AN ABSTRACT OF THE THESIS OF

Chinmaya S. Hardas for the degree of Master of Science in Industrial Engineering presented on December 11, 2003.

Title: Component Placement Sequence Optimization in Printed Circuit Board Assembly using Genetic Algorithms.

Abstract approved:

Redacted for Privacy

Toni L. Doolen

Over the last two decades, the assembly of printed circuit boards (PCB) has generated a huge amount of industrial activity. One of the major developments in PCB assembly was introduction of surface mount technology (SMT). SMT has displaced through-hole technology as a primary means of assembling PCB over the last decade. It has also made it easy to automate PCB assembly process.

The component placement machine is probably the most important piece of manufacturing equipment on a surface mount assembly line. It is used for placing components reliably and accurately enough to meet the throughput requirements in a cost-effective manner. Apart from the fact that it is the most expensive equipment on the PCB manufacturing line, it is also often the bottleneck. There are a quite a few areas for improvements on the machine, one of them being component placement sequencing. With the number of components being placed on a PCB ranging in hundreds, a placement sequence which requires near minimum motion of the placement head can help optimize the throughput rates.

This research develops an application using genetic algorithm (GA) to solve the component placement sequencing problem for a single headed placement machine. Six different methods were employed. The effects of two parameters which are critical to the execution of a GA were explored at different levels. The results obtained show that the one of the methods performs significantly better than the others. Also, the application developed in this research can be modified in accordance to the problems or machines seen in the industry to optimize the throughput rates.

© Copyright by Chinmaya S. Hardas
December 11, 2003
All Rights Reserved

Component Placement Sequence Optimization in Printed Circuit Board Assembly using Genetic Algorithms

by

Chinmaya S. Hardas

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Master of Science

Presented December 11, 2003

Commencement June 2004

Master of Science thesis of Chinmaya S. Hardas presented on December 11, 2003.

APPROVED:

Redacted for Privacy

Major Professor, representing Industrial Engineering

Redacted for Privacy

Head of the Department of Industrial and Manufacturing Engineering

Redacted for Privacy

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Chinmaya S. Hardas, Author

ACKNOWLEDGMENTS

I would like to take this opportunity to thank all the people who have contributed generously towards the completion of this thesis. I must, in particular, thank my advisor Dr. Toni Doolen for her support and guidance. I have learnt a lot working with her for the past two years and have gained some invaluable experience. I would also like to thank my committee members Dr. Brian Paul and Dr Dean Jensen who were instrumental in defining the scope of this research. I would especially thank Dr. Dean Jensen for helping me with understand the working of Genetic Algorithms. Finally, I would like to thank Dr. Glenn Murphy, whose inputs on this document have been most valuable.

During my stay at Oregon State University, I have had the opportunity to interact with a large number of people and have made a lot of friends. I would especially like to thank all my friends seen regularly in the graduate computer lab, BAT 042A, for all their help. The numerous discussions/arguments that we have had, academic and non-academic, have been most enriching.

Finally, I would like to thank my family for their support. They have always believed in me and encouraged me. Without their support, I would not be where I am. This thesis is dedicated to them.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF APPENDIX FIGURES

LIST OF APPENDIX TABLES

Component Placement Sequence Optimization in Printed Circuit Board Assembly using Genetic Algorithms

1. INTRODUCTION

The economic recession has led to fierce competition in the electronics industry. Electronic manufacturers need to be more efficient if they want to stay in business. They need to develop better and more innovative products that their customers want before their competitors do. In the production environment, this often translates to reducing lead times, thus reducing cycle times.

Printed circuit boards (PCB's) form an essential part of the electronics industry. PCB's continue to be the basic interconnection technique for electronic devices. PCB assembly involves the placement of many electronic components which come in different sizes, shapes and functions, on a PCB. Over the years, PCB assembly has evolved from a labor intensive activity to a highly automated activity characterized by steady innovations in the level of design and in the required manufacturing processes. Due to these continuous improvements in the electronics industry, the number of components per board has greatly increased. The number of components involved in a PCB assembly task varies significantly and can range from a few to several hundred.

One of the most important changes in the PCB assembly was the development of Surface Mount Technology (SMT). The introduction of SMT, which has nearly replaced pin-through-hole technology for PCB assembly, has enabled the mounting of an even larger number of electronic components on the board. The three most important steps in surface mount assembly (Figure 1) are solder printing, component placement and solder reflow.

Figure 1: Flowchart Showing Surface Mount Assembly Process

Given the number of components being placed on a PCB, production planning is critical. PCB assembly on the production line can be broadly classified into the following process planning problems:

- Assignment of products to production lines.
- Assignment of components to machines.
- Assignment of components to component feeder magazines for each machine.
- Sequencing of components for placement for each machine.

The first production planning process is concerned with the improvement of the overall system performance. A production line may consist of a single machine or multiple machines for each of the operations described in Figure 1. The goal here is to find the best allocation of products (boards) to different production lines according to the technological, process, and demand constraints. The second planning problem arises when there are multiple machines for a particular operation on the line. It is concerned with improving machine utilization by determining the best distribution of components to the multiple machines in the manufacturing line. Once the components have been assigned to the machines, the next step is to decide the component location on the component placement machine. The final planning problem is the component placement sequencing problem. The last two planning problems are concerned with improving cycle times on a placement machine. The cycle time on a placement machine is decided by the distance the placement head has to travel to place all the components, the number of tool changes required, etc. When looking at reducing the amount of head travel, the location of each component on the machine (feeder location) is critical so as to make sure that the components that are being placed at a

higher frequency are loaded closer to the board as compared to the others. Also, sequencing is equally critical because it helps to reduce the distance traveled by the head.

1.1 Motivation

On an SMT assembly line, the component placement process is often the bottleneck (Prasad, 1997). Thus, improving the processing times of the placement process can ensure better work flow and can help optimize the throughput. Since most of the setup, like loading components onto feeders and programming of the board, can be done externally, the setup times can be reduced drastically. Thus it is the cycle times that need to be reduced in order to optimize the process.

Most of the component placement machines have optimization software installed on the machine. But the equipment manufacturers do not provide information on how the optimization is done or what algorithm is being used because of the market competition. As a result, it is hard to judge if the machine is doing a good job of optimizing the component placement sequence. In addition, the optimization algorithm is often fixed. As a result a user cannot modify the sequence parameters to adjust for the product complexity, batch volumes, etc.

In industry, most users assume that the software on the placement machine is doing a good job of determining an optimal or near optimal placement sequence. However, research to develop an algorithm that can be modified and applied depending upon the problem scenarios exclusive to a particular industrial environment would provide practitioners with an even more flexible means of optimizing the placement operation. In addition, such research also provides a means for assessing the solutions produced by the placement machine.

