/8, respectively.

Since generally different levels of buffers do not have a significant impact on CPU time with good pattern of machine numbers, assuming that they have the same amount of simulation time, the CPU time decreases when the mutation probability increases, because in

most of the trials, mutation carries the solution to a less-loaded system. This result is proved by comparing the probabilities, for first type of strings in 5 of each 8 trials system becomes a less-loaded one, and for second string in 6 of 8 trials same result is obtained. Also the significant difference between the alterations from a good solution to a more loaded system, and from a good solution to a less loaded system has a positive effect on the decrement of CPU time, which is stated in the preliminary analysis.

The only significant interaction effect is the interaction between population size and maximum generation number. For high level of maximum generation number, the difference between the high and low level of population size in terms of effect on CPU time is larger than the low level (Figure 11.e). The reason is that higher the iteration number, larger the number of loaded systems (good solutions) present in the population; since more individuals converge to the same set of good solutions contributing to loaded systems and the simulation time of loaded systems are longer than the other solutions. Thus, for high level of maximum generation number the increase in CPU time by the alteration of population size is more significant. From Figure 11.e, it is observed that high level of both factors lead to the maximum CPU time.

From all these analysis, the final conclusion is; the population size and maximum generation number has positive effects on CPU time. Since we are trying to minimize the CPU time, low levels of these parameters are appropriate. Because the mutation has negative effect on CPU, it is better to set it at its high level. As the crossover probability and initial population type do not have significant impact on CPU time, they can be used in any level.

5.1.4. Conclusion

ANOVA is applied to both fitness and CPU time measures independently, and the significant factors on responses are determined. According to the results gathered, appropriate levels of each numerical parameter are selected. Table 11 summarizes the results.

There are two conclusions, which take our attention. One of them is the insignificance of crossover probability for each response types; and best performance of the high mutation probability in both cases. What is interesting here is that, the power of mutation is greater than the crossover on both performance measures. But it is obvious that the conclusions are relevant for a factor analysis with some pre-specified levels. Thus, we decide to make further

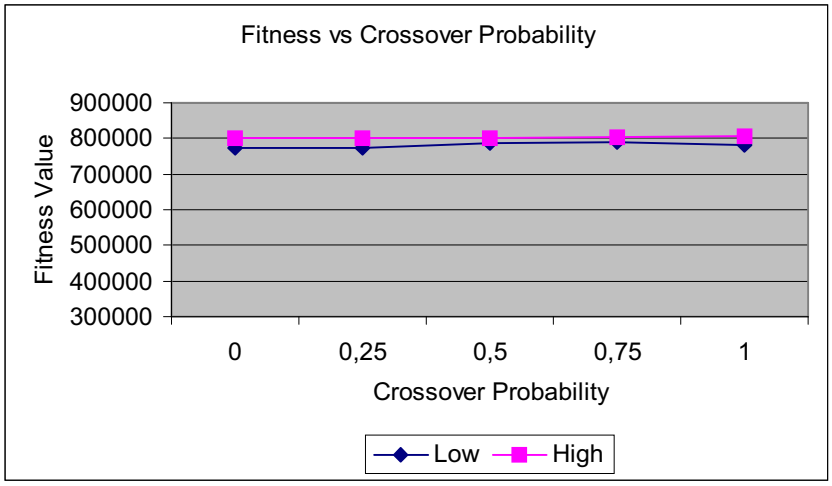
analysis of mutation probability, and crossover probability to observe the behavior at their

Table 11: Results of Factorial Design Analysis

	Pop Type	Pop Size	Max Gen	Cross P	Mut P
	Fitness				
Level	High	High	High	-	High
Value	Seeded	40	100	-	0,05
	CPU Time				
Level	-	Low	Low	-	High
Value	-	20	50	-	0,05

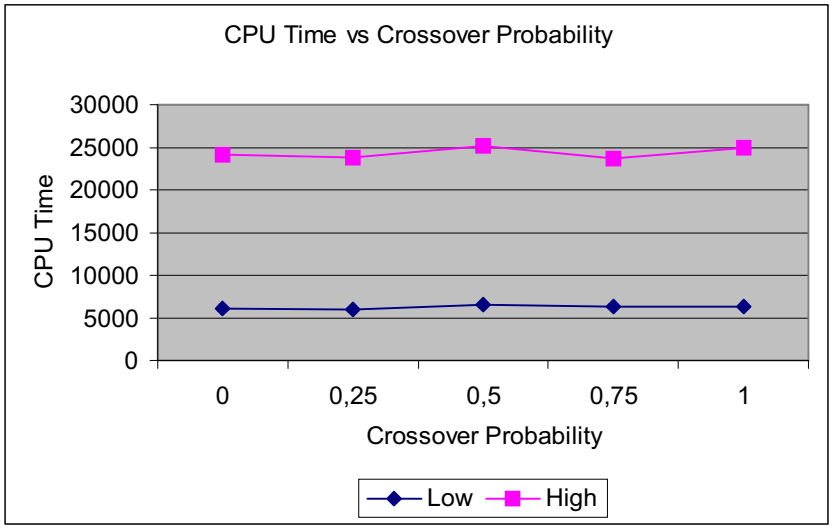
different levels. Since it is a very time consuming task to repeat all the analysis with different levels of these parameters, single factor analysis is chosen for the experimentation. In other words all the other parameters are set to some value, and change in performance measures is examined according to the change in the level of the parameter under consideration. Two distinct experimental conditions are chosen, all the parameters are set to their high levels and low levels separately. Statistical inference about the results is obtained by comparing the performances using paired-t test.

First, we investigate the significance of crossover probability. Our initial values were 0.5 and 1; we also examine the behavior of the performance measures under levels of $p_c = 0$, 0.25 and 0.75. Figure 12.a represents the change in fitness value due to the alteration of p_c, and Figure 12.b presents the same results for CPU time response.



a) Best Fitness with Different Levels of Crossover Probability

Figure 12: Analysis of Different Crossover Probabilities



b) CPU Time with Different Levels of Crossover Probability

Figure 12: Analysis of Different Crossover Probabilities (cont'd)

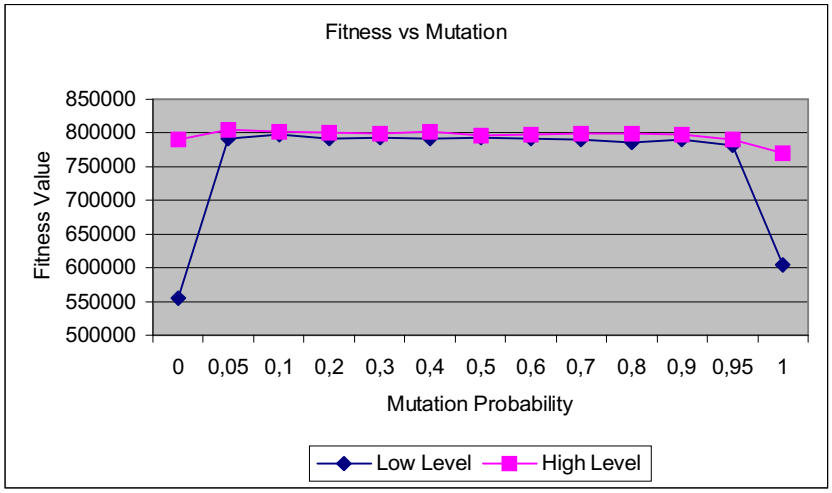
Each point in the graphs corresponds to the average of independent GA applications. For fitness value, as observed from Figure 12.a, there is no significant difference in performance measure for different levels of crossover probability. The paired-t test statistically validates this conclusion, because the difference of means of two successive points present an insignificant behavior.

The CPU time graph also shows the insignificance of the crossover probability, which is also validated by paired-t test results of every consecutive pairs. For design point 2 (all parameters at their high level), alteration of crossover probability do not change the CPU time significantly, it goes around 25000 seconds. For design point 1 (parameters at their low levels), since the population size and maximum generation number are at their low levels, the CPU time is significantly lower, but again the change in crossover probability does not have any significant effect on the performance.

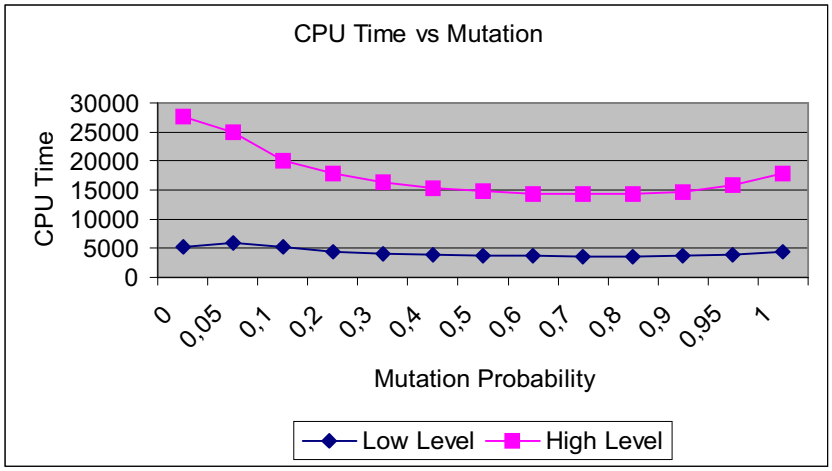
This further analysis backs up the previous results of our factorial design, and also goes in parallel with the preliminary observations made about the solution space. Since there is a dominating set of variables, and the good solutions are defined to be good because they have some specific pattern of this set; and also because the good solutions are very dominating in terms of fitness value; after some initial generations individuals in populations converge to these specific pattern of solutions. Thus, to cross or not does not have any impact on the performance. If crossover is prohibited, this means the copying of selected potential parents to next generation occurs. If crossover takes place they exchange genetic information, but as

both parents have the same string characteristics, and even sometimes they correspond to same individual, exchange of genetic information does not make any sense, so in any level of crossover probability the GA shows the same behavior, reaching to close solutions at the end.

The same type of analysis is applied for different levels of mutation probability. Again two design points, other parameters at high and low levels, are considered, and 5 independent replications of GA are processed for each examination point. Figure 13.a presents fitness value according to different levels of mutation probability, while Figure 13.b shows the CPU time behavior.



a) Best Fitness with Different Levels of Mutation Probability



b) CPU Time with Different Levels of Mutation Probability

Figure 13: Analysis of Different Mutation Probabilities

Although it is not visible from the Figure 13.a, increase in mutation probability has a significant impact on fitness value. According to the results of paired-t test, design point 2

(high level) presents insignificant behavior until $p_m = 0.4$, excluding the $p_m = 0$ case. But there is a significant improvement on fitness going from 0.3 to 0.4 level of mutation probability. The other respective points present insignificant behavior except $p_m = 1$.

The $p_m = 0$ and 1 are the extreme cases with no mutation and always mutation. The results obtained agree with the consequences we gathered from the factorial design. The mutation operator is very effective in terms of finding better solutions. This idea can be observed best by looking at the design point 1 (low level) with $p_m = 0$. There is no mutation and GA cannot find even good solutions. The GA starts with random population and with fewer crossovers and no mutation, convergence to good solutions cannot occur. But this is not valid for the design point 2 (high level). The main reason is that, GA starts with seeded population containing good solutions, and although mutation is not applied, crossover operator provides the maintenance of good solutions. However, no mutation case also has significantly less fitness value than the other points, but the difference is not as obvious as the design point 1.

Increasing the mutation probability increases the fitness value until $p_m = 0.4$. The other probabilities are insignificant after this level when compared respectively until $p_m = 0.95$. There is a significant decrease in fitness at $p_m = 1$. This is because of the inadequate pattern of GA. GA finds good solutions and carry their good properties to future generations by crossover and reproduction, but if mutation occurs nearly all the time, the building blocks that belong to good solutions are always disturbed and changed, so GA cannot converge to good solutions. In other words as the generations proceed the average fitness of populations cannot increase as our case. The negative impact of “always mutation” is more visible in design point 1 (low level). The main reason is the initial population type difference. The random start cannot even reach good solutions like the other extreme case $p_m = 0$. But for seeded populations, since the initial population contains good solutions, at worst case these good starting solutions are the best solutions obtained during GA, which are better than the random start in terms of fitness values

Investigation of Figure 13.b shows that increments in mutation probability make significant reductions in CPU time covered. This result also agrees with our conclusions about solution space. As more mutation occurs more less-loaded systems are processed, so simulation time decreases. But after 0.9 level, there is an increase in CPU time, which is in correspondence with the fitness case. As the pattern of GA is disturbed, no convergence to

good solutions occur, and unpredictable points from anywhere of solution space are processed. The search mechanism looses its logic, so increase in CPU time is observed.

From the two graphs, it is concluded that even more increments in mutation probability lead better solutions in terms of maximum fitness and minimum CPU time. Mutation probability of 0.4 is the most convenient choice, since it has the maximum fitness and significantly less CPU time.

During the analysis, we have two objectives, maximizing the best fitness value found, and minimizing the CPU time elapsed. Table 11 shows the results of analysis made for each objective one by one. But to conclude on the appropriate levels of the factors, we have to combine the results. Obviously we can conclude on the mutation probability level as the high level, since the results of analyses in both cases are consistent with each other. Population type is insignificant for CPU time, and seeded population is better for fitness, so the initial population is set to be seeded population. Crossover probability is insignificant for both response types, so any level of the crossover probability can be used.

Unfortunately, population size and maximum generation number behave in an opposite way when the results of the analyses are considered. If you have the high level, you get better fitness value but in a long run time for both factors. Hence user should give the priority to one objective, and select the appropriate one. If an increment in fitness value is very important for the user, then s/he should set the factors to their high levels, regardless of the CPU time. Otherwise, if the time is the most important constraint, and it is expensive to run the algorithm, then user should set the factors to their low value. In our case, we give equal weight to both objectives; fitness and CPU time, and make the analysis based on this. Table 12 summarizes the analysis made.

Table 12: Percentage Change in Responses According to Change in Factor Levels

Pop Size	Fitness	CPU Time	Max Gen	Fitness	CPU Time
20	793305,5	9767,775	50	794795,5	9729,418
40	800347,5	19678,95	100	798857,5	19717,31
% Change	0,887678	101,4681	**% Change**	0,511075	102,6566

Percent changes reflect the percentage change in the response variable, when the relevant factor is increased form its low level, to high level. For population size, this increase leads to a 0.89% better fitness solution, while a 101.5% increase in CPU time. Similarly,

change in the maximum generation number leads to a 0.5% better fitness solution, but in 102.7% more CPU time. The increments in fitness values cannot compensate the extra CPU time covered, so it is better to set population size and maximum generation number to their low values.

Table 13 gives the final conclusion about the parameters of GA that maximizes the best fitness found and minimizes the CPU time.

Table 13: Final Result of Analysis

	Pop Type	Pop Size	Max Gen	Cross P	Mut P
Level	**High**	**Low**	**Low**	**High/Low**	**High**
Value	Seeded	20	50	1/0,5	0,05

The best parameter configuration of GA is starting with a seeded initial population with low level of population size, and maximum generation number; also the mutation probability should be as large as possible (0.05 in factorial design, 0.4 in further analysis), any crossover probability can be selected. But we have to remind that the selection of population size and maximum generation number depends on the choice between CPU time, and best fitness obtained. If best fitness obtained is much more important than the CPU time then it is better to set them to high levels.

Another interpretation is that, if crossover operator is insignificant on fitness response, and the good solutions dominate the others, and also some set of decision variables has some specific values in all good solutions, then the importance of mutation increases. Especially for simulation-optimization purpose, mutating the solutions in such a search space may have a significant effect on CPU time, and fitness value. If the system under consideration represents a loaded system for good solution set, then it is convenient to increase the mutation rate, which generally leads to reduction in CPU time. Also dominance of a decision variable may support the use of high mutation rates. If good solutions have some specific pattern in terms of dominant decision variables, then these solutions are discriminated by the levels of other decision variables. As in our case, mutation operator may be more successful in searching the different levels of other parameters than the other operators. In such a condition extra analysis should be done to set the mutation probability to the most convenient level.

CHAPTER 6

FURTHER ANALYSES

This chapter presents further analyses made by modifications of the test problem and proposed GA model. We will measure the sensitivity of the results to different experimental conditions. Section 6.1 presents the results for the analysis carried with the modified objective function cost parameters of the test problem. Section 6.2 explains the experimental results for GA-based sim-opt application to the constrained version of the same test problem; thus a linear constraint is added to the optimization problem. Section 6.3 presents the comparison of GA performance under different structural parameter settings.

6.1. Modification of Objective Function Parameters

The objective function parameters guide GA in the search mechanism. For instance if the unit cost of machines is too high, then GA proceeds in the solutions with low level of machine numbers, since these solutions correspond to good solutions. By alteration of the objective function parameters, we expect to force GA search different parts of the whole solution space, and observe if the performance of GA is changing with the searched space.

Currently, the objective function of the original problem is;

$F = 200 \times \text{Throughput} - 25000 \times \text{Total number of machines} - 1000 \times \text{Total number of buffers}$

Throughout the analysis, we do not modify the unit profit for goods manufactured, because this may lead to an unrealistic situation. In order to make a reasonable analysis, the parameters should be altered in significant amounts, like the multiples of current values. But increasing or decreasing the unit profit in multiples will be completely unrealistic, since it is hard to achieve such significant increments in unit profit in real-life applications. Thus we decide to alter the values of the cost parameters, which will be a more rational approach.