1.2 Objective

This research focused on component placement sequencing optimization to help reduce the cycle time for the placement process. Genetic algorithms were used as the optimization tool. Three representation methods, along with three different parameters were compared, to find the combination that produced the least amount of head travel.

1.3 Contribution

A genetic algorithm to solve the component placement sequence problem was developed. Three different types of vector representations were compared. Six different crossover operators, each exclusive to one of the three methods, were used. The research findings show that path representation with an order-based crossover operator performs better than other methods. This representation however, takes longer to find a near optimal solution. Although higher mutation rates yield a better solution, it takes substantially longer to solve the problem. As a result moderate mutation rates are suggested.

2. LITERATURE REVIEW

The first section of this chapter describes surface mount technology and the various steps involved in the surface mount assembly process. The next section discusses the importance of component placement machine and also describes the construction and working of different types of placement machines. The following subsection has a brief description of the traveling salesman problem (TSP) and how the component placement sequence problem can be modeled as a TSP. Genetic algorithms and their working are described in the next subsection. In the final subsection of this chapter, previous researches in the field of electronic assembly systems optimization are discussed.

2.1 Surface Mount Technology Process Overview

Surface mount technology (SMT) makes it possible to produce state of the art miniaturized electronic products at reduced weight, volume and cost (Prasad, 1997). In contrast to conventional technology wherein the components were inserted through the holes on the printed circuit board (PCB), SMT is used to mount electronic components on the surface of the PCB. This deceptively simple difference has changed every aspect of electronics manufacturing.

There are various steps involved in SMT. Solder printing, component placement, and solder reflow are the most important. Solder printing is usually done by a screen/stencil printer. It is the process where the solder is applied to the stencil and the squeegees are used to force the solder paste to flow through the apertures on the stencil and onto the PCB. There are quite a few parameters which affect the quality of solder printing, e.g. solder paste viscosity, snap off distance, stencil thickness, etc. Two of the most critical parameters however, are the printing speed and pressure.

A fast print speed will cause planing of the squeegee, resulting in skips. The squeegee will not have enough time to fill each aperture, resulting in insufficient fill. Though a slow speed is generally preferred, too slow a speed will cause ragged edges or smearing in the printed solder paste. A low squeegee pressure also results in skips and ragged edges. A high pressure print pass causes smeared prints and also tends to scoop solder paste from wide apertures, causing insufficient solder fillets.

Surface mount components are placed on a PCB after deposition of solder paste. The component placement machines which are commonly referred to as pick and place machines, are used for this purpose. The pick and place machine is the most important piece of manufacturing equipment for placing components reliably and accurately enough to meet throughput requirements in a cost-effective manner. Also, the throughput of a manufacturing line is primarily determined by the pick and place machine. The majority of manufacturing defects that require rework stem from placement problems. Thus, accurate placement of components can ensure better throughput rates. The placement process is also often a bottleneck in the SMT assembly line, and faster placement process can ensure better work flow. The pick and place machine absorbs the highest capital investment, and also determines the overall economics of manufacturing (Prasad, 1997).

Once the solder printing and component placement is done, the next step in SMT assembly is solder reflow. It is the process where either two similar or two dissimilar metals or alloys are joined. The objective is to hold the component onto the board and provide a good electrical connection to complete the circuit. There are mainly two soldering processes, wave and reflow soldering. The basic difference between wave and reflow soldering lies in the source of heat and the solder. In wave soldering, the solder wave serves the dual purpose of supplying heat and solder. The source for the supply of solder is unlimited because the wave pot holds a large amount of solder relative to what is needed. In reflow soldering, solder paste is applied first. During reflow, heat is applied to melt the solder paste.

Wave soldering is the main process used for soldering component terminations in conventional through hole mount assemblies. It is also the most widely used process for soldering surface mount discrete components glued to the bottom of SMT assemblies, as in the case of double sided boards.

The reflow soldering process can again be classified under two types. IR dominant systems and convection dominant systems. The systems mainly differ in their heat sources and in their heating mechanism. A minimum of three heating zones is required for reflow soldering – a preheat zone to vaporize the volatiles from the paste, a soak zone to raise the temperature uniformly throughout the board to slightly below the reflow, and finally a reflow zone followed by rapid cooling. More than three zones are not necessary from a technical standpoint, although a higher number of zones in an oven does make it easier to develop the desired thermal profile, one of the key variables in the manufacturing process that significantly impacts product yield.

SMT assembly is used to produce PCBs of various sizes. The size of a particular PCB is decided based on its purpose and functionality. For example, a PCB to be used in a cell phone would have to be much smaller than those used in electronic equipment such as a component placement machine. The PCB used in a cell phone does not have the number or diversity of functions to perform as compared to a PCB used in a placement machine. The functionality determines the number and type of components to be placed on the PCB. As the number of components to be placed increases, the time taken for the placement process also increases.

2.2 Component Placement Machine

The component placement machine, also known as the pick and place machine, is one of the most important pieces of equipment for surface mount assembly. Apart from the fact that it constitutes about 50% of the total capital investment, the throughput of the manufacturing line is primarily determined by the pick and place machine (Prasad,

1997). There are three different types of placement machines used in PCB assembly. The construction and working of these are discussed briefly.

Multi headed component placement machines are the most widely used. These are also referred to as gantry style placement machines. The machine has a moving arm which carries the placement heads on it. The board comes in on a conveyer and is held in place, on the table, during the time the placement operation is being carried out. The machine has a feeder magazine onto which the components to be placed on the board are loaded, a tool changer where different nozzles are stored, since not all components can be placed with the same nozzle. Different size nozzles are needed, depending on the size of the component being placed.

The multi headed machines are again classified as one of two types. The first type has the board fixed and the head moving in the x-y-z directions to pick up the components from the feeders and place them on the board and traversing to the tool changer to change the nozzle as and when the need arises. The second type has the board moving in the x-y direction and the feeder magazine moving in any one direction, mostly x. The head is not fixed, but its movement is restricted. The head travels to a fixed pickup location to pick up a component and back to a fixed placement position. Since the head only moves to a fixed pickup location, the feeder magazine has to move in order to get the components at the fixed pickup location.

The second type of placement machine is the high speed ship shooter (HSCS) also known as turret type placement machines. It uses a rotating turret to hold the placement mechanism, which consists of multiple heads. The turret rotates from a fixed pickup location to a placement location. Since the pickup location is fixed, a moving feeder magazine is essential. Also the board has to be moving to facilitate proper placement. As the turret carries multiple heads, the tool change time encountered in the multi headed machines is eliminated and this makes the placement operation much faster. One

of the disadvantages with this type of machine is the fact that it cannot be used for placing high mass components, because of the moving board.

The third type of pick and place is the robotic arm placement machine. On a robotic arm placement machine, the boards are in continuous motion on the conveyer. Robotic arms are mounted along the conveyer with component feeders located along the conveyer. The movement of board on the conveyer and motion of robotic arms has to be synchronized to accomplish correct placement of components.

For any component placement machine, the type of feeding mechanism used depends on the component's packaging. The components mainly come in reels, trays, tubes or bulk. Reels are the most widely used for smaller components such as the resistors, capacitor, transistors and smaller IC's. Mechanical feeders are used to feed components in reels. Larger components such as QFP's and BGA's are normally supplied in trays. A separate machine called a matrix tray changer (MTC) is used for supplying components in trays. A few components such as PLCC's are normally supplied in tubes. Vibratory feeders are used to feed these components. Bulk components are manually fed, by placing them at a particular place on the feeder bank and programming the pick and place to pick up the component from that location.

2.3 Traveling Salesman Problem

The component placement problem being discussed in this research can be modeled as a TSP. The most prominent member of the set of combinatorial optimization problems is undoubtedly the traveling salesman problem (TSP). TSP states that: given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point. The study of the TSP has attracted research from a variety of fields such as mathematics, operations research and biology. Many practical applications can be modeled as a TSP or a variant of it (Reinelt, 1994).

For the magnitude of component placement sequencing problem being discussed, where hundreds of components are being placed on one board, it is practically impossible to find a best solution using linear programming or other mathematical modeling techniques. In the industrial environment, where time is a premium, it would not be practical to wait for the problem to be solved to find the best sequence.

A genetic algorithm (GA) can be used to find the solution in much less time. Although a GA may not find the best possible solution, it can find a near optimal solution. In the next subsection GAs are defined and their working is described in detail.

2.4 Genetic Algorithms

GA's imitate the biological phenomena where the genetic information in an individual determines its traits and fitness in a particular environment. In general a genetic algorithm has five basic components as summarized by Gen and Cheng, 2000 and Rawlins, 1991:

1. A genetic representation of solutions to the problem
2. A way to create an initial population of solutions
3. An evaluation function rating solutions in terms of their fitness
4. Genetic operators that alter the genetic composition of children during reproduction
5. Values for parameters of GAs

The first component involves choosing the right coding schema. Bit strings, list of 0's and 1's, are the most widely employed. Bit strings have been shown to be capable of usefully encoding a wide variety of information, and they have been shown to be effective representation mechanisms in unexpected domains. But for the TSP, vector representation is more suited. This is because of problems like infeasible solutions and redundant solutions being generated when using bit strings. There are mainly three

vector representations considered in connection with the TSP: adjacency, ordinal and path representation (Michalewicz, 1996).

After the representation to be used is chosen, a population of possible solutions needs to be created. Davis suggests that for research purposes, a good deal can be learned by initializing a population randomly (Davis, 1987). Moving-from a randomly created population to a well-adapted population is a good test of the algorithm. By doing this, critical features of the final solution will have been produced by the search and recombination mechanism of the algorithm, rather than the initialization process. For industrial applications, it may be beneficial to initialize with more directed methods.

To search for an optimal solution, formulation of a function which evaluates the population of solutions is required. This is also called the fitness function, since it ranks the population in accordance to its fitness as a solution. The fitness function is the most crucial part of the GA, as this is the one which decides as to how much time it is going to take to find the optimal solution.

The next step is the selection process whereby the solutions with best fitness values will be selected. Selection provides a driving force in a GA (Gen and Cheng, 2000). Typically, a lower selection pressure is indicated at the start of a search in favor of wide exploration of search space, while higher selection pressure is recommended at the end to narrow the search space. Many selection procedures are in use. A few widely used selection procedures are the tournament selection and roulette wheel selection. The tournament selection process is done by randomly picking solutions, normally two, and the one with a higher fitness value is selected to the new population. In roulette wheel selection, selection probability for each solution is based on their fitness value. The selection process is based on spinning the wheel, displaying these probabilities, the number of times equal to the population size, each time selecting a single solution for the new population.

Once the new population is created, the solutions in it are then allowed to reproduce with one another. There are mainly two reproduction processes, crossover and mutation. In the crossover process, a partitioning point is chosen and all the bits after the partition are exchanged by the two parents. In conventional GAs, the crossover operator is used as the principle operator, and the performance of the system heavily depends on it. The mutation operator, which randomly changes one bit of the solutions, is used as a background operator. It has been proven that the mutation operator can sometimes play a more important role than crossover; thus the two need to be used in accordance with the problem on hand (Gen and Cheng, 2000). Figure 2 illustrates the crossover and mutation processes. The new solutions will constitute a new population for the next generation. The process of evaluation, selection and reproduction is carried out until the whole population converges to an optimal or a near optimal solution.

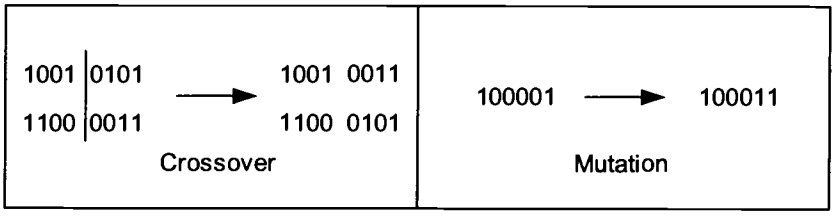


Figure 2: Crossover and Mutation processes

2.5 Previous Research in Electronic Assembly Systems Optimization

Previous research has identified three production planning problems in PCB assembly:

1. The assignment of various components to machines
2. The assignment of the components to the component feeder locations for each machine
3. The component placement sequencing for each machine

The first problem arises when there is a mix of boards. The objective is to assign these boards, and the components that go with them, to different machines in a manufacturing facility. Once assignment to machines is completed the next level of research is to focus on feeder rack assignment and component placement optimization problems i.e. 2 and 3. The remainder of this section discusses various researches previously done in the area of PCB assembly optimization and its relevance to the research being discussed here. Due to the nature of the research, which often addresses portions of the PCB assembly planning problems 1, 2, and 3, it is difficult to classify previous research. The research that has been summarized is ordered so that the scheduling problems are described first. Then the component placement and feeder assignment problems are described, moving from other placement machines to the multi-headed placement machines and finally to research which uses GAs for placement optimization.