Since the machine numbers are dominant set of variables, to change the objective function significantly, the dominance of machines should be somehow eliminated or at least mitigated. Two separate alterations are done separately.

First we decrease the unit cost of a machine from 25000 to 1000 monetary units, to distinguish the dominance of machines in terms of cost effectiveness. Thus the unit costs are

equalized for the first analysis. Although it seems to be unrealistic for real-life situations, the unit cost of the buffer locations is altered to 25000 units while, machining cost is left in 1000 monetary units, which constitutes the second separate analysis. This is an irrational case, since the buffer cost is significantly larger than the machining cost, inappropriate for real-life; but for academic purposes in order to make a complete analysis of the factors, these specified cost parameters are used. We expect that even equalizing the unit costs, cannot eliminate the dominance of machine numbers, because of the effects on throughput level; thus a huge increment in buffer cost may force GA to search different levels of buffer locations distinctively. In the original test problem case, GA first sets the machine numbers to appropriate levels, contributing the good solutions; and then search for different levels of buffers for better solutions. In this modified version, we expect GA to give at least more importance to buffer levels in proceeding to good solutions.

To sum up, the objective function parameters of the original test problem are modified and following two schemes are obtained.

1. f = 200 × throughput – **1000** × Total number of machines – 1000 × Total buffer locations
2. f = 200 × throughput – **1000** × Total number of machines – **25000** × Total buffer locations

From now on, the first one is called as "1000" case, while the second is represented as "buffer" case. Similar to the first section, separate analyses with respect to fitness response and CPU time response are handled for each modified version. The factorial designs are used during the experimentation; but 2^3 factorial designs are used instead. Our preliminary analysis concludes that setting the population size and maximum generation number to their high values do not improve the fitness value significantly when the extensive CPU time requirement is considered, thus we decide to fix the population size and maximum generation number parameters to their low levels, and carry out the analysis with three factors; initial population type, crossover probability and mutation probability. Obviously 8 design points are constructed for each analysis, and 5 independent replications of GA are taken.

To observe the differences occurring according to cost parameter modifications, the same analysis should be handled for the original problem to compare the results. Thus considering the 3 numerical parameters a 2^3 factorial design is also made for the original problem; and the results of this case will be presented in the name of "original" in this section, and computational results of these three analyses (original plus the two modified version) will

be presented with making comparisons and stating rational explanations in the following sections.

6.1.1. Model Adequacy Checking

The independence of replications is provided by use of independent seeds of SIMAN as previously mentioned. To apply ANOVA confidently, normality and homogenous variance assumptions should be checked, but this time we have only 8 points, and fitting residuals for 8 points may result with misleading conclusions. Moreover Bartlett test is very sensitive to normality assumption, so testing homogeneity of variances under this situation is irrelevant. But it is known that, moderate departures from normality assumption are of little concern in fixed effects ANOVA; since the F test is only slightly affected (Montgomery, 1991 pp.96-102). Thus we continue with ANOVA and determination of parameter effects. Table 14 and 15 represents the GA results for original case, 1000 case and buffer case for fitness and CPU time response respectively.

Table 14.a) GA Results for Fitness Response of Original Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	799400	800280	806560	789800	791080
-1	1	1	785000	800760	796920	789920	780800
1	-1	1	801480	794760	792760	801800	791720
-1	-1	1	800080	799440	794720	792360	789040
1	1	-1	792960	792640	792800	793600	788720
-1	1	-1	774320	768160	797040	802280	743400
1	-1	-1	806360	800080	799440	785440	785840
-1	-1	-1	778240	787720	798520	810160	767560

Table 14.b) GA Results for Fitness Response of 1000 Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	1128080	1124920	1124280	1126360	1121800
-1	1	1	1126240	1128360	1127200	1123120	1122360
1	-1	1	1125920	1123360	1128520	1127800	1116840
-1	-1	1	1120200	1122360	1127280	1130920	1118840
1	1	-1	1131320	1121440	1125480	1126160	1117000
-1	1	-1	1133040	1125240	1122560	1121480	1120160
1	-1	-1	1130400	1118520	1120520	1120640	1119960
-1	-1	-1	1114640	1120520	1118680	1115320	1112240

Table 14.c) GA Results for Fitness Response of Buffer Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	1005840	994440	1011760	971520	1003200
-1	1	1	975080	980600	988760	1016240	982120
1	-1	1	980280	1012320	967480	1016240	971280
-1	-1	1	980280	993800	967760	1009520	977040
1	1	-1	977320	906040	989680	1009120	932960
-1	1	-1	1005840	1012320	1005400	926840	986920
1	-1	-1	981360	946240	927800	996200	1000480
-1	-1	-1	997280	1012320	938080	1016340	1004160

Table 15.a) GA Results for CPU Time Response of Original Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	5929,93	5822,16	5824,43	5926,42	5864,48
-1	1	1	5746,66	5297,97	5825,66	5630,84	5499,36
1	-1	1	5760,22	5840,01	5912,98	5812,63	5783,68
-1	-1	1	5656,01	5889,85	5296,02	5364,08	5534,52
1	1	-1	7141,63	6621,57	6801,78	6628	6736,08
-1	1	-1	5961,12	5982,93	6453,72	6153,29	4792,35
1	-1	-1	6417,06	6291,32	6581,89	6659,2	6600,68
-1	-1	-1	6276,26	6156,25	6063,63	6408,02	6072,87

Table 15.b) GA Results for CPU Time Response of 1000 Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	5661,8	5422,61	5587,75	5435,03	5638,8
-1	1	1	5722,59	5480,87	4903,36	5222	5434,26
1	-1	1	5445,29	5639,51	5775,05	5547,47	5477,7
-1	-1	1	5592,07	5373,23	5264,04	5122,34	5380,11
1	1	-1	6163,43	5742,61	6482,34	6625,6	6610,48
-1	1	-1	6262,81	6077,83	5818,22	5489,13	5965,93
1	-1	-1	6942,51	6612,96	6526,81	6792,33	6302,58
-1	-1	-1	5877,65	6003,7	5825,08	5593,01	5383,97

Table 15.c) GA Results for CPU Time Response of Buffer Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	5880,75	5792,62	5558,55	5614,74	5823,85
-1	1	1	5492,19	5461,88	5934,56	5655,22	5766,87
1	-1	1	5842,05	5460,34	5981,52	5795,77	5715,67
-1	-1	1	5621,41	5674,35	5714,41	5870,11	5770,21
1	1	-1	6394,53	6072,88	6239,46	6502,45	6372,38
-1	1	-1	5943,15	5928,61	6231,3	5528,61	6310,79
1	-1	-1	6423,39	6522,6	6271,64	6729,64	6361,42
-1	-1	-1	5726,68	6189,2	5296,59	6228,24	6560,55

5.2.2. Fitness Response

In this section the significant effects in terms of fitness value is analyzed separately for three distinct cases of objective function parameters.

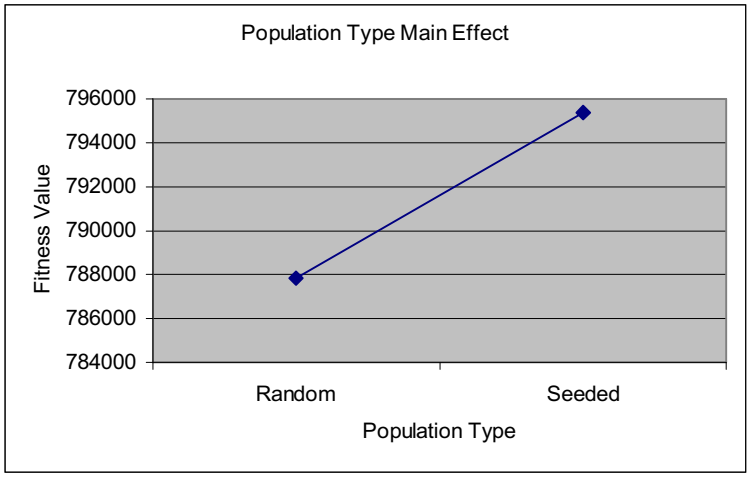
6.1.2.1. Original Case

Table 16 is the ANOVA table of the original case.

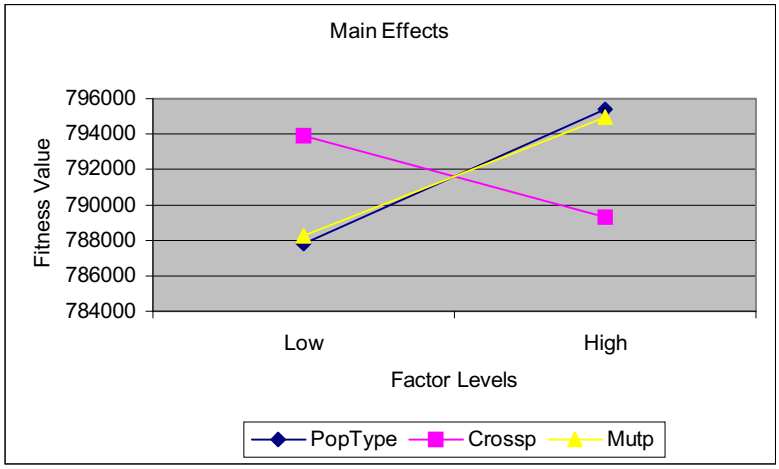
Table 16: ANOVA Results for Fitness Response for Original Case

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	1541103640	7	220157662,9	1,61	0,169
Intercept	25065159072040	1	2,50652E+13	183181,38	0,000
POPTYPE	570629160	1	570629160	4,17	0,049
CROSSP	207389160	1	207389160	1,52	0,227
MUTP	444889000	1	444889000	3,25	0,081
POPTYPE * CROSSP	113569000	1	113569000	0,83	0,369
POPTYPE * MUTP	122080360	1	122080360	0,89	0,352
CROSSP * MUTP	77841000	1	77841000	0,57	0,456
POPTYPE * CROSSP * MUTP	4705960	1	4705960	0,03	0,854
Error	4378638720	32	136832460		
Total	25071078814400	40			
Corrected Total	5919742360	39			

According to 95% precision level the only significant factor is the initial population type. Figure 14.a represents the population type effect, and Figure 14.b shows all the main effects.



a) Population Type Main Effect on Fitness of the Original Case



b) All Factors Main Effects on Fitness of the Original Case

Figure 14: Graphs of Significant Effects with respect to Fitness for Original Case

The only significant effect is the initial population type (Figure 14.b). As discussed in page 59, starting with a seeded initial population accelerates GA in finding the good solutions with appropriate machine number patterns. A random start looses time in converging to good solutions; thus GA has more time in seeded case to search for different levels of buffers for good patterns of machine numbers. The significance of population type is consistent with the original test problem analysis.

The main difference from the previous 2^5 factorial design results is the insignificance of the mutation probability. The main reason is that, the mutation operator cannot find enough time to show its property or power in finding better solutions; since the population size and maximum generation number parameters are set to their low levels. The minimum number of

solutions is processed by GA under these factor levels; thus increment in mutation rate cannot achieve a significant improvement in fitness response. But as seen from Table 16, it is statistically significant for 90% precision level, which can be considered as mutation just starts to present its ability.

The crossover probability is insignificant as in the original test problem's case, whose reasons are explained in detail in pp 59-61.

6.1.2.2. 1000 Case

Table 17 is the ANOVA table of 1000 case, where the unit machine operating cost is set to 1000 as in the case of buffer operating cost.

Table 17: ANOVA Results for Fitness Response for 1000 Case

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	314815360	7	44973622,86	2,41	0,042
Intercept	50467802220160	1	5,04678E+13	2709936,74	0,000
POPTYPE	20391840	1	20391840	1,09	0,303
CROSSP	99603360	1	99603360	5,35	0,027
MUTP	88327840	1	88327840	4,74	0,037
POPTYPE * CROSSP	29584000	1	29584000	1,59	0,217
POPTYPE * MUTP	17635840	1	17635840	0,95	0,338
CROSSP * MUTP	43597440	1	43597440	2,34	0,136
POPTYPE * CROSSP * MUTP	15675040	1	15675040	0,84	0,366
Error	595943680	32	18623240		
Total	50468712979200	40			
Corrected Total	910759040	39			

According to 95% precision level, crossover and mutation probabilities are significant factors. In order to explain such behavior, it is necessary to examine the nature of solution space under such cost parameters. Since the machine cost is equalized with buffer cost and decreased to 1000 monetary units, good solutions are supposed to have 4 – 4 – 4 – 4 as machine combination. All decision variables have the same unit cost, thus for optimization purposes it is better to increase the one having the largest positive effect on profit, in other words throughput rate.

As expected, good solutions are the ones with machine number patterns of (4 – 4 – 4 – 4), (4 – 4 – 3 – 3), (4 – 4 – 4 – 3), and (4 – 4 – 3 – 4). Different from the original case, good solutions have only 4 distinct patterns. Number of machines in first two stations is at

maximum level, because these stations correspond to the bottlenecks of the system. For 40 independent GA applications, the best fitness value achieved shows the following behavior;

Solution	# Times Encountered
4 – 4 – 4 – 4	24 times
4 – 4 – 4 – 3	6 times
4 – 4 – 3 – 4	6 times
4 – 4 – 3 – 3	4 times

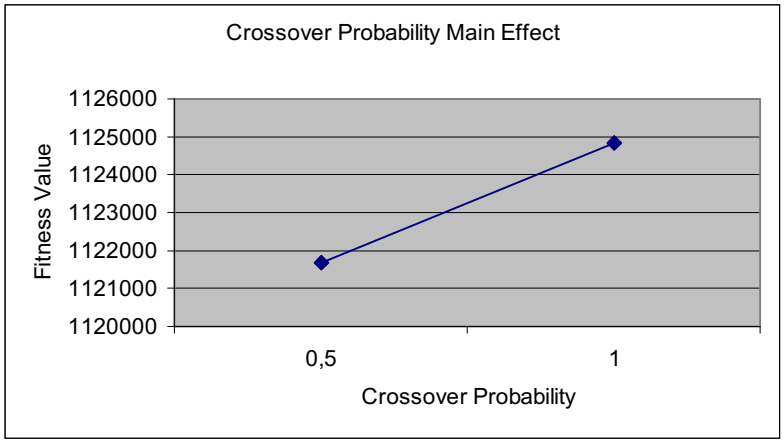
The solutions with these machine number patterns are dominating in terms of fitness value, thus GA converges to these solutions and proceeds among different buffer levels as in the case of original problem; but this time dominance of good solutions is more definite.

Another interpretation is that, the solution space in 1000 case has different properties compared with the original case. For the original problem solutions with same machine patterns but different buffer levels have close fitness values (pp. 55-56). For 1000 case, any alteration of a buffer variable has more significant impact on fitness value, due to the larger increment on throughput level. The main reason is that, the workstations now have more machines and can process more parts, thus any small alteration on buffer leads to distinctive changes on the profit. The following scheme will clarify the situation.

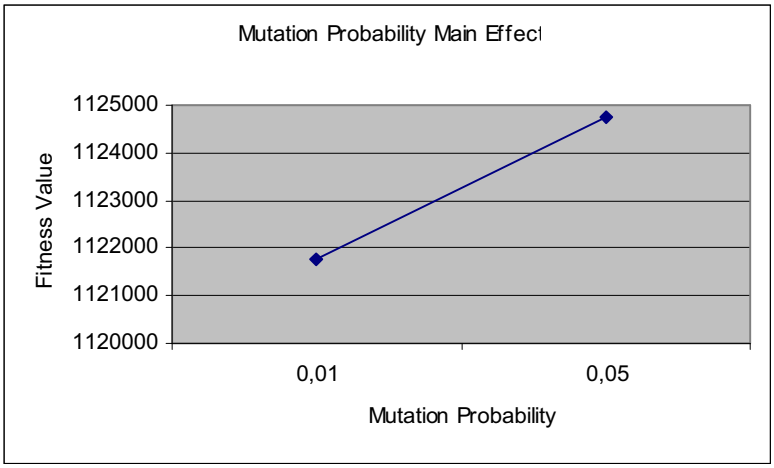
Solution	Fitness Value (monetary units)
4 – 4 – 4 – 4 – 10 – 5 –1	1126240
4 – 4 – 4 – 4 – 8 – 4 – 2	1133040
4 – 4 – 4 – 4 – 11 – 3 – 11	1114640
4 – 4 – 4 – 4 – 5 – 2 – 3	1122960
4 – 4 – 4 – 4 – 8 – 8 – 7	1121280
4 – 4 – 4 – 4 – 7 – 9 – 2	1111680
4 – 4 – 4 – 4 – 6 – 7 – 1	1114600

There is a nearly 22,000 units difference between solutions, whereas the same difference is only 5,000 for the original case.

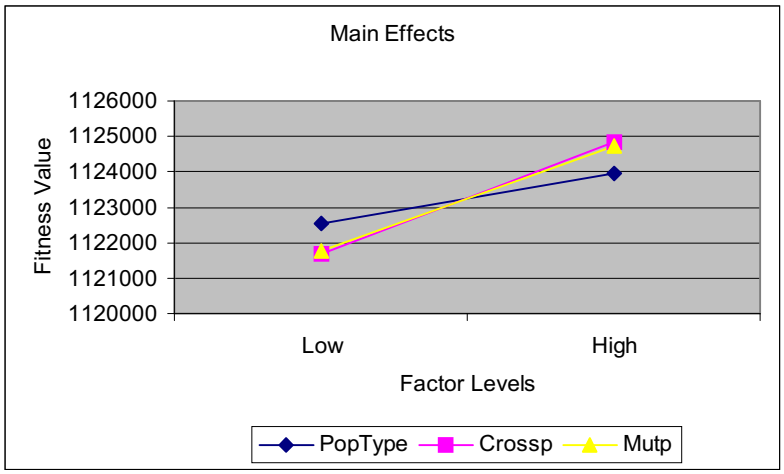
After this preliminary information, the significance of effects can be analyzed thoroughly. Figure 15.a represents the crossover probability effect on fitness, while Figure 15.b belongs to mutation probability factor. Figure 15.c presents all the main effects at a time.



a) Crossover Probability Main Effect on Fitness of the 1000 Case



b) Mutation Probability Main Effect on Fitness of the 1000 Case



c) All Factors Main Effects on Fitness of the 1000 Case

Figure 15: Graphs of Significant Effects with respect to Fitness for 1000 Case

From Figure 15.c it can be observed that the initial population type is insignificant considering the best fitness found. There are only four patterns of machine numbers

corresponding to good solutions, and they are excessively dominating over other solutions, thus regardless of the starting population, GA converges to these good solutions rapidly. Thus a random or seeded start does not make any sense for GA performance.

Different from the previous analysis crossover probability has a positive effect on fitness, as in Figure 15.a. As discussed in pages 60-61, crossover fails to create diverse solutions for the original problem case. The same reasoning is still valid in this situation, where crossover within the machine number variables results with same individuals. The difference is the significant alteration of fitness for same number of machines and different buffer levels. In original problem, crossover produces individuals with same machining pattern with different buffer levels, but as the fitness values of these solutions are very close to each other, crossover cannot have a significant impact on performance. But this time these types of solutions have enough fitness differences to result with significance of crossover probability.

Mutation probability is effective on fitness, where high values of mutation increase the performance of GA, which can be observed from Figure 15.b. The explanations held for the original problem are also valid for 1000 case.

6.1.2.3. Buffer Case

Table 18 shows the ANOVA results of buffer case, which has 1000 as unit machine operating cost, and 25000 as buffer operating cost.

Table 18: ANOVA Results for Fitness Response for Buffer Case

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	4800322950	7	685760421,4	0,86	0,549
Intercept	38766184015690	1	3,87662E+13	48558,51	0,000
POPTYPE	766850490	1	766850490	0,96	0,334
CROSSP	5083690	1	5083690	0,01	0,937
MUTP	1355594490	1	1355594490	1,70	0,202
POPTYPE * CROSSP	8704890	1	8704890	0,01	0,917
POPTYPE * MUTP	2271953290	1	2271953290	2,85	0,101
CROSSP * MUTP	368327610	1	368327610	0,46	0,502
POPTYPE * CROSSP * MUTP	23808490	1	23808490	0,03	0,864
Error	25546868160	32	798339630		
Total	38796531206800	40			
Corrected Total	30347191110	39			

According to 95% precision level, none of the factors become significant. The nature of solution space is even more drastic for the buffer case when compared to 1000. The cost of machine is excessively smaller than the buffer cost, and machines have larger positive effects on throughput, thus it is predictable that the number of machines in each workstation will be in its high level, or closer. The good solutions have the same patterns in terms of machine numbers with the 1000 case. There are only 4 different patterns observed, where 4 – 4 – 4 – 4 is the most frequently seen (20 of 40 independent GA runs). Different from 1000 case, the buffer locations have also specific patterns in good solutions. As the buffer cost is extremely high, for optimization purposes they are set at low levels. Although each buffer position can range from 1 to 16, in good solutions buffer values do not exceed 3, since in the case of any increment over this value, the increase in cost cannot be compensated by extra throughput processed. Generally good solutions have buffer patterns of (1 – 1 – 1), (1 – 1 – 2), (2 – 1 – 1), (3 – 2 – 1) and (2 – 1 – 2). Thus for buffer case, number of good solutions is less than the other cases, since there is no buffer diversification for the same machine numbers. And it is obvious that, the good solutions are extremely dominating among the solution space, thus rapid convergence proceeds in GA.

According to ANOVA analysis, as observed from Figure 16, all factors are insignificant in terms of fitness response.

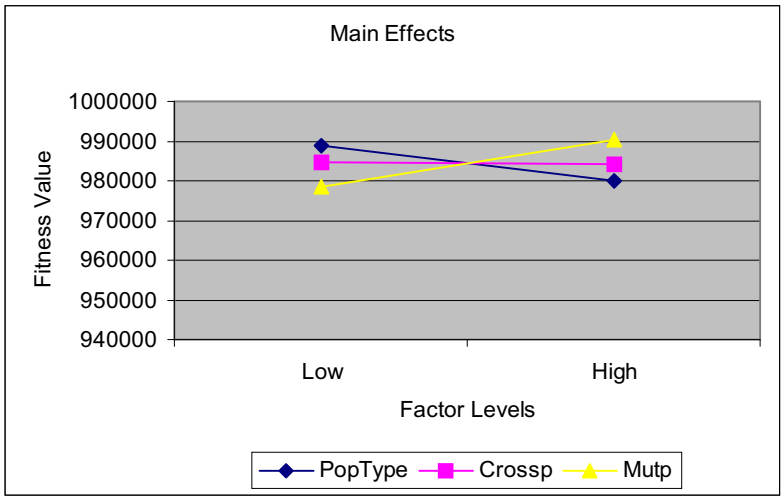


Figure 16: All Factors' Main Effects on Fitness of Original Case

The reason is that, the over dominance of good solutions force GA to a rapid convergence. And less number of good solutions obscures the diversification among different combination of GA parameters, thus regardless of the parameter combination GA finds these

solutions rapidly. As there are only few good solutions, and their fitness values are close to each other, while extremely higher than the other solutions, GA parameter selection cannot make a significant impact on performance. Wherever or however GA starts, it proceeds to the same points at the end.

6.1.3. CPU Time Response

The original case is analyzed separately, while 1000 and buffer cases are examined under one title, because of the similarities they show throughout the analysis.

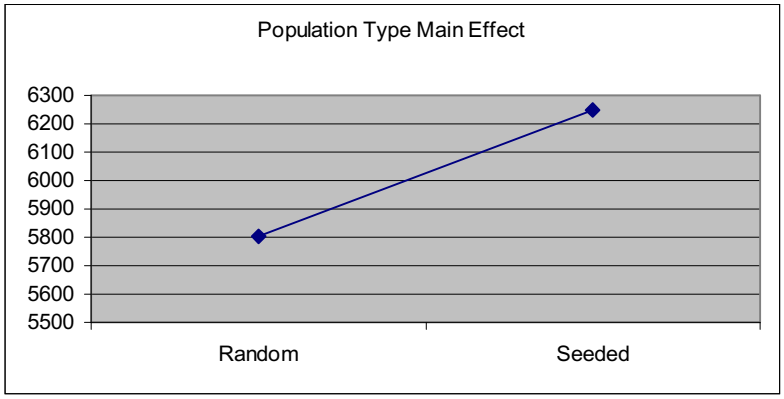
6.1.3.1. Original Case

Table 19 presents the ANOVA results for original case in terms of CPU time response.

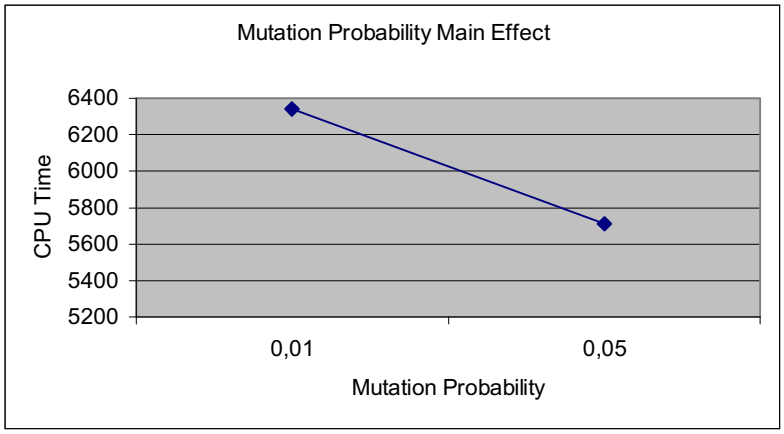
Table 19: ANOVA Results for CPU Time Response for Original Case

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	6698726	7	956961	12,82	0,000
Intercept	1452236606	1	1452236606	19458,93	0,000
POPTYPE	1977910	1	1977910	26,50	0,000
CROSSP	1732	1	1732	0,02	0,880
MUTP	3957505	1	3957505	53,03	0,000
POPTYPE * CROSSP	226566	1	226566	3,04	0,091
POPTYPE * MUTP	292889	1	292889	3,92	0,056
CROSSP * MUTP	14924	1	14924	0,20	0,658
POPTYPE * CROSSP * MUTP	227201	1	227201	3,04	0,091
Error	2388187	32	74631		
Total	1461323519	40			
Corrected Total	9086913	39			

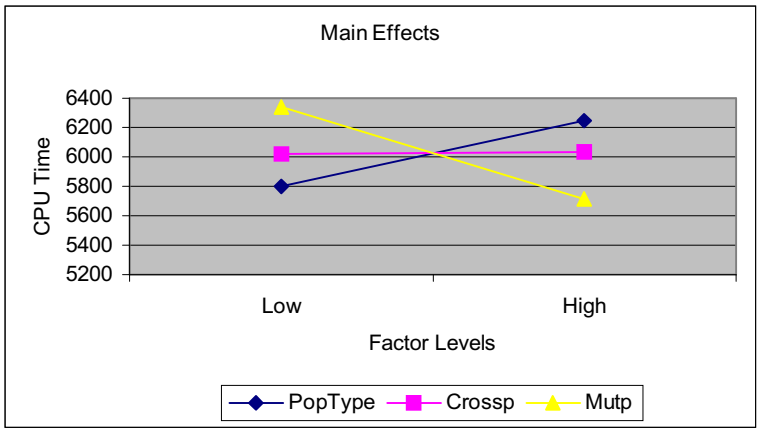
According to ANOVA table, the significant factors for 95% precision level are the initial population type and mutation probability. As discussed in Section 5.1.3, CPU time merely depends on the simulation time, and simulation time is associated with the load of the system under consideration. Figure 17.a represents the initial population type effect, whereas Figure 17.b is the mutation probability effect, and Figure 17.c shows the main effects altogether.



a) Population Type Main Effect on CPU Time of Original Case



b) Mutation Probability Main Effect on CPU Time of the Original Case



c) All Factors Main Effects on CPU Time of the Original Case

Figure 17: Graphs of Significant Effects with respect to CPU Time for Original Case

Initial population type is insignificant for 2^5 factorial design analysis, but significant for this case. A seeded start converges more rapidly to good solutions compared with a random start. Good solutions are loaded systems, thus CPU time is more in the seeded population case. But this is valid only for first few generations (at most 10). After some initial

generations, GA with different starting conditions contains same good solutions with similar simulation time. For the first analysis (Chapter 5), the maximum generation number is twice the current value; hence the simulation time difference between the first few generations cannot make significant effect on CPU time. But this time as the generation number is low, and GA proceeds only 50 generations, the excess of loaded systems in first generations of a seeded start creates a significant impact on CPU time. Seeded initial population increases the CPU time covered as observed from Figure 17.a.

The significance of mutation probability and insignificance of crossover probability are explained in detail in Section 5.1.3 in pages 69-71. Increasing the mutation probability, decreases the CPU time as seen from Figure 17.c, since less-loaded system simulation occurs more frequently as discussed before, while crossover probability does not effect the GA performance in terms of CPU time significantly.

6.1.3.2. 1000 and Buffer Cases

Table 20.a and 20.b represents the ANOVA results for 1000 case and buffer case respectively.

Table 20.a: ANOVA Results for CPU Time Response for 1000 Case

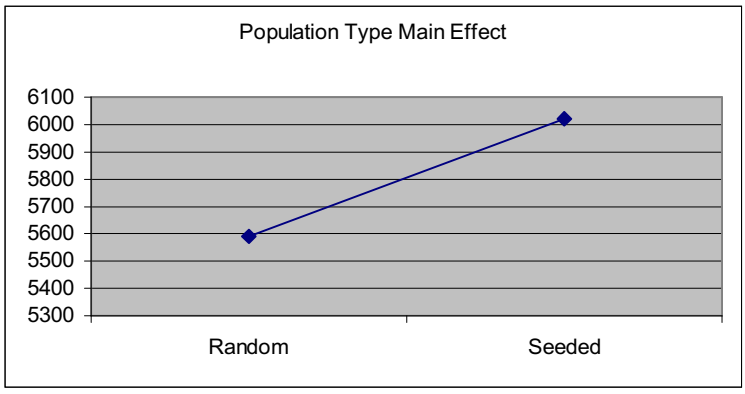
Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	7554365	7	1079195	17,19	0,000
Intercept	1348209640	1	1348209640	21474,79	0,000
POPTYPE	1866439	1	1866439	29,73	0,000
CROSSP	13321	1	13321	0,21	0,648
MUTP	4881188	1	4881188	77,75	0,000
POPTYPE * CROSSP	176035	1	176035	2,80	0,104
POPTYPE * MUTP	477025	1	477025	7,60	0,010
CROSSP * MUTP	6617	1	6617	0,11	0,748
POPTYPE * CROSSP * MUTP	133740	1	133740	2,13	0,154
Error	2008993	32	62781		
Total	1357772998	40			
Corrected Total	9563358	39			

Table 20.b: ANOVA Results for CPU Time Response for Buffer Case

Source	Type III Sum of Squares	df	Mean Square	F	Significance Level
Corrected Model	3070154	7	438593	7,01	0,000
Intercept	1419209747	1	1419209747	22692,97	0,000
POPTYPE	495356	1	495356	7,92	0,008
CROSSP	39088	1	39088	0,63	0,435
MUTP	2212310	1	2212310	35,37	0,000
POPTYPE * CROSSP	5136	1	5136	0,08	0,776
POPTYPE * MUTP	296188	1	296188	4,74	0,037
CROSSP * MUTP	2579	1	2579	0,04	0,840
POPTYPE * CROSSP * MUTP	19498	1	19498	0,31	0,580
Error	2001268	32	62540		
Total	1424281169	40			
Corrected Total	5071422	39			

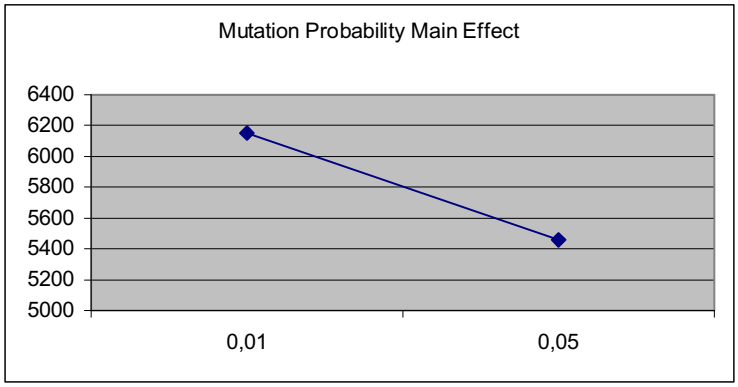
For 95 % precision level the significant factors are appeared to be; initial population type, mutation probability and the interaction between them. The good solutions for both cases have the same four patterns in terms of machine numbers, that corresponds to more loaded systems compared with the original case. Good solutions have (4 – 4 – 4 – 4), (4 – 4 – 3 – 3), (4 – 4 – 4 – 3), and (4 – 4 – 3 – 4) as machine number patterns corresponding to loaded solutions, since they have high level of machining pattern.

Figure 18.a presents the initial population type effect on CPU time of 1000 case, whereas 19.a is the same graph for buffer case. Figure 18.b and 19.b; Figure 18.c and 19.c, and Figure 18.d and 19.d represents the graphs of mutation probability effect, all main factor effects, and the interaction effect of population type and mutation probability for 1000 case and buffer case respectively.