2.5.1 PCB Scheduling Problem Research

Gunther, Gronalt and Zeller (1998), addressed the problem of sequencing PCB assembly jobs on an automated SMT placement machine. The objective was to minimize the makespan. A considerable set-up time is incurred when switching from one feeder type to another. The setup depends on the number of component feeders to be replaced in the magazine of the assembly machine. The exchange of assembly feeders is complicated by the fact that each feeder occupies a different number of magazine positions. Theoretically, the minimum makespan required for a given batch of jobs could be derived by solving the order sequencing and the component set-up problems simultaneously. However, optimal solutions are practically unattainable for problems that are realistic in size. Therefore, efficient heuristics solutions procedures were developed. These procedures exploit component commonality between PCB types.

A straightforward approach is to sequence the jobs initially and then determine the exchange of component feeders based on the job sequence obtained, before finally assigning the feeders to magazine positions. The first stage of solving the job sequencing and component setup problem involves determining a job sequence which minimizes the total processing time. Given the changeover time between two consecutive jobs, the job sequencing problem corresponds to a TSP. The job sequencing and component setup problem is similar to finding the shortest path in the TSP except that the distance between two cities is not known in advance. To overcome this deficiency, approximation scheme for estimating the changeover time between two successive jobs are suggested. The heuristic solution procedure proposed consists of two steps. The first step is a construction heuristic which schedules the jobs sequentially, choosing at each stage the job with the minimum changeover time from those available for scheduling. In the second step, the initial job sequence is iteratively improved.

Two related heuristic solution procedures for the job sequencing and the component setup problem are presented. Both heuristic solution procedures are comprised of the same modules as described earlier. They differ, however, in the organization of the computational procedure. Due to its sequential organization, the first approach is referred to as a 'sequential heuristic'. A major drawback of the sequential heuristic outlined above is that it only considers the changeover time between a pair of jobs. Clearly, the number of setups between two successive jobs does not depend merely on the immediate predecessors, but also on all of the preceding jobs. This observation led to a 'Composite heuristic'. Computational experiments were carried out to evaluate relative performance of both heuristic solution procedures. Numerical evaluations indicated that both the heuristics tend to generate near optimal solutions.

In one of the earliest works on PCB assembly optimization, Brandeau and Billington (1991) worked on operation assignment in PCB assembly. The objective was to determine an assignment of components (operations) to a set of capacitated machines, with the reminder of the components inserted manually, to minimize the total setup and processing cost for assembling all boards. These costs can be expressed in terms of time units required for setup and processing, yielding an objective equivalent to minimizing the average time required to produce each board.

The problem being analyzed arose from one faced by Hewlett-Packard in one of its PCB assembly operations. The process was not fully automated because a wide mix of boards was produced, and the volume of production did not justify automation. Component insertion in the hand load cell could be performed manually or by a semi-automated machine. Both setup and processing are faster, and thus cheaper, on a machine. However, the machines had a limited capacity for holding different components, and only a limited number of machines were available. As a result of these constraints, it was necessary to determine which components to assign to each process.

Brandeau and Billington (1991) have formally defined the problem as a mixed integer program. Two different heuristics for the case of single machine problem are proposed. The first solution approach, referred to as 'Stingy Component' algorithm, starts by assigning all components to the machine and then sequentially removing components which cause the smallest cost increase, until machine capacity constraint is met. All components not assigned to the machine are assigned to the manual process. The second approach, referred to as the 'Greedy Board' algorithm, starts by assigning all boards to the manual process, and then assigns entire boards to the machine, one by one, to maximize incremental boards produced per incremental component bin used. For the multiple machine problem, the single machine heuristics

are applied to each machine sequentially. Then, as before, once an initial assignment of components to machines is made, post processing steps are carried out to determine if some boards can be made more cheaply by reassigning them. For each single machine heuristic, two different multiple machine heuristics are developed. These heuristics differ in the extent to which part assignment to more than one machine is allowed. In first case, a component, if assigned to a machine, cannot be assigned to any subsequent machines, while in the second case, potential assignment of the same component to various machines is considered.

These heuristics were tested with real world problems at Hewlett-Packard. The results showed that the Greedy Board algorithms provided slightly better solutions. Stingy component algorithms tend to provide better solutions when setup costs are low, while the Greedy Board algorithms may provide better results when setup costs are higher. While each of the heuristics can be arbitrarily bad, on average, the algorithms performed quite well.

Crama, Flippo, Klundert and Spieksma (1996) investigated the component retrieval problem in printed circuit board assembly. Decisions involved in this problem are concerned with the specification of the order in which the components are to be placed on the board, as well as the assignment of component types to the feeder slots of the placement machine. If some component types are assigned to multiple feeder slots, then the additional problem emerges of selecting, for each placement on the board, the feeder slot from which the required component type is to be retrieved. In this research, the component retrieval problem is considered for turret type placement machines.

Crama et. al. (1996), reformulated this component retrieval problem as a longest path problem in a PERT/CPM network with design aspects. As an alternative interpretation, the component retrieval problem can be viewed as a shortest path

problem with side constraints. Crama et. al. (1996) define the formalization of the CRP graphs and problems with the help of four definitions. Those are definition for CRP graph, Selection, Selection induced subgraph and the CRP problem. The definitions reveal that CRP is basically a PERT/CPM network problem with design aspects. Crama et. al. (1996) present the algorithm developed for CRP and have also compared it with the forward dynamic programming scheme proposed by Bard, Clayton and Feo (1994), to explain why the latter approach cannot possibly lead to a correct algorithm for CRP. Finally, Crama et. al. (1996) have sharply delineated the complexity of the problem by proving that it becomes NP-hard when additional structure on the activity durations in the PERT/CPM network is absent.

Again, Crama, Flippo, Klundert and Spieksma (1997) worked to solve the case of assembly planning with multiple board types and multiple machines. The planning problem faced is how to assemble boards of different types using a single line of placement machines. The multiplicity of the boards adds significantly to the complexity of the problem, which is already very hard to solve in the case of a single board type case. Turret style placement machine were used for the investigations.

In summary, the problem was to find:

- For each machine, a feeder rack assignment,
- For each board type, a component placement sequence on each machine, such that for each PCB of that type, the sequences form a partitioning of the components required by the PCB,
- For each pair consisting of a machine and a board type, a component retrieval plan.

The planning procedure was divided into two phases: Phase 1 determined the feeder rack assignment for each machine and Phase 2 produced, for each pair consisting of a machine and a board type, a component placement sequence and a component retrieval plan, given the feeder rack assignment of Phase 1. Phase 1 consisted of five steps.