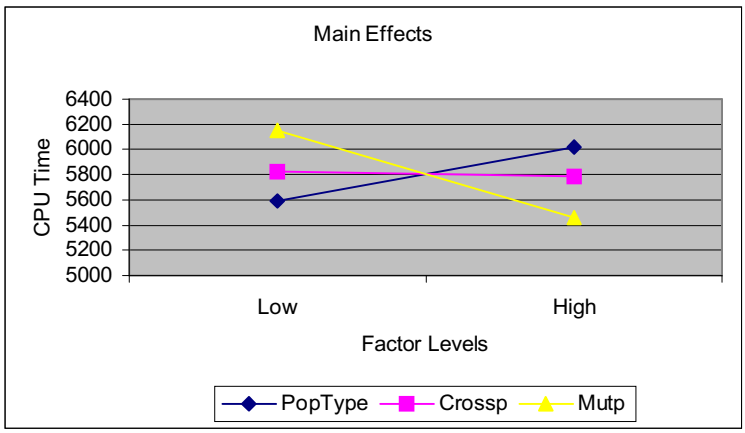


a) Population Type Main Effect on CPU Time of Original Case

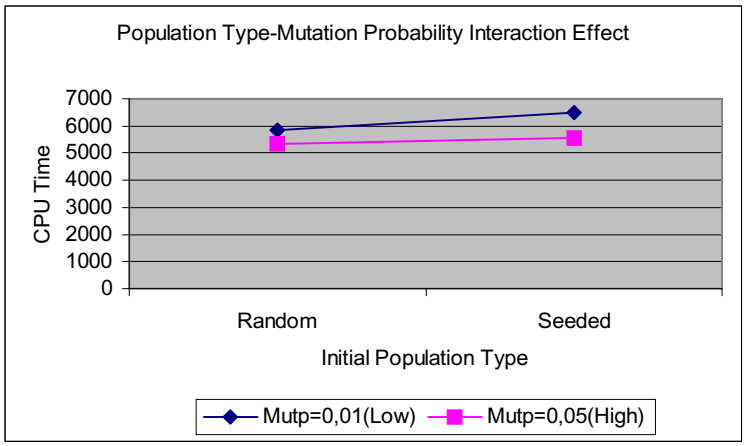
Figure 18: Graphs of Significant Effects with respect to CPU Time for 1000 Case



b) Mutation Probability Main Effect on CPU Time of the 1000 Case

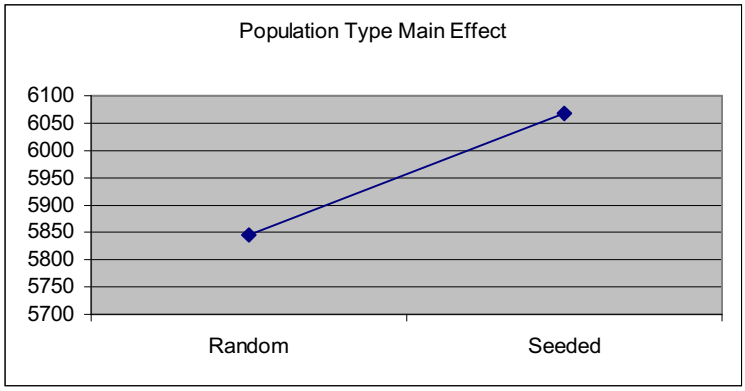


c) All Factors Main Effects on CPU Time of the 1000 Case

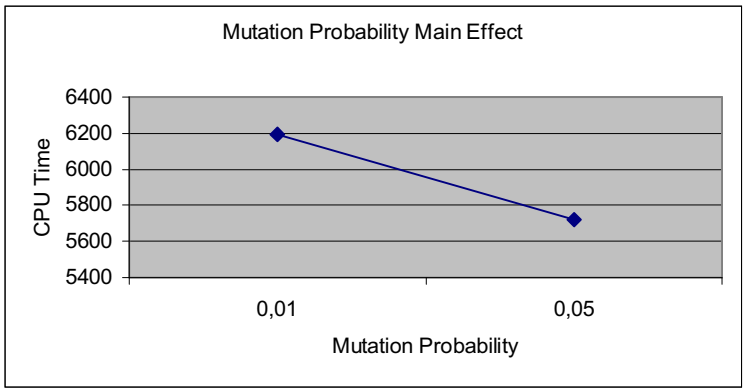


d) Population Type-Mutation Probability Interaction Effect on CPU of 1000 Case

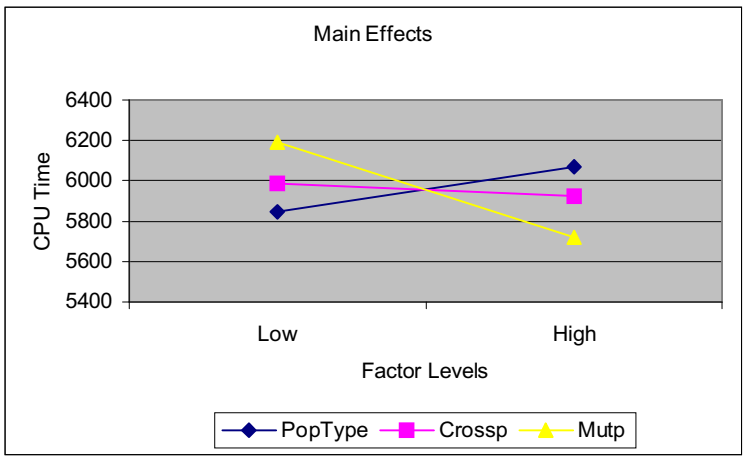
Figure 18: Graphs of Significant Effects with respect to CPU Time for 1000 Case (cont'd)



a) Population Type Main Effect on CPU Time of 1000Case

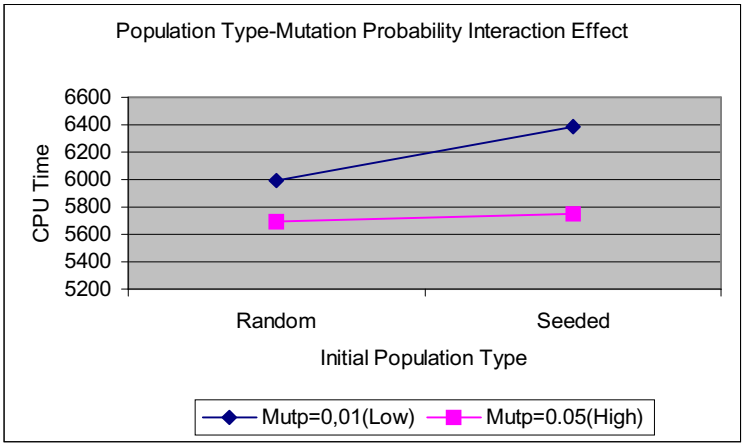


b) Mutation Probability Main Effect on CPU Time of the Buffer Case



c) All Factors Main Effects on CPU Time of the Buffer Case

Figure 19: Graphs of Significant Effects with respect to CPU Time for Buffer Case



d) Population Type-Mutation Probability Interaction Effect on CPU of 1000 Case

Figure 19: Graphs of Significant Effects with respect to CPU Time for Buffer Case (cont'd)

For both cases the initial population type is effective on CPU time, seeded start increases the duration of GA run. The reasoning stated for the original case previously is valid for these two situations. Seeded initial solution contains more loaded systems than random start at least in first few generations, and only 50 generations proceed thus the difference affects the CPU time significantly. Seeded started GA's requires more CPU time.

Crossover probability is insignificant for both 1000 case (Figure 18.c) and buffer case (Figure 19.c). Since good solutions have only some specific patterns in machine numbers, and over-dominant among other solutions, selection procedure always selects these solutions as potential parents. And crossover exchanges genetic material among them, creating individuals having same machining patterns but different levels of buffers (same buffer levels are also frequently obtained as discussed previously). These individuals correspond to solutions with same amount of CPU time, as solutions with different buffer levels and same machine combinations correspond to same loaded systems having close simulation time. Thus crossover cannot achieve diverse solutions with different simulation times, and ineffective on CPU time. The following examples will clarify the situation:

Solution		CPU Time
4 – 4 – 4 – 4 – 10 – 5 – 1	→	0.27 minutes
4 – 4 – 4 – 4 – 8 – 4 – 2	→	0.27 minutes
4 – 4 – 4 – 4 – 11 – 3 – 11	→	0.27 minutes
4 – 4 – 4 – 4 – 5 – 2 – 3	→	0.27 minutes
4 – 4 – 4 – 4 – 8 – 8 – 7	→	0.28 minutes

4 – 4 – 4 – 4 – 7 – 9 – 2 ⟶ 0.28 minutes

4 – 4 – 4 – 4 – 6 – 7 – 1 ⟶ 0.27 minutes

These solutions are good solutions for 1000 case, and although they have diverse fitness values as stated previously, they correspond to systems with same workload, thus simulation time of these systems are close to each other. Only two of the listed solutions have 0.01 minutes more simulation time than the others having equal simulation duration. Good solutions have some specific set of machine numbers, and specific set of buffer locations, and simulation time of these systems are close to each other for the buffer case.

Mutation probability is significant in terms of CPU time response for both 1000 and buffer case. It is a predicted result, since the good solutions have 4 – 4 – 4 – 4), (4 – 4 – 3 – 3), (4 – 4 – 4 – 3), and (4 – 4 – 3 – 4) as machine combinations corresponding to;

Solution		Binary Representation
4 – 4 – 4 – 4	⟶	11111111
4 – 4 – 3 – 4	⟶	11111011
4 – 4 – 4 – 3	⟶	11111110
4 – 4 – 3 – 3	⟶	11111010

Supposing the buffer levels do not have any significant impact on simulation time, mutation operator affects the CPU time by altering the machine numbers. In all of the strings presented above, number of 1's are excessively dominant over 0's, thus mutation usually generates solutions with fewer number of machines corresponding to less-loaded systems, having less simulation time. For instance, for the first string in all trials, mutation generates a less-loaded system, because of absence of any 0's in the string. For the fourth case, on 6 of 8 trials mutation results with a decrement in machine numbers. Hence, mutation probability has a negative effect on CPU time, an increment in mutation rate decreases the CPU time.

Initial population type-mutation probability interaction effect is significant for both 1000 case and buffer case (Figure 18.d and 19.d). For low level of mutation probabilities, the difference in CPU time between a seeded start and a random start is more observable. When the mutation probability is increased, the discrepancy between the GA with different starting conditions will disappear in some extent. GA with a seeded start requires more CPU time, because in few starting generations, the populations have more loaded systems (good solutions) compared with a random start. But increase in mutation rate, increases the

deviations from good solutions, thus for both random and seeded cases, good solutions are mutated and less-loaded solutions are achieved more frequently. This compensates the starting effect of seeded population on CPU time. Assuming that on the average same number of mutations are carried for both GA's, similar number of alterations is made in some extent. The good solutions creating the CPU time discrepancy in few starting generations are more frequently mutated, thus their negative effect on CPU time lessens. Hence, it is better to set the mutation probability to high level, while population type to random population for minimization of CPU time, which is consistent with main factor analyses.

6.1.4. Conclusion

The cost parameters of the objective function are altered in order to make GA search different parts of the solution space. Two modifications are made; 1000 case, where the unit machine operating cost is decreased to 1000 monetary units, and buffer case, where buffer operating cost is set to 25000 monetary units, while machine cost remains at 1000 level. As increments on the population size, and maximum generation number factors do not have significant effect on fitness value, but increases the CPU time substantially, these factors are set to their low levels, 20 and 50 respectively.

For 1000 case, the crossover and mutation probabilities appear to have significant effect on fitness value, where their high levels maximize the fitness. All factors are insignificant for buffer case on fitness response. When the same analysis is repeated for the original problem with 3 factors, only initial population type found significant, whose high level (seeded population) is useful for fitness maximization.

Considering the fitness response, GA parameters present different behaviors under different cost parameters. The searched portion of solution space even affects the impact of the numerical parameters; thus GA performance and selection of parameters depend on the nature of solution space.

Hopefully, the situation is more fortunate for CPU time response. In all three analyses, mutation probability has a negative effect on CPU time; high levels of mutation probability reduce the CPU time. The initial population type presents a significant behavior on CPU time, because of the less number of generations used. Setting the initial population type to random population decreases the CPU time for all three cases. Table 21 summarizes the results for all 3 cases for both response types.

Table 21: Results of Factorial Design Analysis for Modified Objective Function Parameters

	Pop Type	Cross P	Mut P
	Fitness		
Original	High	-	-
1000	-	High	High
Buffer	-	-	-
General	High	High	High
	CPU Time		
Original	Low	-	High
1000	Low	-	High
Buffer	Low	-	High
General	Low	-	High

According to Table 21, although different behaviors of parameters are experienced for fitness response, in general setting all the factors to high levels will increase the fitness value as in the case of the first analysis of Section 5.1. For CPU time reduction, it is better to select random initial population with high level of mutation probability.

The mutation probability levels are consistent for each response, thus it is obvious that high level of mutation rate is better for both objectives. Since crossover probability is ineffective for CPU time, but has a positive effect on fitness, setting it to high level is appropriate for both objectives. The only contradiction occurs in initial population type, thus further analysis is required for the selection of its adequate level. Table 22 presents the results of the analysis.

Table 22: Percentage Change in Responses According to Change in Population Type

	Original		1000		Buffer	
Pop Type	**Fitness**	**CPU Time**	**Fitness**	**CPU Time**	**Fitness**	**CPU Time**
Random	787822	5803,071	1122538	5589,61	988835	5845,247
Seeded	795376	6274,808	1123966	6021,633	980078	6067,813
% Change	0,9	7,7	0,1	7,7	-0,8	3,8

Seeded initial population increases the best fitness found except in buffer case, but also CPU time positively affected, GA requires more computation time. Although the percentages are not significantly different as in the case of the original problem (Section 5,1), considering

the both objectives with equal weights, it is better to start with a random population. But if only slight importance is given to fitness response over CPU time, seeded population will be the adequate choice.

Thus, to sum up, although the results are not so much in parallel in some extent, our further analysis with different cost parameters reveal that high levels of crossover probability and mutation probability; and low level of initial population type (random population) is appropriate for better performance of GA in terms of fitness and CPU time responses. These results are consistent with the original problem's analysis except the initial population type; but as stated above seeded populations can also be selected since the random population does not have significant superiority over seeded initial population.

6.2. Constrained Version Of Test Problem

Another modification we made on the test problem is adding a linear constraint to the problem. The constraint will make some combinations of decision variables infeasible, thus shrink the solution space. If the feasible space is too small, then any combination of GA parameters can manage to find optimal or near-optimal solutions, hence the analysis of effects become meaningless.

To create a different experimental condition; the linear constraint is added to the test problem in such a way that it forces GA to search different parts of the solution space compared with the other two cases (original case and the cost parameter modified version). By this way, we think that we can analyze the dependence of GA parameters' effectiveness on the searched space.

The machine numbers are the dominating set of variables, thus we decide to add constraint on the total number of machines. In fact, in the original problem (Law and McComas, 2000) authors impose a linear constraint on total number of machines. We use the same constraint in our analysis, which is;

$$M_1 + M_2 + M_3 + M_4 \leq 10$$

which means the total number of machines at the workstations cannot exceed 10.

During the GA implementation, the strings created corresponding to infeasible solutions are discarded, and new strings are generated instead. To compare the results with the original problem and the other modified versions, same analysis of 2^3 factorial design is used.

Throughout the section, the constrained version of the test problem will be defined as "constrained" case

6.2.1. Model Adequacy Checking

Again the independence is ensured by the use of independent random number seeds. According to the reasons stated for other modified versions (pp.80), we continue with ANOVA without any other diagnostic checking. Table 23.a shows the GA results for fitness response, whereas Table 23.b presents the results for CPU time.

Table 23.a: GA Results for Fitness Response of Constrained Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	592080	597240	599680	591160	593120
-1	1	1	581440	595360	597520	595440	594440
1	-1	1	593280	589280	595240	590360	596320
-1	-1	1	594080	596560	601400	595520	594640
1	1	-1	594640	598040	603680	589120	598520
-1	1	-1	596160	596640	604800	595560	596840
1	-1	-1	594960	589800	603680	590600	570760
-1	-1	-1	594680	593280	599600	585320	585800

Table 23.b: GA Results for Fitness Response of Constrained Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	4298,31	4454,53	4433,92	4244,26	3884,19
-1	1	1	4142,9	4379,8	4239,86	4169,13	4248,79
1	-1	1	4316,62	4194,25	4247,8	4404,33	4643,58
-1	-1	1	4185,78	4237,64	4608,26	4309,32	4252,18
1	1	-1	4890,49	4898,7	4763,45	5215,53	5092,04
-1	1	-1	4797,52	4603,59	4757,63	4296,32	4801,63
1	-1	-1	5119,26	4678,59	4898,01	4787,65	4574,72
-1	-1	-1	4668,31	4506,72	4608,26	4521,48	4567,84

The results of the analysis with the rational explanations and comparisons with the original test problem results will be presented in the following sections..

6.2.2. Fitness Response

Table 24 represents the results of ANOVA for fitness response.

Table 24:ANOVA Results for Fitness Response for Constrained Case

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	172117017	7	24588145	1,05	0,416
Intercept	14154132152900	1	14154132152900	605802,73	0,000
POPTYPE	486644	1	486644	0,02	0,886
CROSSP	20146964	1	20146964	0,86	0,360
MUTP	17229188	1	17229188	0,74	0,397
POPTYPE * CROSSP	109412	1	109412	0,00	0,946
POPTYPE * MUTP	11393428	1	11393428	0,49	0,490
CROSSP * MUTP	55535636	1	55535636	2,38	0,133
POPTYPE * CROSSP * MUTP	67215748	1	67215748	2,88	0,100
Error	747656307	32	23364260		
Total	14155051926224	40			
Corrected Total	919773324	39			

According to 95% precision level none of the factors are significant considering the effects on fitness value. Before investigating the reasons for such a behavior, it is better to make some preliminary analysis about the results of GA applications.

Throughout the constrained case study, 40 independent replications of GA are taken, and all of these result with a best solution having machine combinations of 2 – 4 – 2 – 2 and 3 – 3 – 2 - 2. The former one appears 34 times, and the latter result is achieved at 6 trials. We impose a limit of 10 on the total number of machines; since the first two stations are bottleneck stations, we expect to have more machines in these two stations. Thus a machine combination of 3 –3 – 2 – 2 is an expected pattern for good solutions. Also the second station has more processing time than the first, according to this number of machines in station 2 should be larger than the others. Depending on this, machine pattern of 2 – 4 – 2 – 2 is another predictable machine combination of good solutions.

Under the dominance of machine numbers on the objective value of the system, good solutions have two different patterns of machine combinations as stated above. Like all other cases, the discrimination among the solutions with these machine patterns is achieved by the different levels of buffer combinations. But this time the fitness value of such solutions is closer to each other as presented with the following examples.

Solution	Fitness Value (monetary units)
2 – 4 – 2 – 2 – 12 – 5 – 2	⟶ 596160
2 – 4 – 2 – 2 – 8 – 4 – 4	⟶ 594080
2 – 4 – 2 – 2 – 15 – 5 – 2	⟶ 594680

2 – 4 – 2 – 2 – 15 – 3 – 7 ⟶ 592080

2 – 4 – 2 – 2 – 11 – 1 – 4 ⟶ 594640

2 – 4 – 2 – 2 – 12 – 8 – 3 ⟶ 593280

Since there are only two good patterns of machine numbers, number of good solutions is less in some extent when compared with the other cases of the problem. In fact, the feasible solution space is smaller compared with the other cases.

Like the other cases, the good solutions are dominant among all the feasible solutions in terms of fitness value. Thus we expect the rapid convergence of GA to these solutions.

After these preliminary discussions, we can analyze the results of ANOVA. Figure 20 represents the effects of all factors on the fitness value.

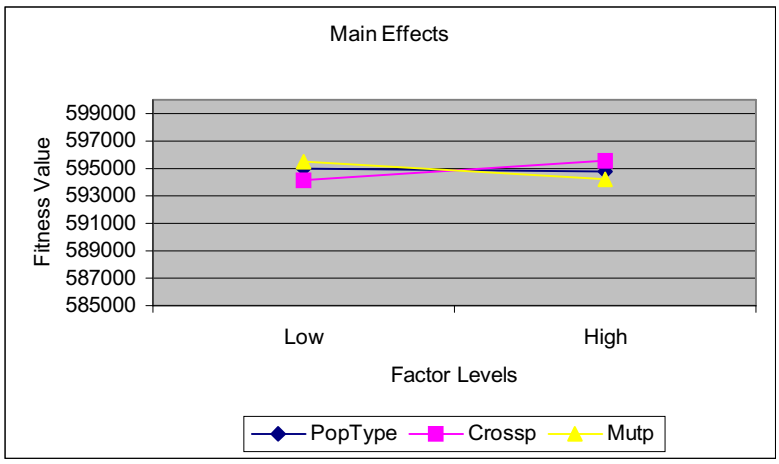


Figure 20: All Factors Main Effects on Fitness of Constrained Case

As observed from the figure, all the factors are insignificant. Because the good solutions are highly dominant over other feasible solutions and they have close fitness values among each other, regardless of the type of GA processed, algorithm converges to one of these good solutions.

The initial population type is insignificant, although seeded populations may accelerate GA in converging to good solutions. Since the population size, and maximum generation number are at their low levels, even the effect of a seeded start is supposed to be more significant. But, because the good solutions are dominant and there is only few good solutions, GA converges to these solutions rapidly regardless of the starting conditions.

Crossover and mutation operators are also insignificant considering the fitness response. The main contribution of these two operators is generating different solutions while maintaining the good pattern of fit individuals. Mutation is responsible for adding diversity to

populations, while crossover is in charge of maintenance of good properties through the generations. The insignificance of these factors can be explained in two manners.

From the previous analysis we know that GA converges to solutions with machine patterns of 2 – 4 – 2 – 2 and 3 – 3 – 2 – 2 rapidly, because of dominance of the good solutions. There are only few good solutions present in the feasible solution space, which are dominant over the others; thus regardless of the levels of the factors GA can proceed to one of these good solutions.

Another interpretation is that, since the fitness values of solutions with these machine patterns and different buffer combinations are very close to each other, as discussed in the preliminary analysis; although mutation and crossover achieves searching different solutions, distinct levels of their probabilities cannot create significant difference on fitness value.

6.2.3. CPU Time Response

Table 25 present the ANOVA results for CPU time response.

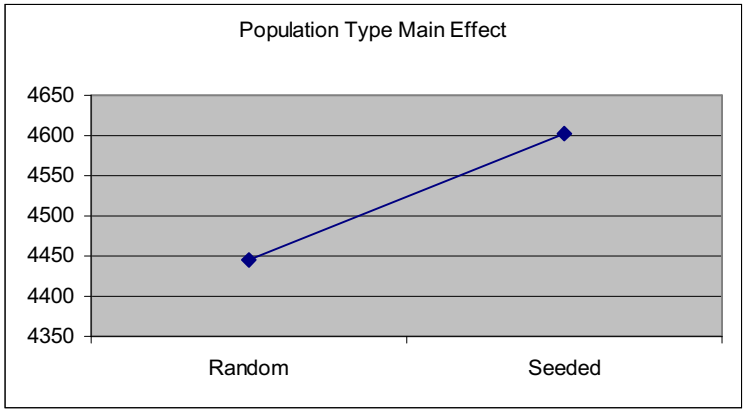
Table 25:ANOVA Results for CPU Time Response for Constrained Case

Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	2609381	7	372768,663	12,07	0,000
Intercept	818510950	1	818510950,2	26494,66	0,000
POPTYPE	246062	1	246061,5763	7,96	0,008
CROSSP	1988	1	1987,959003	0,06	0,801
MUTP	2094110	1	2094110,306	67,78	0,000
POPTYPE * CROSSP	2877	1	2876,924823	0,09	0,762
POPTYPE * MUTP	148963	1	148963,2455	4,82	0,035
CROSSP * MUTP	109216	1	109216,0854	3,54	0,069
POPTYPE * CROSSP * MUTP	6165	1	6164,544122	0,20	0,658
Error	988590	32	30893,43279		
Total	822108921	40			
Corrected Total	3597970	39			

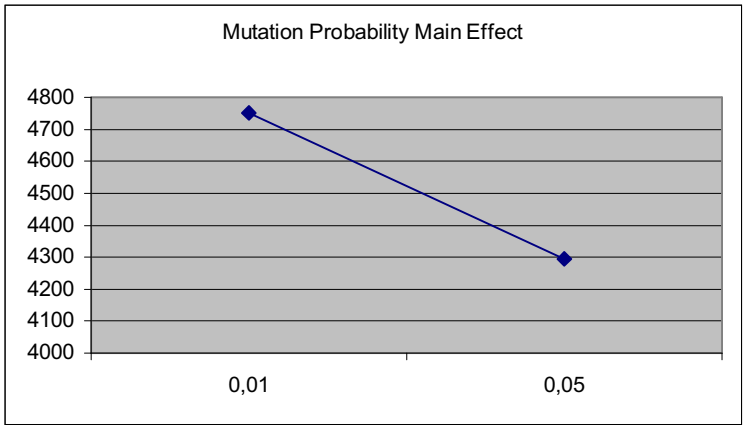
For 95% precision level, the significant factors are population type, mutation probability and the interaction between them. Good solutions of constrained case, which have machine number patterns of 2 – 4 – 2 – 2 and 3 – 3 – 2 – 2 contribute to loaded systems of their feasible solution space, since no other values of feasible machine pattern can have more throughput level, because the maximum efficient use of machines in these systems. Moreover the simulation time of good solutions, with these machine patterns but different buffer levels are close. The following examples will clarify the situation.

Solution	CPU Time
2 – 4 – 2 – 2 – 12 – 5 – 2	0.18 minutes
2 – 4 – 2 – 2 – 8 – 4 – 4	0.18 minutes
2 – 4 – 2 – 2 – 15 – 5 – 2	0.18 minutes
2 – 4 – 2 – 2 – 15 – 3 – 7	0.18 minutes
2 – 4 – 2 – 2 – 11 – 1 – 4	0.18 minutes
2 – 4 – 2 – 2 – 12 – 8 – 3	0.18 minutes

All of the above solutions have same simulation time. Figure 21.a represents the initial population type effect on CPU time, whereas Figure 21.b shows the mutation probability effect. In Figure 21.c the alteration of CPU time with respect to all factors can be observed, finally Figure 21.d resembles the significant interaction between the population type and mutation probability.

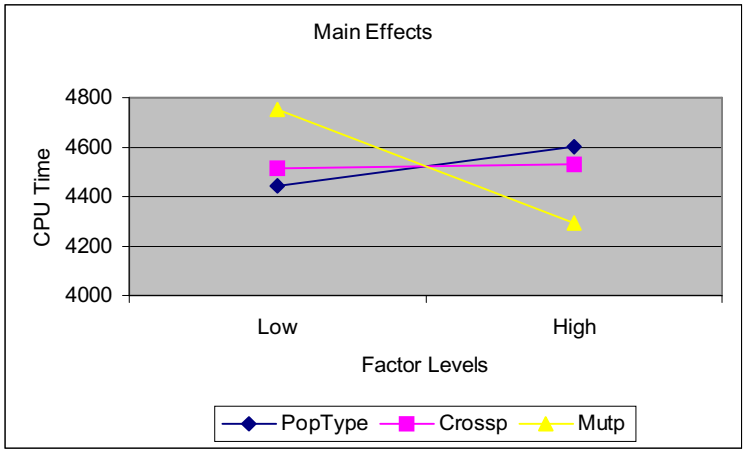


a) Population Type Main Effect on Fitness of the Constrained Case

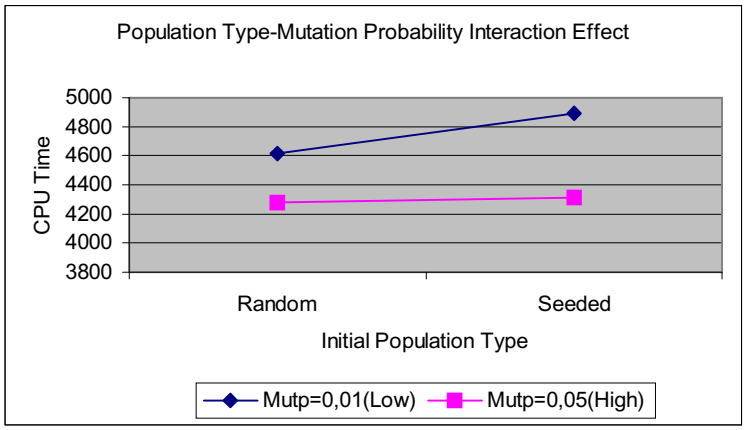


b) Mutation Probability Main Effect on Fitness of the Constrained Case

Figure 21: Graphs of Significant Effects with respect to Fitness for Constrained Case



c) All Factors Main Effects on Fitness of Constrained Case



d) Population Type-Mutation Probability Interaction Effect on Fitness of Constrained Case

Figure 21: Graphs of Significant Effects with respect to Fitness for Constrained Case (cont'd)

From Figure 21.a, it is observed that the initial population type has a positive effect on CPU time, thus seeded starts increase the CPU time. This result is consistent throughout all the cases we have analyzed, and the reasoning is the same. (To hinder the repetitions, reader can look at pp 90-91 for discussion).

Crossover is insignificant for CPU time response, because crossover can generate solutions with similar patters having same simulation time, thus to cross more frequently does not have any impact on the elapsed time, as discussed in Section 5.3 previously (Figure 21.c).

Mutation probability has a negative effect on CPU time; high mutation rates decrease the CPU time, as observed from Figure 21.b. The good solutions have machine number patterns of 2 – 4 – 2 – 2 and 3 – 3 – 2 – 2 corresponding to strings of

01110101

10100101

,respectively. High mutation rates decrease the simulation time, meaning that mutation generally generates less-loaded systems, which is true for previous analyses. At first sight number of 0's and 1's are close to each other, and one can suspect that the mutation can increase and decrease the machine numbers at the same rate, meaning that same frequency is valid for positive and negative increments on machine numbers. But there comes the effect of the linear constraint; any positive deviation from these machine settings will lead to infeasible solutions, thus mutation cannot generate more loaded systems. In fact good solutions are the most loaded systems of their feasible solution space. Depending on this, mutation always generates solutions with low levels of machine numbers, thus less-loaded systems. That's why, high mutation rates decreases CPU time.

The only significant interaction effect, population type-mutation probability interaction presents a consistent behavior with the other cases. From Figure 21.d, the difference between the required CPU time for seeded and random starts is more significant for low mutation rates. (For further discussion check pp 96-97)

6.2.4. Conclusion

We impose a linear constraint on the optimization problem. The constraint hinders the solutions with total number of machines exceeding 10, thus shrinks the feasible space. The infeasible solutions generated during GA applications are discarded and new individuals are created instead. By this constraint, we force GA to search different parts of the solution space compared with the other cases.

Separate analyses are carried considering both the fitness response and CPU time. Table 26 summarizes the results of the analysis by stating the appropriate levels of the factors.

Table 26: Results of Factorial Design Analysis for Constrained Case

	Pop Type	**Cross P**	**Mut P**
	Fitness		
Constrained	-	-	-
	CPU Time		
Constrained	Low	-	High
General	Low	-	High

In summary, initial population type, crossover and mutation probabilities present insignificant behavior on fitness value. In fact it is a predictable result, since the nature of the good solutions shows similarities with the buffer case, and the results of buffer analysis is similar with these findings. In buffer case, there are only 4 good settings of machine numbers, besides the buffer combinations in good solutions cannot exceed 3. This results with only a very few number of good solutions in the problem domain. The same situation is relevant for the constrained case. There are only 2 good combinations of machine numbers, but this time buffer variables can have more combinations. However, the number of good solutions are also few for the constrained case as in the buffer case, and regardless of the parameter levels; GA can manage to find good solutions as in the buffer case.

The results for CPU time analysis are more fortunate, since the factors present the same behavior throughout all the cases. Starting with random populations and using high mutation rates increase the performance of GA in terms of CPU time. The consistency of the results depends on the nature of the solution space. In all cases good solutions are loaded systems of their feasible space, and mutation deviates to solutions to less-loaded systems more frequently than a more-loaded alteration. Since the population size and maximum generation number factors are at their low levels contributing a shorter GA application, the initial effect of seeded population becomes significant for the constrained case as the other cases. The comprehensive discussion about the results and resulting general pinpoints will be presented in Chapter 7 in details.

6.3. Modification of Crossover Operator

The analysis up to now has been conducted using the same settings of structural parameters. One of the main findings of this study, which is surprising to some extend, is the insignificance of crossover probability on the GA performance. Thus, in this section we further examine the crossover operator by changing its type. The results of this analysis may provide background for the further research in this area.

Recall that the well-known simple one-point crossover has been used throughout in this thesis. In our problem domain we have found that this type of crossover is ineffective in creating diverse solutions that can alter the GA performance. We now consider a new type of crossover, *uniform crossover* in our further analysis.

Uniform crossover is invented by Syswerda (1989). In this crossover mechanism, the genetic exchange occurs bit by bit. A child takes the value of its bit from first parent or second parent with equal probability of 0.5. A uniform random number stream is generated for the crossover, first to determine the crossover schedule. If the generated random number is over 0.5, the bit of the schedule corresponds to 1, and 0 in the opposite case. After determination of the crossover schedule, for every 0 in the schedule, first child takes the value of the first parent of corresponding bit. For every 1 in the schedule, first child takes the value of the bit of the second parent, while second child takes the first parent's value. The following example will clarify the situation. Suppose we have the following two strings as potential parents, and the generated crossover schedule below.

Parent 1: 11111111

Parent 2: 00000000

Schedule: 10011011

The schedule is generated by flipping coins, or generating uniform random variates as discussed previously. According to the schedule, the following children are generated.

Child 1: 01100100

Child 2: 10011011

For the first bit of child 1, since the schedule is "1", the first child takes the value of second parent, whereas second child takes the first parent's value of the same bit. The schedule is "0" for the second bit, thus first child takes the value of the first parent's second bit.

This type of crossover is applicable on various problems; since the mechanism is somewhat different from the simple one-point crossover, uniform crossover may produce more diverse solutions in our case.

Another 2^3 factorial design analysis is conducted using the original problem case. The population size and maximum generation number parameters are set to their low values as usual, and the initial population type, mutation probability factors are investigated. Instead of crossover probability, the low level of crossover corresponds to one-point crossover with 0.5 probability, and the high level corresponds to uniform crossover. 5 independent runs of GA are processed. From now on "crossover" terminology is used for the analysis of the case. Table 27.a shows the results for fitness response, and Table 27.b presents the CPU time.