1. Determine which component type will have two feeders in the flowshop.
2. Decide, for each feeder, which locations it serves on each board type.
3. Construct an arbitrary feeder rack assignment.
4. Estimate the makespan for each board type on each machine, given the current feeder rack assignment.
5. If the stopping criterion is satisfied, exit. Else, improve the feeder rack assignment using local search and go to Step 4.

Crama et. al. have developed an algorithm for Steps 1 and 2 and another algorithm for Step 4. The optimization part of the feeder rack assignment is done by using two heuristics alternatively. One heuristic tries to exchange between two machines a pair of clusters, together with the corresponding feeders, to better balance the workload. The other heuristic reoptimizes the feeder rack assignment for a single machine. Together these two heuristics deliver better solutions faster than other approaches that have been tested by Crama et. al.

Phase 2 consists of three steps.

1. Determine, for each machine – board type combination, a component placement sequence.
2. Determine, for each machine – board type combination, a component retrieval plan.

3. Improve the component placement sequencing using local search. If no improvements are found, stop, else go to Step 2.

The improvement of the placement sequence is done by TSP-like local search techniques.

2.5.2 Component Placement and Feeder Assignment Research

Burke, Cowling and Keuthen (1999) proposed a few new models and heuristics for component placement in PCB assembly. The component placement sequencing problem is modeled as a TSP by considering the placement locations as the cities of the TSP and define distances by the time of distance traveled between two successive placements.

More formally the component placement sequencing problem is modeled as follows. Let $\{c_1, c_2, \ldots, c_n\}$ be the finite set of placement locations, d_{ij} denote the distance in terms of time or distance traveled between placements c_i and c_j and $\sigma : \{1,2,\ldots,n\} \rightarrow \{1,2,\ldots,n\}$ be a one tone mapping which orders the set of placements. Then the component placement sequencing problem is equivalent to finding σ so as to

$$\min_{\sigma} \sum_{i=1}^{n-1} d_{\sigma(i),\sigma(i+1)} + d_{\sigma(n),\sigma(1)}.$$

The assignment of component types to feeder slots is of substantial importance. Burke, Cowling and Keuthen (1999) have introduced more general models and have considered a single headed component placement machine, placement locations, each with its own component, $\{c_1, c_2, \ldots, c_n\}$ and $\{s_1, s_2, \ldots, s_f\}$ available feeder slots. Let $\rho : \{c_1, c_2, \ldots, c_n\} \rightarrow \{s_1, s_2, \ldots, s_f\}$ be a mapping assigning each placement location to

a feeder slot and d_{ij}^{ρ} be the time taken between finishing placement of component c_i and c_j for component assignment ρ. Thus the problem is now a two level optimization problem, to find σ and ρ so that:

$$\min_{\sigma,\rho} \sum_{i=1}^{n-1} d^{\rho}_{\sigma(i),\sigma(i+1)} + d^{\rho}_{\sigma(n),\sigma(1)}.$$

Burke, Cowling and Keuthen (1999) go on to discuss the case of multiple head machines and also described a model incorporating tool changes. Heuristics to tackle the minimum weight hypertour problem, arising from the multi headed placement machines are also proposed. The heuristics developed take their inspiration from two well known heuristics for the TSP, nearest neighbor tour construction and the local search algorithm *k-opt*, which were modified to be suitable for the problem on hand.

Lee, Lee and Park (1999), employed a hierarchical method to improve the productivity of a multi-headed surface mount machine. The problem of minimizing the assembly time of multi-head surface mount machines was decomposed into a hierarchy of related subproblems. Since all subproblems in the hierarchy were known to be of combinatorial nature and computationally intractable, heuristics based on dynamic programming and nearest neighbor TSP technique were developed.

The following outline contains the subproblems constituting the hierarchy in the order in which they were solved:

1. Construction of reel groups
    1.1. Determining which nozzle is used for each reel.
    1.2. Determining which head is used for each reel.

1.3. Grouping the reels into subsets of size (at most) N (N denotes the total number of heads at the machine), which are called reel groups.

2. Assignment of reel groups

   2.1. Determining in which order the reel groups are assigned.

   2.2. Assigning the reel groups to slots on feeder racks.

3. Sequencing of the pick and place movements

   3.1. Determining in which order nozzles are changed.

   3.2. Grouping the components into subsets of size (at most) N, which are called component groups.

   3.3. Determining the sequence of the component groups.

Assumptions to the problem were as follows:

- The order in which the heads operate is predetermined as follows: the first head picks up first, then the second head, and so forth until all heads are loaded. Then the first head places first, the second places second and so on until the cycle is complete.
- Components of one type cannot be carried by two or more reels.

The performance of the hierarchical method was compared with the results from the heuristic algorithm based on a greedy algorithm. Results showed that the hierarchical algorithm works better as the number of reels and the number of mounting positions increased. The performance also improved rapidly as the number of heads increased.

Su and Srihari (1996) used artificial neural networks (ANN) for placement sequence identification. A decision support system was designed and developed for the

placement process associated with multiple batches of surface mount PCB's. The feeder locations were established for multiple batches, and the component placement sequence was identified while considering the need to minimize tooling and nozzle changes and the actual distance traveled by the placement head.

Feeder location optimization was based on the "largest candidate rule" and the slot numbers of the PCB's center on each workstation. These two factors determined a component's feeder location. The location of the feeder was arranged and mounted according to the frequency of use of components. The more frequently a component was used, the closer that components feeder should be to the center of the PCB. The dimensions of the PCB and the number of feeder slots available for the placement head are used to determine the center feeder slot. This approach reduces the traveling distance of the component head.

The difference in mass and geometry of the components to be placed on a board dictates the use of specific tooling sets for component placement. The tooling and nozzle arrangement module minimizes the tooling and nozzle changeover times associated with placement operation. The methods considered for minimizing tooling and nozzle change are based on three rules, namely, "placement head configuration", "tooling and nozzle configuration" and "surface mount component characteristics". The "placement head configuration" rule determines the specific placement head for all the components to be placed. Next, the "tooling and nozzle configuration" rule sorts the components to be placed by the tool and nozzle identification number. To complete the task, the "surface mount characteristic" rule is applied. First all unleaded components are placed. Next the leaded components are placed. In each case the smaller components are placed first and then the larger components are placed.

For the placement sequence optimization, Su and Srihari (1996) proposed to use the Hopfield and Tank representation. Hopfield and Tank proposed an energy function that will find the tour which has the shortest length among many admissible solutions. Using this representation, the TSP amounts to a minimization problem with constraints. Thus the component placement sequence module is based on the unsupervised learning through an ANN using the Hopfield-Tank model to find the minimal traveling path. The number of components using the same tooling and nozzle configuration could be large. To reduce computer time, the components using the same tooling and nozzle are clustered together by the clustering algorithm. After clustering data, the Hopfield-Tank ANN model is executed for each group in succession. The saturation value of the ANN is used to decide if an acceptable solution is reached or not. The final traveling path is indicated by a saturation value of 1.

A novel tabu search approach to find the best placement sequence and magazine assignment in dynamic robotic assembly was proposed by Su, Ho and Fu (1998). Two types of robot assembly problems have been characterized on the basis of different robot motions:

- Fixed robot motion between fixed pick and place (FPP) points, and
- Robot motion with dynamic pick and place (DPP) points.

In the FPP motion model, the feeder magazine moves horizontally along the X-axis and the robots moves only along a Y-axis. The assembly board (X-Y table) moves freely in any direction, allowing the feeder magazine to move required components to the fixed pickup points. When the assembly board moves to a fixed placement location, the robot picks up and places the components along these two fixed points. In the DPP model, the robot moves along the X and Y axes, and the pickup and placement points are dynamically allocated. The assembly board and feeder magazine move only along the X-axis. This study was done on a DPP robot.

Tabu search, a metaheuristic approach for solving combinatorial optimization problems, is an adaptive procedure that can be superimposed on many other methods to prevent them from being trapped at a local optimal solution. The basic components of tabu search are the configuration, move mechanism, objective function, tabu list, tabu restrictions and aspiration criteria. The configuration represents the feasible solutions. Move mechanism generates the move set. Objective function evaluates all solutions. Tabu list remembers where it has searched recently and does not revisit these solutions. Tabu restrictions command that the selected current move is forbidden for the next few moves. Aspiration criteria follow that if an "enough good" solution generated is found so far, then the tabu status of this move is overridden.

Simulation results demonstrated that the tabu search approach is more efficient in comparison to the dynamic pick and place approach. Also, results presented confirm that the larger the numbers of placements, the better the performance.

2.5.3 Component Placement Optimization Research using GAs

Leu, Wong and Ji (1993), used the genetic algorithm approach to solve printed circuit board assembly planning problems. The developed genetic algorithm finds the sequence of component placement/insertion and arrangement of feeders simultaneously, for three main types of assembly machines. Three different sequencing problems are defined for each of the placement machines described. First is the traveling salesman problem for head fixed and board moving type of gantry machine. Second is the pick and place problem for the head moving and board fixed type of gantry machine. The final problem is called the moving board with time delay problem and is for the turret type placement machine.

Four genetic operators were used: crossover operator, inversion operator, rotation operator and mutation operator. A two-link genetic algorithm was devised for dealing with the planning problems that involve determining both the sequence of component placement/insertion and the assignment of components to feeders. A sample PCB with 200 components and 10 different component types are used as an example in solving the pick and place problem. An improvement of 11.84% in comparison to the original solution is observed by the end of 6000 iterations.

Khoo and Ng (1998) developed a genetic algorithm-based planning system for PCB component placement. Due to the variation in component size, quantity and shape, two main issues concerning PCB component placement planning need to be tackled:

- Component placement priority and
- Optimal component placement sequence.

Component placement priority arises when certain components need to be mounted before others or towards the end of an assembly process. Since not all the PCB components are of the same shape and size, taller components if assembled earlier may result in insertion difficulties. Thus four basic insertion rules for PCB component placement have been identified:

- Insert smaller PCB components before larger components to avoid interference
- Place, in one pass, all the components of the same type and value.
- Prepare components with identical size and shape but different electrical value with other components of non-similar size and shape.
- Choose a near-optimal component placement sequence to minimize machine bed's movement.

Khoo and Ng (1998) chose to use path representation which expresses a string (chromosome) comprising the identification numbers of the "cities". Order-based cross-over operator which is a modified version of the classical cross-over operator was employed to deal with the chromosomes coded by path representation. This method requires a section of the parent's chromosomes to be preserved during cross-over operation. Using path representation and the order based cross-over operator, the "genetic traits" from the parents are preserved and passed on to their children. A modified mutation operator known as inversion was employed to perform the much needed search for alternative solutions. A repair algorithm was incorporated to repair the offspring should the arrangement of genes in a chromosome become disorganized. Once this was done, the fitness of the offspring was evaluated using objective function.

System validation was then carried out. Results showed that once the system started producing a converging population, the genetic operators i.e. crossover and mutation, became inefficient. Further iterations to generate new population could not improve the search. The path finder was then modified to include swap mutation. This inclusion improved the system performance further. This work demonstrated the possibility of using GAs for planning PCB component placement sequences.

Wang, Nelson and Tirpak (1999) used genetic algorithms to optimize the feeder slot assignment problem for a high speed parallel, multi station SMT placement machine. The machine used for this research was a Fuji QP-122. The machine consisted of two major subsystems: a pallet circulating system (conveyer system) that transfers and indexes PCBs to each placing station and a placing station that is responsible for placing the components onto the PCB. Each placing station is composed of a vision system, fixed multi feeder unit, and a placement head with a single nozzle.

Wang, Nelson and Tirpak (1999) made a few assumptions to simply the model. Some of the important assumptions were:

- All boards of a given type will be produced consecutively.
- The quantity of components on each reel is sufficient to produce the required quantities of all the boards.
- The components on each board can be placed in any given order.
- Repeating same components of the same type on the feeder carriage is not allowed.

The goal of Wang, Nelson and Tirpak's research was to minimize the total assembly time of the entire machine for the given product. Since each station works concurrently, the total assembly time will be the maximum placement time of all the stations. Thus,

T_{total} = max[$PlaceTime_l$], for all stations less than or equal to s (total number of stations).

$PlaceTime_l$ can be represented using the following equation:

$$PlaceTime_l = \Sigma^{m}_{i=1} [T_i * P_{ij} * A_{jl}]$$

where, T_i is the placement time of placement i. P_{ij} will be 1 if and only if placement i uses a component of type j. A_{jl} will be 1 if and only if component type j is located at station l.

Four crossover operators (valid crossover, PMX, cycle and ordered crossover) and four selection methods (Roulette wheel, elitist model, stochastic tournament and ergodic matching selection) were tested using a production scenario that contained more than one hundred unique type of components. Two sets of experiments with different operator probabilities were conducted. The first set used a probability of reproduction of 40% and the probability of mutation and crossover were set at 5% and

55% respectively. The second set also used the same probability of reproduction of 40%, while the probability of mutation and crossover were set at 40% and 20% respectively. For both sets of experiments, the best combination was the PMX crossover operator using elitist selection method. GAs were also compared with other optimization methods such as human experts, optimization software provided by the vendors and rule-based systems. GAs performed better than all the other methods mentioned.