Table 27.a: GA Results for Fitness Response for Crossover Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	803400	796840	795040	809880	799680
-1	1	1	787200	793400	810080	792720	798840
1	-1	1	801480	794760	792760	801800	791720
-1	-1	1	800080	799440	794720	792360	789040
1	1	-1	796480	792840	805240	802480	790240
-1	1	-1	803200	802280	778040	767880	775520
1	-1	-1	806360	800080	799440	785440	785840
-1	-1	-1	778240	787720	798520	810160	767560

Table 27.a: GA Results for CPU Time Response for Crossover Case

A	D	E	Rep1	Rep2	Rep3	Rep4	Rep5
1	1	1	5702,75	5752,61	5827,71	5741,9	5738,7
-1	1	1	5574,79	5762,64	5559,94	5816,22	5371,13
1	-1	1	5760,22	5840,01	5912,98	5812,63	5783,68
-1	-1	1	5656,01	5889,85	5296,02	5364,08	5534,52
1	1	-1	6474,15	6867,91	6560,38	6675,34	6146,45
-1	1	-1	6411,24	6277,04	5431,14	5816,02	5738,7
1	-1	-1	6417,06	6291,32	6581,89	6659,2	6600,68
-1	-1	-1	6276,26	6156,25	6063,63	6408,02	6072,87

6.3.1. Fitness Response

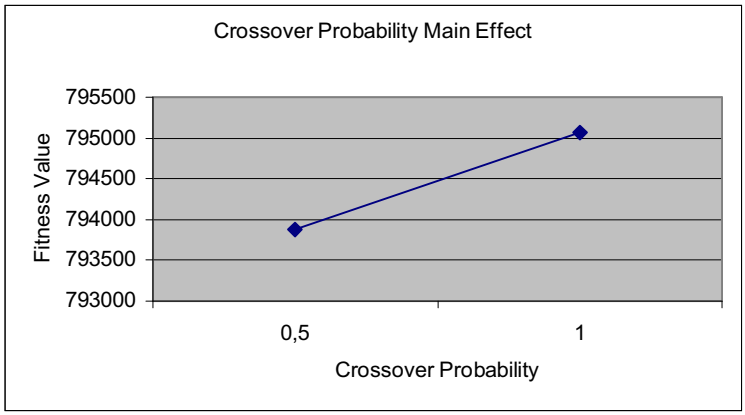
Table 28 represents the ANOVA results according to fitness response.

Table 28: ANOVA Results for Fitness Response for Crossover Case

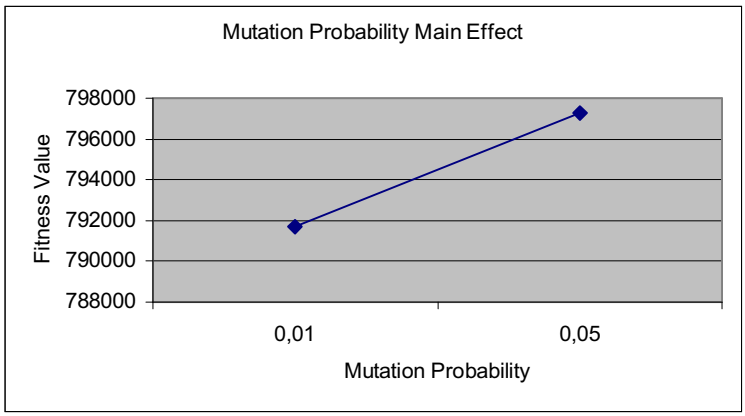
Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	897322720	7	128188960	1,24	0,310
Intercept	25247303236000	1	25247303236000	244547,40	0,000
POPTYPE	389376000	1	389376000	3,77	0,061
CROSSP	14113440	1	14113440	0,14	0,714
MUTP	311810560	1	311810560	3,02	0,092
POPTYPE * CROSSP	42271360	1	42271360	0,41	0,527
POPTYPE * MUTP	108372640	1	108372640	1,05	0,313
CROSSP * MUTP	29036160	1	29036160	0,28	0,600
POPTYPE * CROSSP * MUTP	2342560	1	2342560	0,02	0,881
Error	3303710080	32	103240940		
Total	25251504268800	40			
Corrected Total	4201032800	39			

According to 95% precision level, all the factors are insignificant, but for 90% precision level initial population type and the mutation probability have significant effect on GA performance. Even the type of the crossover is insignificant. The results are same with the results of Section 6.1.2 for 90% precision level, seeded initial population have just started to show its strength in finding better fitness, thus initial population type is insignificant for 95%

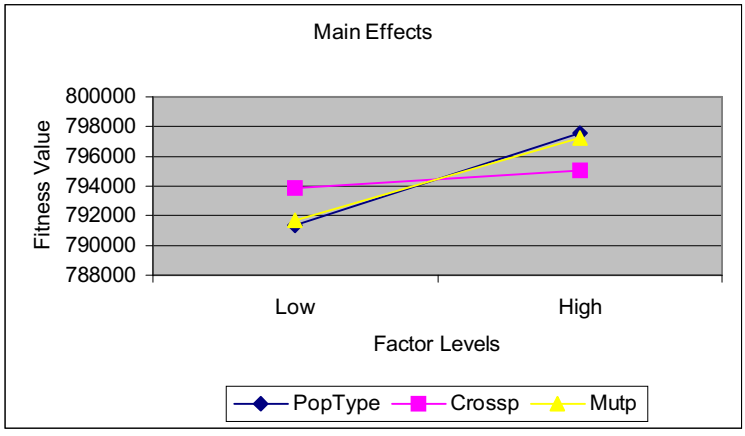
precision level, but significant for 90%. All the factors have shown the same behavior previously for the original case of Section 6.1.2. Figure 22.a presents the significance of initial population type, whereas Figure 22.b shows the mutation probability effect on fitness. All the main effects can be observed from Figure 22.c



a) Population Type Main Effect on Fitness of the Crossover Case



b) Mutation Probability Main Effect on Fitness of the Crossover Case



c) All Factors Main Effects on Fitness of the Crossover Case

Figure 22: Graphs of Significant Effects with respect to Fitness for Crossover Case

Both the initial population type, and the mutation probability has positive effects on fitness, and the reasoning behind them has already discussed in previous sections. They are valid for the crossover case, too. The main conclusion is that the crossover type does not have any effect on GA performance, and it does not have any interaction with the other parameters of the GA. In fact, this is a predictable result. The insignificance of crossover probability causes from the lack of diversity among solutions generated by the crossover operator as discussed in previous analyses. The same argument is relevant for uniform crossover operator. For the original case the good solutions (3 – 4 – 2 – 3, 4 – 4 – 3 – 3) generally have the following representations of machine numbers

$$10110110$$

$$11111010$$

The 2nd, 5th and 6th bits of the strings are different, thus any schedule of the other bits concludes with the same strings. Different from the patterns discussed in page 61, only the following extra patterns could be generated by uniform crossover

$$11110010 \longrightarrow 4 - 4 - 1 - 3$$

$$10111110 \longrightarrow 3 - 4 - 4 - 3$$

Both strings correspond to poor solutions, since the third station has only one machine in the first case, which decreases the throughput drastically. The second string is representative of a poor solution, because the 4 machines in third workstation are unnecessary. 3 – 4 – 3 – 3 is a good solution, but cost of the extra machine in third workstation cannot be compensated by the extra production achieved, thus the second one is not a good solution at all. Since these are poor solutions, the dominance of good solutions will make them disappear in the following generations.

Uniform crossover is successful in generating more diverse solutions in terms of buffer positions, but both crossover types can create same set of solutions with same machine pattern on the average. Since the fitness depends merely on the machine number, then both operators generate solutions with close fitness values, hence no significance effect occurs on GA performance depending on the type of the crossover.

According to these results, uniform crossover generates solutions with same machining patterns with the one-point crossover. Since one-point crossover is ineffective on fitness, similarly the uniform crossover does not have significant effect on fitness.

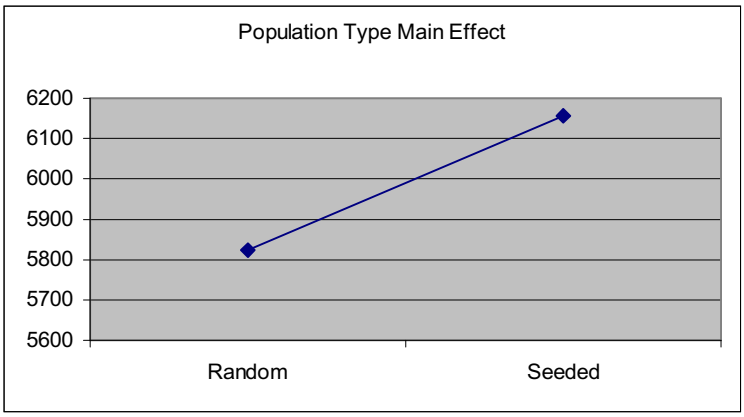
6.3.2. CPU Time Response

Table 29 presents the ANOVA results of 2^3 factorial design analysis with respect to CPU time response.

Table 29: ANOVA Results for CPU Time Response for Crossover Case

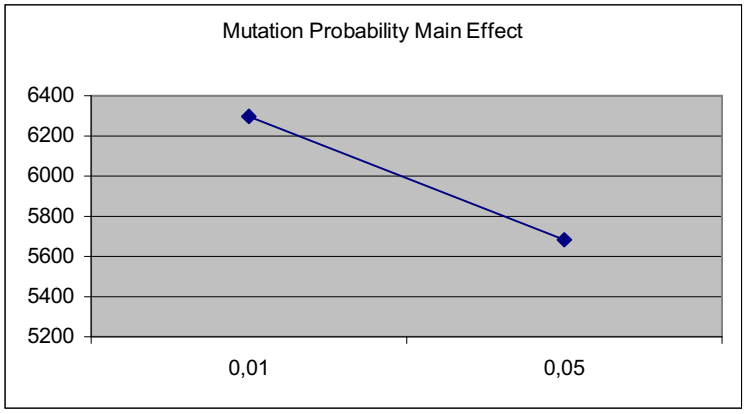
Source	Type III Sum of Squares	df	Mean Square	F	Sig.
Corrected Model	5212597	7	744656,6698	16,01	0,000
Intercept	1435490816	1	1435490816	30864,31	0,000
POPTYPE	1112623	1	1112622,736	23,92	0,000
CROSSP	31946	1	31946,23441	0,69	0,413
MUTP	3737586	1	3737586,042	80,36	0,000
POPTYPE * CROSSP	15480	1	15479,50336	0,33	0,568
POPTYPE * MUTP	165794	1	165793,9512	3,56	0,068
CROSSP * MUTP	31764	1	31764,496	0,68	0,415
POPTYPE * CROSSP * MUTP	117404	1	117403,7261	2,52	0,122
Error	1488312	32	46509,73936		
Total	1442191724	40			
Corrected Total	6700908	39			

According to 95 % precision level, population type, and mutation probability have significant effect on CPU time of GA. Figure 23.a presents the effect of initial population type; whereas Figure 23.b shows the mutation probability effect on CPU time. Figure 23.d represents the only significant interaction effect, and Figure 23.d is the graph of main effects of all three factors.

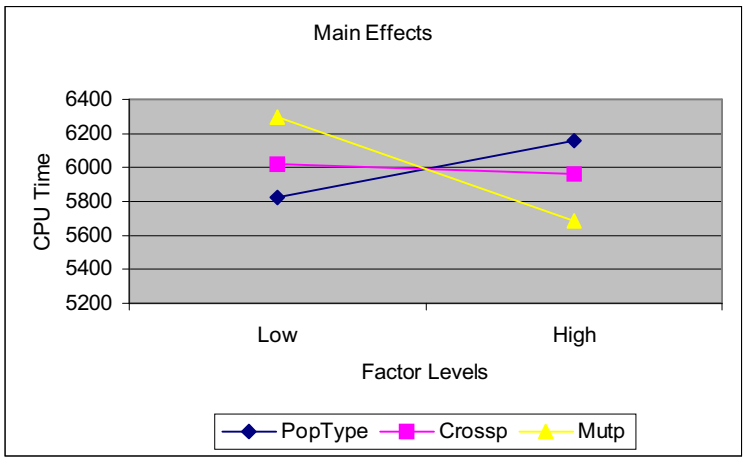


a) Population Type Main Effect on CPU Time of Crossover Case

Figure 23: Graphs of Significant Effects with respect to CPU Time for Crossover Case



b) Mutation Probability Main Effect on CPU Time of Crossover Case



c) All Factors Main Effects on CPU Time of Crossover Case

Figure 23: Graphs of Significant Effects with respect to CPU for Crossover Case (cont'd)

The results and the effects of the factors are similar with the analysis conducted in Section 6.1 for the original case. The random initial population and high mutation rate is appropriate for minimizing the CPU time. The reasons and implications are discussed in pp.69-71 and pp.90. The main result is again the insignificance of the crossover operator. This time different type of crossover is used, but even uniform crossover cannot have significant impact on CPU time. The reason is that the uniform crossover generates the same solutions with the one-point crossover on the average as discussed in the previous fitness analysis. Although uniform crossover is more successful in creating diverse solutions in terms of buffer levels, the same set of machine numbers are created by both crossover types. Since the load of the system depends on the machine number pattern, and both operators generate solutions of

same workload, there is no significant difference between CPU times of GA with different crossover types.

6.3.3. Conclusion

Our previous analyses have indicated the insignificant effect of crossover probability on both fitness and CPU time, except 1000 case. As a further experimentation, we try different type of crossover operators; hence change the structural parameter settings of proposed GA. Uniform crossover is compared with the one-point crossover having 0.5 crossover probability. Both analyses on fitness and CPU time using the original test problem reveal that not only the crossover rate but also the crossover type is insignificant to our test problem. The other factors, initial population type and mutation probability presents the same behavior as in the original case of Section 6.1 under 90% precision level. The uniform crossover does not have any effect on the other numerical factors.

CHAPTER 7

DISCUSSION

This study is conducted to analyze the effects of numerical parameters of GA on its performance under same setting of structural parameters. A test problem is used for the GA-based simulation-optimization experimentation. According to the analysis conducted on the original version of the test problem, the high levels of each factor except crossover probability are preferable for maximizing the fitness value. Low levels of population size and maximum generation number, and high level of mutation probability are concluded to be adequate for CPU time minimization objective.

After making further analysis on conflicting parameters, we determine that GA 's starting with seeded populations, having low population sizes, and maximum generations but high mutation rates produce the best performance on the test problem. Insignificance of crossover probability under the current levels forces us to apply further analysis, and it is observed that even lack of crossover operator will not produce significant alterations in both fitness and CPU time.

To observe the behavior of the parameters under different experimental conditions, we modify the test problem, and set the population size and maximum generation number factors to their low levels, since they are not effective on improving the fitness significantly; but they have drastic effects on CPU time, resulting with high run times.

The modifications can be classified into two groups, objective function's cost parameter alterations, and imposing a linear constraint to the problem. Two different alterations of cost parameters are conducted, each one forcing GA to search for different parts of the solution space; creating distinct experimental conditions. The constrained version of the problem shrinks the solution space, because there become infeasible solutions. Table 30 summarizes the experimental analyses' results for each case of the test problem.

Table 30: Summary of Results of Analyses for All Cases

	Pop Type	Cross P	Mut P
	Fitness		
Original	High	-	-
1000	-	High	High
Buffer	-	-	-
Constrained	-	-	-
General	High	High	High
	CPU Time		
Original	Low	-	High
1000	Low	-	High
Buffer	Low	-	High
Constrained	Low	-	High
General	Low	-	High

As observed from the above table, the GA parameters present consistent behavior for the minimization of CPU time objective, while different behaviors of parameters are experienced for fitness response under distinct experimental conditions. According to these results, GA starting with a seeded initial population with high crossover and mutation rates is appropriate for the fitness maximization of the test problem. The high mutation rate but random initial population with any level of crossover probability will lead good performance of GA in terms of CPU time. According to the Table 30, it can be concluded that high mutation and crossover rates are appropriate for meeting both objectives, but the general result of initial population type requires further analysis among seeded and random populations. Table 31 shows the percentage changes in fitness and CPU time according to change in initial population type from random to seeded case.

Table 31: Percentage Change in Responses According to Population Type for All Cases

	Original		1000		Buffer		Constrained	
Pop Type	**Fitness**	**CPU Time**	**Fitness**	**CPU Time**	**Fitness**	**CPU Time**	**Fitness**	**CPU Time**
Random	787822	5803,071	1122538	5589,61	988835	5845,247	594754	4445,148
Seeded	795376	6274,808	1123966	6021,633	980078	6067,813	594966	4602,012
% Change	0,9	7,7	0,1	7,7	-0,8	3,8	0,03	3,5

According to the results, seeded initial population increases the CPU time more considering the percent improvement on the fitness value. Thus giving equal weights for each response types, it is better to use random initial population. But the percentage differences are very low, hence the only a slight increase in the importance of fitness response will state the seeded population as the adequate level for initial population type. To sum up, the initial population selection is dependent on the user's preferences given to each objective.

Up to now, we have presented the computational results gathered from the analyses of different versions of the test problem under consideration. But the main aspect of this study is making general conclusions about GA parameters and presenting general guidelines for the GA applicators. Thus we have to generalize the results we obtain from the previous analyses.

The first main conclusion is that, the performance of GA depends on the nature of solution space, because even under different experimental conditions of the same test problem, significance of factors is different. Thus, GA presents different behaviors in different portions of the solution space. To generalize the computational findings, we have to state the characteristic features of the problem domain we experienced, and make conclusions about GA applications in such problem domains.

Considering the fitness response as the performance measure; the main characteristic features of our problem domain are;

- There is a dominance of a set of decision variables over *other variables* with respect to the objective function value of the optimization problem, the objective function value is directly related with the combination of this dominant set of variables (machine number dominance over buffer variables). Alteration of the values of dominant variable alters the solution's performance substantially when compared with the other decision variables.
- The good solutions for problems with different objective function parameters are highly dominant over *other solutions* with respect to the objective function value, but not very significantly diverse among each other. (Good solutions are significantly better than the other feasible solutions, but no high dominance of a good solution occurs among good solutions)
- Combining the above two features; good solutions have specific set of patterns in dominant set of decision variables, and these solutions are dominant over other feasible solutions among the solution space.