Jeevan, Parthiban, Seetharamu, Azid and Quadir (2002) also looked at optimization of PCB component placement using genetic algorithms. The placement machine used for this research was a four headed gantry type component placement machine.

The fundamental idea of the model was to consider the sequencing problem for the multi-headed placement machine as a generalized TSP. The number of component placement location and the component pick up locations were equal. The number of components picked ranges from 1 to 4. To obtain a valid sequence tour, the pick up process must come before the placement process. Otherwise the move is considered illegal as it is illogical for an empty head to move to a placement location. Thus the fitness function is given by the following conditions:

$1 < H \leq 4$

$1 < P \leq 4$

where, H = number of component pick-up and

P = number of component placed

The following rule is observed to avoid illogical moves:

$P < H$

By satisfying these three conditions, the shortest distance sequence is obtained.

The tool change operation is unavoidable in any multi-headed component placement machine. However, in most cases, the nozzle is able to pick up more than one type of

component. Since the tool change process is time consuming, it is therefore preferred to exhaust all components that can be placed by a certain nozzle before the nozzle is changed. Although this requires the problem to be decomposed into separate TSPs, the method proposed by Jeevan et. al. avoids solving TSPs individually but combines all TSPs into a single TSP sequence. Therefore the following extra condition was added:

$S_1 < X < S_2$

where S_1 and S_2 represent two different tool sizes for the nozzle and X represents the component size. Thus the condition indicates that if a component is bigger than tool size S_1 and smaller than tool size S_2, the component will be picked and placed by tool of size S2. The remaining will be picked and placed by tool of size S1. This condition can be easily expanded to any number of tool sizes.

Parametric studies on GA were carried out before the GA was used to optimize an actual placement problem. This was done because of the numerous parameters which govern GA performance. All GA parametric studies were carried out for a placement problem involving 12 components to be placed on a PCB by a triple head machine without tool change. The study indicated that the increment of order based crossover and uniform crossover produced small reductions to the total distance covered whereas change in other types of crossover brings little or no effect. For the mutation operation, the inverse and swap mutation played a vital role. The results from the GA parametric studies carried out were used for solving the actual component placement problems.

2.6 Summary

Previous research has identified three production planning problems in PCB assembly:

1. The assignment of various components to machines.
    a. Gunther, Gronalt and Zeller (1998)
    b. Brandeau and Billington (1991)

2. The assignment of the components to the component feeder locations.
    a. Crama, Flippo, Klundert and Spieksma (1996)
3. The component placement sequencing for each machine.
    a. Bruke, Cowling and Keuthen (1999)
    b. Lee, Lee and Park (1999)
    c. Su and Srihari (1996)
    d. Leu, Wong and Ji (1993)
    e. Khoo and Ng (1998)
    f. Jeevan, Parthiban, Seetharamu, Azid and Quadir (2002)

In this research, the focus is on problem 3 – component placement sequence optimization. While research has been conducted in this field previously, the approach used has been different. Researchers have employed integer programming, developed heuristics and used evolutionary programming to solve the problem. A few researchers have employed GAs, but only certain representation schemes have been tested. This research further expands the existing body of knowledge by applying GAs to a fixed component placement sequencing problem and by focusing on comparing and contrasting different vector representation schemes and reproduction operators.

3. METHODOLOGY

This chapter describes the methodology followed to develop the genetic algorithms. In the first subsection, the mathematical model of the problem is defined. The second subsection discusses the reasons for choosing genetic algorithms and talks in detail about the different variations compared. The implementation of these various algorithms is discussed in the third subsection.

3.1 Mathematical Model

The objective of this research was to optimize the component placement sequencing i.e. to minimize the distance traveled by the placement head. Thus the problem can be defined in mathematical terms as

Minimize:

$$\sum_i \sum_j d_{ij} X_{ij}$$

where, $X_{ij} = 1$, if position j follows position i

$X_{ij} = 0$, otherwise.

and d_{ij} is the distance between position i and position j.

Subject to the following constraints:

- All components must be placed.
- The placement head visits each placement location only once. Multiple placement of a particular component is not allowed.
- If there are components being placed on top of other components, components at a lower level are placed before components at a higher level.

The problem being defined was for a single headed placement machine. The complexity of the optimization problem increases drastically for a dual-headed

placement machine. For this research, only a single headed placement machine was considered.

3.2 Genetic Algorithms

Genetic algorithms (GAs) were used for finding the near optimal placement sequence. Because of the magnitude of the problem being discussed (hundreds of components are placed on a single circuit board), it is difficult to find an optimal solution using linear programming. Genetic algorithms can be used to find a near optimal solution in significantly less time. As discussed previously, GAs are basically made up of five components. These five components are each discussed in greater detail.

3.2.1 Representation Scheme

Encoding a solution of the problem into a chromosome is a key issue when using GAs. Various encoding methods have been created for particular problems to provide efficient implementation of the GAs. According to the kind of symbol used for the bits of the gene, the encoding methods can be classified as:

- Binary encoding
- Real number encoding
- Integer or literal permutation encoding
- General data structure encoding

Binary encoding is the most common encoding technique used because it is simple to create and manipulate. Just about anything can be encoded using binary encoding, so one point crossover and mutation can be applied without modification to a wide range of problems (Davis, 1991). But for many problems in the industrial engineering world it is nearly impossible to represent solutions with binary encoding. The other types of representation schemes are better suited for problems in industrial engineering. Real

number encoding is best used for function optimization problems. It has been widely confirmed that real number encoding performs better than binary or gray encoding for function optimizations and constrained optimizations. Integer or literal permutation encoding is best used for combinatorial optimization problems. Since the essence of combinatorial optimization problems is the search for a best permutation or combination of items subject to constraints, literal permutation encoding can be the best way to this type of problem. For more complex real world problems, an appropriate data structure is suggested as the bits of a gene, to capture the nature of the problem (Gen and Cheng, 2000).

In this research, the component placement optimization problem has been modeled as a TSP. TSP is one of the most prominent members of the set of combinatorial optimization problems. Integer or vector representation is the most suited representation scheme for solving a TSP. There are three vector representations in connection with the TSP (Michalewicz, 1996). These three representations are described in detail.

3.2.1.1 Path Representation

The path representation is perhaps the most natural representation of a tour. For example, a tour

5-2-3-4-9-7-1-8-6

is simply represented as

(5 2 3 4 9 7 1 8 6).

3.2.1.2 Adjacency Representation

The adjacency representation also represents a tour as a list of n cities. The city j is listed in the position i if and only if the tour leads from city i to city j. For example, the vector

(2 4 8 3 9 7 1 5 6)

represents the following tour:

1-2-4-3-8-5-9-6-7.