These properties of the problem domain generate a rapid convergent behavior of GA. The effect of the dominant decision variable on the fitness value (objective function value), guides GA to search for different levels of that variable instead of dealing with the patterns of other variables. Once a good solution is reached, the convergence occurs rapidly. As the fitness value of that solution is highly dominant over other feasible solutions, reproduction mechanism always selects that solution as the potential parent of the following generation. Thus GA will search among the neighborhood of that solution.

Considering the CPU time, more general statements can be made. With a solution space presenting a rapid convergent behavior, the CPU time performance of GA is directly related with the load of the good solutions. The dominant decision variable of the fitness case, also presents dominance in terms of the effects on simulation time in the problem domain we discuss.

Presenting the features and the type of the problem domain, we can now discuss the relevance of effects of each factor one by one, and make general conclusions in this problem domain.

Initial Population Type

Initial population type is effective in such problem domains. Dominance of good solutions creates a rapid convergence of GA, which is experienced in all our analyses. The performance of GA in terms of fitness will depend on the search proceeded among the solutions with these specific patterns of dominant variables. Seeded initial populations having good solutions help GA use its time more efficiently, because convergence occurs more rapidly when compared to random start. A random start should find a good solution first, thus GA will spend some run time for this process. At the same time seeded start has already started with such a good solution and search among the good solutions by changing the values of other variables' levels.

The positive effect of seeded population also depends on the number of good solutions. As this number decreases the difference in performance of GA between random and seeded starts diminishes. Regardless of the type of the initial population, since convergence occurs rapidly and the converged solutions is one of these few good solutions; thus probability of finding the same solution under different starting conditions increases as the number of good solution decreases. Moreover, less number of good solutions corresponds to few combinations

of dominant decision variable. This property is relevant for our analysis; although seeded population is effective in the original problem, in the cases with less number of good solutions (buffer, constrained cases) the significance of a seeded start diminished.

If good solutions correspond to loaded systems in such domains; then seeded initial population may produce an increase in CPU time. The main reason is; at least for the few starting generations, GA proceeding with seeded initial population will have more loaded systems compared with the random start case. Because a random start is searching for good solutions with loaded systems while seeded started GA is continuing its search among good hence loaded solutions. This may create discrimination among the initial population types, where random initial population is better for such applications. But the opposite is true, when the good solutions correspond to less-loaded systems. That time from the same reasoning seeded initial population decreases the simulation time, so the CPU time.

The appropriate level of initial population type depends on the levels of other parameters. If the population size or/and maximum generation number is high enough, the difference among the GA applications with distinct starting conditions may not show a significant behavior. As discussed previously, seeded start has more time to proceed among good solutions, but if enough time is given for GA application, a random start can process adequate number of good solutions, hence the difference will disappear.

Under the problem domain we consider, initial population type effect on CPU time depends on the rate of mutation. This result is experienced in all of our cases. As the mutation probability increases, the initial advantage of seeded populations will decrease, since more frequent alterations are made on good solutions of the initial few generations of seeded start, which may result with poor solutions. The frequent deviations from good solutions decrease the number of more good solutions processed in the seeded start compared with a random start, thus the effect of seeded start is mitigated.

Population Size and Maximum Generation Number

For the problems with domains presenting features mentioned above, high levels of population size and maximum generation number are inappropriate. Increase in the levels of these factors, increases the number of solutions processed by the algorithm, thus the probability of hitting a better solution improves, but CPU time increases drastically at the same time. Rapid convergent behavior of GA under such problem domains leads to reaching

good solutions in short time, thus moderate levels are appropriate for a GA application. High levels will increase the CPU time enormously without improving the fitness value significantly as experienced in the original case of our analyses.

Crossover Probability

The crossover operator generally does not have significant effects on GA performance in such problem domains. Crossover is mainly responsible from the maintenance of good traits of solutions in future generations. It provides the exchange of genetic materials among the previously selected solutions, and generates new solutions by this manner. The dominance of good solutions leading to a rapid convergence of GA, forces selection procedure to select the solutions, with same good patterns in terms of dominant decision variable, as the potential parents for the next generation. The dominant decision variable's settings of good solutions are similar to each other; thus, two selected potential parents cannot create diverse solutions.

Even for many trials, since the dominant decision variables' patterns of both potential parents are exactly the same, to cross or not, from a cross-site selected inside the binary representation of the dominant variable will produce same children. So crossover does not have a significant impact on fitness response of problems with such domains.

Any level of crossover probability will be adequate for GA applications on such problem domains. But since crossover is one of the strongest features of the algorithm, it is convenient to set the crossover rate to positive values (not 0); because discarding the crossover operator may affect the basic principles of GA, resulting with GA presenting erratic or unexpected behavior.

Mutation Probability

The strongest feature of GA on such problem domains is the mutation operator. Rapid convergence of GA, depending on the nature of solution space, decreases the number of different solutions in populations as the generations proceed. Mutation creates diversity in the populations. Probability of obtaining better solutions is directly proportional with the number of solutions processed, which has similar dominant variable pattern but different levels of other variables. After some initial generations all the populations are full of solutions with good pattern of dominant variable. Thus the discrimination among these solutions is achieved by selecting the appropriate levels for the other decision variables.

As discussed previously, and shown analytically (pp.63) that mutation operator can search more such solutions compared with the crossover operator. Thus increasing the mutation probability increases the chance of hitting a better solution. Although mutation rate ranges between 0.001 and 0.05 in GA literature, our further analyses revealed that higher levels like 0.4 might result with the best performance.

The explanations stated in the previous paragraphs aim to make general conclusions about the effects of numerical parameters of GA on the performance in terms of fitness and CPU time on the problem domains with the defined characteristic features. These findings construct guidelines for GA-based simulation-optimization applications on problems with similar domains, and help the researchers in understanding the reasons and implications of the effects of numerical parameters of GA on its performance.

As a future research direction, the effects of same parameters on different problem domains may be investigated, and empirical studies can be conducted to state whether numerical parameters present the same behavior or not. This analysis is conducted with a pre-specified setting of structural parameters of GA. The significance of the numerical parameters with different structural parameter setting can be examined. In further analyses we compare the uniform crossover with one-point crossover; and the computation results present the insignificance of uniform crossover, too. But other levels of structural parameters may alter the effect of the numerical parameters even under the same problem domain. Studies analyzing the effectiveness and interactions of both structural and numerical parameters on GA performance will make up another research direction.

APPENDIX A

SIMAN CODE OF SIMULATION MODEL

Model Frame

```
BEGIN;
CREATE;
READ,IN1:MC1;
READ,IN2:MC2;
READ,IN3:MC3;
READ,IN4:MC4;
READ,IN5:BUF2;
READ,IN6:BUF3;
READ,IN7:BUF4;
ALTER:MACHINE1,(MC1-1);
ALTER:MACHINE2,(MC2-1);
ALTER:MACHINE3,(MC3-1);
ALTER:MACHINE4,(MC4-1);
DUPLICATE:10000;
QUEUE,WORKSTATIONQ1;
SEIZE:MACHINE1;
IF:(TNOW.LT.(961*60)).AND.(TNOW.GE.57600).AND.(CONTROL.EQ.0);
   WRITE,OUTFILE1:TOTALCOST;
   ASSIGN:CONTROL=1;
ENDIF;
DELAY:EXPO(0.33333*60,5);
QUEUE,DUMMY1;
SCAN:NQ(WORKSTATIONQ2).LT.BUF2;
RELEASE:MACHINE1;

QUEUE,WORKSTATIONQ2;
SEIZE:MACHINE2;
DELAY:EXPO(0.5*60,5);
QUEUE,DUMMY2;
SCAN:NQ(WORKSTATIONQ3).LT.BUF3;
RELEASE:MACHINE2;

QUEUE,WORKSTATIONQ3;
SEIZE:MACHINE3;
DELAY:EXPO(0.2*60,5);
QUEUE,DUMMY3;
SCAN:NQ(WORKSTATIONQ4).LT.BUF4;
RELEASE:MACHINE3;

QUEUE,WORKSTATIONQ4;
SEIZE:MACHINE4;
DELAY:EXPO(0.25*60,5);
RELEASE:MACHINE4;
IF:(TNOW.GE.(240*60));
ASSIGN:THROUGHPUT=THROUGHPUT+1;
ASSIGN:TOTALCOST=200*THROUGHPUT-25000*(MC1+MC2+MC3+MC4)-
1000*(BUF2+BUF3+BUF4);
ENDIF;
DELAY:0:DISPOSE;
END;
```

Experimental Frame

```
BEGIN;
PROJECT,THESIS1,ONUR BOYABATLI;
VARIABLES:MC1:MC2:MC3:MC4:BUF2:BUF3:BUF4:THROUGHPUT:TOTALCOST:CONTROL
;
QUEUES:WORKSTATIONQ1:WORKSTATIONQ2:WORKSTATIONQ3:WORKSTATIONQ4
      :DUMMY1:DUMMY2:DUMMY3;
RESOURCES:MACHINE1:MACHINE2:MACHINE3:MACHINE4;
FILES:IN1,"D1.dat",SEQ,FREE,,,Rewind:IN2,"D2.dat",SEQ,FREE,,,Rewind:
      IN3,"D3.dat",SEQ,FREE,,,Rewind:IN4,"D4.dat",SEQ,FREE,,,Rewind:
      IN5,"D5.dat",SEQ,FREE,,,Rewind:IN6,"D6.dat",SEQ,FREE,,,Rewind:
      IN7,"D7.dat",SEQ,FREE,Rewind:OUTFILE1,"Out1.dat",SEQ,FREE;
REPLICATE,5,,57660,,,14400;
END;
```

APPENDIX B

C CODE OF GENETIC ALGORITHM

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>

#define PopSize 10
#define MaxGen 2
#define NumDec 7
#define NumRep 5
#define NumBest 1

#define pcross 0.50
#define pmut 0.05

#define LowerD1 1
#define LowerD2 1
#define LowerD3 1
#define LowerD4 1
#define LowerD5 1
#define LowerD6 1
#define LowerD7 1
#define LengthD1 2
#define LengthD2 2
#define LengthD3 2
#define LengthD4 2
#define LengthD5 4
#define LengthD6 4
#define LengthD7 4
#define MaxString 20 /* MaxString=LengthD1+LengthD2+...+LengthD7 */

struct indiv{  int chromosome[MaxString];
               short int dec1;
               short int dec2;
               short int dec3;
               short int dec4;
               short int dec5;
               short int dec6;
               short int dec7;
               double fitness[NumRep+1];
               short int individ;
               short int parent1;
               short int parent2;
               short int xsite;
               short int nmut;
             } oldpop[PopSize],newpop[PopSize],bestindiv[NumBest];

int child1[MaxString];
int child2[MaxString];

int gen,numcross,totmut,bestgen;
double sumfitness,maxfit,minfit,avgfit,globalmax,temp;
FILE *infile,*outfile;
char inputstr[MaxString];
long int seed;
```

```
time_t t1,t2;

double randnumgen()
{double result;
 result=(double) rand() / 32767;
 return result;
};

int main()
{
 int cnt1,cnt2,cnt3;
 int choice1,choice2;
 double randomnum;

 srand(time(NULL));
 t1=time(NULL);
 seed=rand();

 globalmax=0;

ReadProblem();
gen=0;
while(gen<MaxGen)
{
 ++gen;
 printf("\n\nGeneration number: %i\n\n",gen);
 sumfitness=0.0;
 numcross=0;temp=100000000;
 for(cnt1=0;cnt1<PopSize;++cnt1)

{if(oldpop[cnt1].fitness[NumRep]<temp){temp=oldpop[cnt1].fitness[NumR
ep];};
 };
 if(temp<0)
 {for(cnt1=0;cnt1<PopSize;++cnt1)
  {oldpop[cnt1].fitness[NumRep]=oldpop[cnt1].fitness[NumRep]-temp;
  };
 };
 for(cnt1=0;cnt1<PopSize;++cnt1)
 {sumfitness=sumfitness+(oldpop[cnt1].fitness[NumRep]);
 };
 cnt1=0;
 while(cnt1<PopSize)
 {choice1=selectindiv();
  choice2=selectindiv();
  crossover(choice1,choice2,cnt1);
  cnt1=cnt1+2;
 };
 for(cnt2=0;cnt2<PopSize;++cnt2)
 {
   outfile=fopen("D1.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec1);
   fclose(outfile);
   outfile=fopen("D2.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec2);
   fclose(outfile);
   outfile=fopen("D3.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec3);
   fclose(outfile);
   outfile=fopen("D4.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec4);
```

```
   fclose(outfile);
   outfile=fopen("D5.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec5);
   fclose(outfile);
   outfile=fopen("D6.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec6);
   fclose(outfile);
   outfile=fopen("D7.dat","w");
   fprintf(outfile,"%hi\n",newpop[cnt2].dec7);
   fclose(outfile);

  system("siman law.p");
  infile=fopen("Out1.dat","r");
  for(cnt3=0;cnt3<NumRep;++cnt3)
  {fscanf(infile,"%lf\n",&(newpop[cnt2].fitness[cnt3]));
  };
  fclose(infile);
  newpop[cnt2].fitness[NumRep]=0;
  for(cnt3=0;cnt3<NumRep;++cnt3)
  {newpop[cnt2].fitness[NumRep]=
   newpop[cnt2].fitness[cnt3]+newpop[cnt2].fitness[NumRep];
  };
  newpop[cnt2].fitness[NumRep]=newpop[cnt2].fitness[NumRep]/NumRep;
 };
 compstats();
 if(maxfit>globalmax)
 {globalmax=maxfit;
  bestgen=gen;
  cnt1=0;
  while(newpop[cnt1].fitness[NumRep]!=maxfit)
  {++cnt1;
  };
  bestindiv[0].individ=newpop[cnt1].individ;
  bestindiv[0].dec1=newpop[cnt1].dec1;
  bestindiv[0].dec2=newpop[cnt1].dec2;
  bestindiv[0].dec3=newpop[cnt1].dec3;
  bestindiv[0].dec4=newpop[cnt1].dec4;
  bestindiv[0].dec5=newpop[cnt1].dec5;
  bestindiv[0].dec6=newpop[cnt1].dec6;
  bestindiv[0].dec7=newpop[cnt1].dec7;
  bestindiv[0].fitness[0]=newpop[cnt1].fitness[0];
  bestindiv[0].fitness[1]=newpop[cnt1].fitness[1];
  bestindiv[0].fitness[2]=newpop[cnt1].fitness[2];
  bestindiv[0].fitness[3]=newpop[cnt1].fitness[3];
  bestindiv[0].fitness[4]=newpop[cnt1].fitness[4];
  bestindiv[0].fitness[5]=newpop[cnt1].fitness[5];
  bestindiv[0].fitness[6]=newpop[cnt1].fitness[6];
  bestindiv[0].parent1=newpop[cnt1].parent1;
  bestindiv[0].parent2=newpop[cnt1].parent2;
  bestindiv[0].xsite=newpop[cnt1].xsite;
  bestindiv[0].nmut=newpop[cnt1].nmut;
  for(cnt2=0;cnt2<MaxString;++cnt2)
  {bestindiv[0].chromosome[cnt2]=newpop[cnt1].chromosome[cnt2];
  };
 outfile=fopen("best.dat","w");
  fprintf(outfile,"%i %hi %hi %hi %hi %hi %hi %hi %hi %lf %lf %lf %lf
%lf %lf\n",
 bestgen,
 (bestindiv[0].individ),
 (bestindiv[0].dec1),
 (bestindiv[0].dec2),
```

```
 (bestindiv[0].dec3),
 (bestindiv[0].dec4),
 (bestindiv[0].dec5),
 (bestindiv[0].dec6),
 (bestindiv[0].dec7),
 (bestindiv[0].fitness[0]),
 (bestindiv[0].fitness[1]),
 (bestindiv[0].fitness[2]),
 (bestindiv[0].fitness[3]),
 (bestindiv[0].fitness[4]),
 (bestindiv[0].fitness[5])
  );
 fclose(outfile);
 };
 writeresult();
 popassign();
};
t2=time(NULL);

outfile=fopen("time.dat","w");
fprintf(outfile,"%f seconds \n",difftime(t2,t1));
fclose(outfile);

return 0;
};

int selectindiv()
{double sum,num;
 int cnt;
 sum=0;cnt=0;
 num=randnumgen()*sumfitness;
 sum=sum+oldpop[cnt].fitness[NumRep];
 while(sum<=num)
 {++cnt;
  sum=sum+oldpop[cnt].fitness[NumRep];
 };
 return cnt;
};

int crossover(int choice1, int choice2, int cnt1)
{
double randnum;
short int jcross,cnt2,cnt3,mutnum1,mutnum2;

mutnum1=0;mutnum2=0;cnt3=0;

randnum=randnumgen();
if(randnum<=pcross)
{jcross=1+floor(randnumgen()*(MaxString-1));
 ++numcross;
}
else
{jcross=MaxString;};

for(cnt2=0;cnt2<jcross;++cnt2)
{
child1[cnt2]=oldpop[choice1].chromosome[cnt2];
child2[cnt2]=oldpop[choice2].chromosome[cnt2];
};

for(cnt2=jcross;cnt2<MaxString;++cnt2)
```

```
{
child1[cnt2]=oldpop[choice2].chromosome[cnt2];
child2[cnt2]=oldpop[choice1].chromosome[cnt2];
};

for(cnt2=0;cnt2<MaxString;++cnt2)
{
if(randnumgen()<=pmut)
 {if(child1[cnt2]==1){child1[cnt2]=0;}
  else {child1[cnt2]=1;};
  ++mutnum1;
 };

};

for(cnt2=0;cnt2<MaxString;++cnt2)
{
if(randnumgen()<=pmut)
 {if(child2[cnt2]==1){child2[cnt2]=0;}
  else {child2[cnt2]=1;};
  ++mutnum2;
 };

};

/* check for infeasibility */
/* check for infeasibility */

newpop[cnt1].individ=cnt1;
newpop[cnt1+1].individ=cnt1+1;
newpop[cnt1].xsite=jcross;
newpop[cnt1+1].xsite=jcross;
newpop[cnt1].nmut=mutnum1;
newpop[cnt1+1].nmut=mutnum2;
newpop[cnt1].parent1=choice1;
newpop[cnt1+1].parent1=choice1;
newpop[cnt1].parent2=choice2;
newpop[cnt1+1].parent2=choice2;

for(cnt2=0;cnt2<MaxString;++cnt2)
{newpop[cnt1].chromosome[cnt2]=child1[cnt2];};
for(cnt2=0;cnt2<MaxString;++cnt2)
{newpop[cnt1+1].chromosome[cnt2]=child2[cnt2];};

newpop[cnt1].dec1=LowerD1;
newpop[cnt1+1].dec1=LowerD1;
for(cnt2=0;cnt2<LengthD1;++cnt2)
{
 newpop[cnt1].dec1=
 newpop[cnt1].dec1+
 (pow(2,(LengthD1-1)-cnt2)*newpop[cnt1].chromosome[cnt2]);
 newpop[cnt1+1].dec1=
 newpop[cnt1+1].dec1+
 (pow(2,(LengthD1-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2]);
};
 cnt3=cnt3+LengthD1;

newpop[cnt1].dec2=LowerD2;
newpop[cnt1+1].dec2=LowerD2;
for(cnt2=0;cnt2<LengthD2;++cnt2)
{
```

```
newpop[cnt1].dec2=
 newpop[cnt1].dec2+
 (pow(2,(LengthD2-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec2=
 newpop[cnt1+1].dec2+
 (pow(2,(LengthD2-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD2;

newpop[cnt1].dec3=LowerD3;
newpop[cnt1+1].dec3=LowerD3;
for(cnt2=0;cnt2<LengthD3;++cnt2)
{
newpop[cnt1].dec3=
 newpop[cnt1].dec3+
(pow(2,(LengthD3-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec3=
 newpop[cnt1+1].dec3+
(pow(2,(LengthD3-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD3;

newpop[cnt1].dec4=LowerD4;
newpop[cnt1+1].dec4=LowerD4;
for(cnt2=0;cnt2<LengthD4;++cnt2)
{
newpop[cnt1].dec4=
 newpop[cnt1].dec4+
(pow(2,(LengthD4-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec4=
 newpop[cnt1+1].dec4+
(pow(2,(LengthD4-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD4;

newpop[cnt1].dec5=LowerD5;
newpop[cnt1+1].dec5=LowerD5;
for(cnt2=0;cnt2<LengthD5;++cnt2)
{
newpop[cnt1].dec5=
 newpop[cnt1].dec5+
(pow(2,(LengthD5-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec5=
 newpop[cnt1+1].dec5+
(pow(2,(LengthD5-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD5;

newpop[cnt1].dec6=LowerD6;
newpop[cnt1+1].dec6=LowerD6;
for(cnt2=0;cnt2<LengthD6;++cnt2)
{
newpop[cnt1].dec6=
 newpop[cnt1].dec6+
(pow(2,(LengthD6-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec6=
 newpop[cnt1+1].dec6+
(pow(2,(LengthD6-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD6;
```

```
newpop[cnt1].dec7=LowerD7;
newpop[cnt1+1].dec7=LowerD7;
for(cnt2=0;cnt2<LengthD7;++cnt2)
{
newpop[cnt1].dec7=
 newpop[cnt1].dec7+
(pow(2,(LengthD7-1)-cnt2)*newpop[cnt1].chromosome[cnt2+cnt3]);
newpop[cnt1+1].dec7=
 newpop[cnt1+1].dec7+
(pow(2,(LengthD7-1)-cnt2)*newpop[cnt1+1].chromosome[cnt2+cnt3]);
};
 cnt3=cnt3+LengthD7;

return 0;
};

int compstats()
{int cnt;
 double max,min,avg;

 max=0;avg=0;totmut=0;
 min=1000000000;
 for(cnt=0;cnt<PopSize;++cnt)
 {

if(newpop[cnt].fitness[NumRep]<min){min=newpop[cnt].fitness[NumRep];}
;

if(newpop[cnt].fitness[NumRep]>max){max=newpop[cnt].fitness[NumRep];}
;
 avg=avg+newpop[cnt].fitness[NumRep];
 totmut=totmut+newpop[cnt].nmut;
 };
 avg=avg/PopSize;

 minfit=min;
 maxfit=max;
 avgfit=avg;

 return 0;
};

int writeresult()
{ char filename[10];
  int cnt1,cnt2;

  sprintf(filename,"%s%i%s","gen",gen,".dat");
  outfile=fopen(filename,"w");
 if(temp<0){fprintf(outfile,"Warning! Negative minfitness value!
%f\n",temp);};
 for(cnt1=0;cnt1<PopSize;++cnt1)
{
 fprintf(outfile,"%3hi ",newpop[cnt1].individ);
 for(cnt2=0;cnt2<MaxString;++cnt2)
 {fprintf(outfile,"%i",newpop[cnt1].chromosome[cnt2]);
 };
 fprintf(outfile," ");
 fprintf(outfile,"%3hi %3hi %3hi %3hi %3hi %3hi %3hi %10lf %10lf
%10lf %10lf %10lf %10lf %3hi %3hi %3hi %3hi\n",
 (newpop[cnt1].dec1),
 (newpop[cnt1].dec2),
```

```
 (newpop[cnt1].dec3),
 (newpop[cnt1].dec4),
 (newpop[cnt1].dec5),
 (newpop[cnt1].dec6),
 (newpop[cnt1].dec7),
 (newpop[cnt1].fitness[0]),
 (newpop[cnt1].fitness[1]),
 (newpop[cnt1].fitness[2]),
 (newpop[cnt1].fitness[3]),
 (newpop[cnt1].fitness[4]),
 (newpop[cnt1].fitness[5]),
 (newpop[cnt1].parent1),
 (newpop[cnt1].parent2),
 (newpop[cnt1].xsite),
 (newpop[cnt1].nmut)
 );
};

fprintf(outfile,"Generation number=%i\n",gen);
fprintf(outfile,"Maximum fitness=%lf\n",maxfit);
fprintf(outfile,"Minimum fitness=%lf\n",minfit);
fprintf(outfile,"Average fitness=%lf\n",avgfit);
fprintf(outfile,"Total fitness=%lf\n",avgfit*PopSize);
fprintf(outfile,"Total number of crossovers=%i\n",numcross);
fprintf(outfile,"Total number of mutations=%i\n",totmut);
fclose(outfile);
return 0;
};

int popassign()
{int cnt1,cnt2;

 for(cnt1=0;cnt1<PopSize;++cnt1)
{
 oldpop[cnt1].individ=newpop[cnt1].individ;
 oldpop[cnt1].dec1=newpop[cnt1].dec1;
 oldpop[cnt1].dec2=newpop[cnt1].dec2;
 oldpop[cnt1].dec3=newpop[cnt1].dec3;
 oldpop[cnt1].dec4=newpop[cnt1].dec4;
 oldpop[cnt1].dec5=newpop[cnt1].dec5;
 oldpop[cnt1].dec6=newpop[cnt1].dec6;
 oldpop[cnt1].dec7=newpop[cnt1].dec7;
 oldpop[cnt1].fitness[0]=newpop[cnt1].fitness[0];
 oldpop[cnt1].fitness[1]=newpop[cnt1].fitness[1];
 oldpop[cnt1].fitness[2]=newpop[cnt1].fitness[2];
 oldpop[cnt1].fitness[3]=newpop[cnt1].fitness[3];
 oldpop[cnt1].fitness[4]=newpop[cnt1].fitness[4];
 oldpop[cnt1].fitness[5]=newpop[cnt1].fitness[5];
 oldpop[cnt1].fitness[6]=newpop[cnt1].fitness[6];
 oldpop[cnt1].parent1=newpop[cnt1].parent1;
 oldpop[cnt1].parent2=newpop[cnt1].parent2;
 oldpop[cnt1].xsite=newpop[cnt1].xsite;
 oldpop[cnt1].nmut=newpop[cnt1].nmut;
 for(cnt2=0;cnt2<MaxString;++cnt2)
 {oldpop[cnt1].chromosome[cnt2]=newpop[cnt1].chromosome[cnt2];
 };
};
return 0;
};

int ReadProblem()
```

```
{
int cnt1,cnt2;
 infile=fopen("sicik.dat","r");
 for(cnt1=0;cnt1<PopSize;++cnt1)
{
 fscanf(infile,"%hi
%20c%hi%hi%hi%hi%hi%hi%hi%lf%lf%lf%lf%lf%lf%hi%hi%hi%hi\n",
 &(oldpop[cnt1].individ),
 inputstr,
 &(oldpop[cnt1].dec1),
 &(oldpop[cnt1].dec2),
 &(oldpop[cnt1].dec3),
 &(oldpop[cnt1].dec4),
 &(oldpop[cnt1].dec5),
 &(oldpop[cnt1].dec6),
 &(oldpop[cnt1].dec7),
 &(oldpop[cnt1].fitness[0]),
 &(oldpop[cnt1].fitness[1]),
 &(oldpop[cnt1].fitness[2]),
 &(oldpop[cnt1].fitness[3]),
 &(oldpop[cnt1].fitness[4]),
 &(oldpop[cnt1].fitness[5]),
 &(oldpop[cnt1].parent1),
 &(oldpop[cnt1].parent2),
 &(oldpop[cnt1].xsite),
 &(oldpop[cnt1].nmut)
 );

 for(cnt2=0;cnt2<MaxString;++cnt2)
 {if(inputstr[cnt2]=='1'){oldpop[cnt1].chromosome[cnt2]=1;}
 else {oldpop[cnt1].chromosome[cnt2]=0;};
 };
};
fclose(infile);

return 0;
};
```

BIBLIOGRAPHY

Alander, J.T. (1992) On optimal population size of genetic algorithms. *Proceedings of CompEuro 92*, 65-70

Azadivar, F. (1992) A tutorial on simulation optimization. *Proceedings of the 1992 Winter Simulation Conference*,198-204

Azadivar, F. and Lee, Y. (1988) Optimization of discrete variable stochastic systems by computer simulation. *Mathematics and Computers in Simulation*, **30**, 331-345

Azadivar, F. and Tompkins, G. (1999) Simulation optimization with qualitative variables and structural model changes: A genetic algorithm approach. *European Journal of Operational research*, **113**, 169-182

Back, T. (1993) Optimal mutation rates in genetic search. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 2-8

Bengu, G. and Haddock, J. (1986) A generative simulation optimization system. *Computers&Industrial Engineering*,**10**,301-313

Cruz, J.A. and Haddock, J. (1993) A general scheme of simulation optimization using a gnetic algorithm. Technical Report, Rensselaer Polytechnic Institute, New York

Davis, L. (1991) Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York.

De Jong, K.A. (1975) An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan.

Fogarty, T.C. (1989) Varying the probability of mutation in the genetic algorithm. *Proceedings of the Third International Conference on Genetic Algorithms*, 104-109

Fontanilli, F., Vincent, A. and Ponsonnet, R. (2000) Flow simulation and genetic algorithms as optimization tools. *International Journal of Production Economics*, **64**, 91-100

Fu, C.M. (1994) Optimization via simulation: A review. *Annals of Operations Research*,**53**,199-247

Glover, F. and Laguna M. (1997) Tabu Search. Kluwer Academic Publishers, Norwell, Massachusettes

Goldberg, D. (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, reading, M.A.

Goldberg, D. (1989) Sizing populations for serial and parallel genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 70-79

Haddock, J. and Mittenhal, J. (1992) Simulation optimization using simulated annealing. *Computers and Industrial Engineers*, **22**, 387-395

Hamamoto, S.,Yih, Y. and Salvendy, G. (1999) Development and validation of genetic algorithm-based facility layout-a case study in the pharmaceutical industry. *International Journal of Production Research*, **37**, 749-768

Ho, Y.C., Eyler, A. and Chien, T.T. (1979) A gradient technique for general buffer-storage design in a serial production line. *International Journal of Production Research*,**17**, 557-580

Holland, J.H. (1975) Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor

Hooke, R. and Jeeves, T.A. (1961) A direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*,**8**,212-229

Jacobsen, S.H. and Schruben, L.W. (1987) Techniques for simulation response optimization. *Operations Research Letters*,**8**, 1-9

Janikow, C.Z., and Michalewicz, Z. (1991) AN experimental comparison of binary and floating point representations in genetic algorithms. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 31-36

Kirkpatrick, S., Gelatt, C. and Vecchi, P. (1983) Optimization by simulated annealing. *Science*,**221**,671-680

Law, A.M., McComas M.G. (2000) Simulation-based optimization. *Proceedings of the 2000 Winter Simulation Conference*, 46-49

Lee, S.G., Khoo L.P. and Yin, X.F. (2000) Optimizing assembly line through simulation augmented by genetic algorithms. *The International Journal of Advanced Manufacturing Technology*, **16**, 220-228

Man, K.F., Tang K.S. and Kwong S. (1999) Genetic Algorithms. Springer-Verlag, London

Meketon, M.S. (1987) Optimization in simulation: A survey of recent results. *Proceedings of the 1987 Winter Simulation Conference*, 58-67

Michalewicz, Z. (1992) Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag, Berlin Heidelberg

Montgomery, D.C. (1991) *Design and Analysis of Experiments*, Wiley, New York

Nakano, R. and Yamada, T. (1991) Conventional genetic algorithm for job shop problems. *Proceedings of the Fourth International Conference on Genetic Algorithms*, 474-479

Nelder, J.A. and Mead, R. (1965) A simplex method for function minimization. *Computation Journal*, **7**, 308-311

Pierreval, H. and Tautou L. (1997) Using evolutionary algorithms and simulation for the optimization of manufacturing systems. *IIE Transactions*, **29**, 181-189

Reeves, C.R. (1992) A genetic algorithm for flowshop sequencing. *Computers and Operational Research*.

Reeves, C.R. (1993) Modern Heuristic Techniques for Combinatorial Problems. Wiley, New York.

Richardson, J.T., Palmer M.R., Liepins, G.E. and Hilliard, M. (1989) Some guidelines for genetic algorithms with penalty functions. *Proceedings of the Third International Conference on Genetic Algorithms*, 191-197

Robbins, H. and Monro, S. (1951) A stochastic approximation method. *Annals of Mathematical Statistics*,**22**,400-407

Safizadeh,M.H. (1990) Optimization in simulation: Current issues and future outlook. *Naval Research Logistics*,**37**,807-825

Schaffer, J.D., Caruna R. A., Eshelman L.J. and Das, R. (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proceedings of the Third International Conference on Genetic Algorithms*, 51-60

Schruben, L.W. and Cogliano, V.J. (1981) Simulation sensitivity analysis: A frequency domain approach. *Proceedings of Winter Simulation Conference*,455-459

Shang, J.S. and Tadikammalla, P.R. (1993) Output maximization of a CIM system: Simulation and statistical approach. *International Journal of Production Research*, **31**, 19-41

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

Suresh, G. and Sahu, S. (1994) Stochastic assembly line balancing using simulated annealing. *International Journal of Production Research*, **32**, 1801-1810

Suresh, G., Vinod, V.V. and Sahu, S. (1995) A genetic algorithm for facility layout. *International Journal of Production Research*, **33**, 3411-3423

Syswerda, G. (1989) Uniform Crossover in Genetic Algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, 2-9

Tekin, E. and Sabuncuoglu, I. (2000) Simulation Optimization: A Comprehensive Review on Theory and Applications

Teleb, R. and Azadivar, F. (1994) A methodology for solving multi-objective simulation problems. *European Journal of Operational Research*, **72**, 135-145

Yunker, J.M. and Tew, J.D. (1994) Simulation optimization by genetic search. *Mathematics and Computers in Simulation*, **37**, 17-28

Weintraub, A., Cormier, D., Hodgson, T., King, R., Wilson, J. and Zozom, A. (1999) Scheduling with alternatives: A link between process planning and scheduling. *IIE Transactions*, **31**, 1093-1102.

Wellman, M.A. and Gemmill, D.D. (1995) A genetic algorithm approach to optimization of asynchronous automatic assembly systems. *The International Journal of Flexible Manufacturing Systems*, **7**, 27-46