Disintegrating the vector into parts and then combining the parts appropriately leads to the tour it represents. For our example, the vector can be disintegrated into:

1-2

2-4

3-8

4-3

5-9

6-7

7-1

8-5

9-6

and then combining these parts leads to the following tour:

1-2-4-3-8-5-9-6-7.

Each tour has only one adjacency list representation. However, some adjacency lists can represent illegal tours. For example, the vector

(2 4 8 1 9 3 5 7 6)

leads to 1-2-4-1, i.e. a tour with a premature cycle.

3.2.1.3 Ordinal Representation

The idea behind ordinal representation is as follows. There is some ordered list of cities C, which serve as a reference point for lists in ordinal representations. Assume, for example, that an ordered list is simply:

C = (1 2 3 4 5 6 7 8 9).

A tour

1-2-4-3-8-5-9-6-7

is represented as a list l of references,

l = (1 1 2 1 4 1 3 1 1),

and should be interpreted as follows:

- The first number on the list l is 1, so take the first city from the list C as the first city of the tour, and remove it from C. The partial tour is 1.
- The next number on the list l is also 1, so take the first city from the current list C as the next city of the tour and remove from C. The partial tour is 1-2.
- The next number on the list l is 2, so take the second city from the current list C as the next city of the tour and remove it from C. The partial tour is 1-2-4.
- Continuing with the same steps, final tour is 1-2-4-3-8-5-9-6-7.

In this research, path, adjacency and ordinal vector representations were used to represent the solution for the component placement sequencing problem.

3.2.2 Population Initialization

The initial population for the sequencing problem was generated at random. Davis (1987) suggested that for research purposes, a good deal can be learnt by initializing a population randomly. By generating the population at random, critical features of the final solution will have been produced by the search and recombination mechanism of the algorithm, rather than the initialization process.

3.2.3 Evaluation Function and Selection

The evaluation function for the component placement sequencing problem is the total distance traveled by the head. For each of the tours being evaluated, the total distance that the head travels, in visiting each of the placement location in the sequence given by the tour, is calculated. Parent selection techniques were employed to give more reproductive chance, on the whole, to those population members that are the most fit. Four commonly used parent selection techniques are roulette wheel, tournament, elitism, and ranking and scaling.

Roulette wheel selection: In fitness proportional selection, the expected value of an individual (i.e. the expected number of times an individual will be selected to reproduce) is that individual's fitness value divided by the average fitness of the population. The most common method for implementing this is roulette wheel selection. In roulette wheel selection each individual is assigned a slice of a circular roulette wheel, the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation.

Tournament selection: This method randomly chooses a set of individuals and picks out the best individual for reproduction. The number of individuals in a set is called the tournament size. A common tournament size is 2 and this is called a binary tournament. A random number r is then generated between 0 and 1. If $r < k$, where k is a parameter, then the fitter of the two individuals is selected to be a parent, otherwise the less fit individual is selected. The two individuals are then returned to the original population and can be selected again.

Elitism: Elitism is an addition to many selection methods that forces the GA to retain some number of the best individuals at each generation. Such individuals can be lost if they are not selected to reproduce or if they are destroyed by crossover or mutation.

Ranking and scaling: The purpose of ranking and scaling method is to prevent quick convergence. Here, the individuals in the population are ranked according to fitness, and the expected value of each individual depends on its rank rather than on its absolute fitness. Ranking avoids giving the far largest share of offspring to a small group of highly fit individuals, and thus reduces the selection pressure when fitness variance is high.

These four techniques are just a few selection techniques used. In this research, Roulette wheel selection technique was used for parent selection. The reason for choosing roulette wheel selection was that it is the best known selection type (Gen and Cheng, 2000). The other reason for using roulette wheel selection was to reduce the number of factors. The scope of this research was to compare the different representation schemes and the crossover and mutation operators that go along with them. Having selection type as a factor would have increased the problem size for this research.

3.2.4 Genetic Operators

Using the selection techniques listed in the previous subsection, the fitter individuals from the parent population were selected. These individuals reproduce to form new members, called offspring, to be included in the next generation. There are mainly two reproduction operators: the crossover and the mutation operator. The classical crossover and mutation operators have already been presented in the previous chapter with the aid of examples using binary representation. For the vector representation being used in this research, the classical operators do not work well. More often than not the classical operators lead to illegal tours. Michalewicz (1996) has presented different types of crossover operators for each vector representation discussed earlier. These are described in detail in the following paragraphs.

3.2.4.1 Path Representation

Partially mapped crossover (PMX) builds an offspring by choosing a subsequence of a tour from one parent and preserving the order and position of as many cities as possible from the other parent. A subsequence of a tour is selected by choosing two random cut points, which serve as boundaries for swapping operations. For example, the two parents (with two cut points marked by '|')

$P_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$ and

$P_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$

would produce offspring in the following way. First, the segments between cut points are swapped (the symbol 'x' can be interpreted as at present unknown)

O_1 = (x x x | 1 8 7 6 | x x) and

O_2 = (x x x | 4 5 6 7 | x x).

This swap also defines a series of mappings:

$1 \leftrightarrow 4$, $8 \leftrightarrow 5$, $7 \leftrightarrow 6$, and $6 \leftrightarrow 7$.

Replace further cities (from the original parents), for which there is no conflict:

O_1 = (x 2 3 | 1 8 7 6 | x 9) and

O_2 = (x x 2 | 4 5 6 7 | 9 3).

Finally, the first x in the offspring O_1 (which should be 1, but there was a conflict) is replaced by 4, because of the mapping $1 \leftrightarrow 4$. Similarly, the second x in the offspring O_1 is replaced by 5 and the x's in the offspring O_2 are replaced by 1 and 8. The offspring are:

O_1 = (4 2 3 | 1 8 7 6 | 5 9) and

O_2 = (1 8 2 | 4 5 6 7 | 9 3).

Order crossover (OX) builds offspring by choosing a subsequence of a tour from one parent and preserving the relative order of the cities from the other parent. For example, two parents (with two cut points marked by '|'):

P_1 = (1 2 3 | 4 5 6 7 | 8 9) and

P_2 = (4 5 2 | 1 8 7 6 | 9 3)

would produce offspring by first copying the segments between the cut points into the offspring:

O_1 = (x x x | 4 5 6 7 | x x) and

O_2 = (x x x | 1 8 7 6 | x x).

Next, starting from the second cut point of one parent, the genes from the other parent are copied in the same order, omitting symbols already present. Reaching the end of the string, continue from the first place of the string. The sequence of the genes in the second parent (from the second cut point) is:

9-3-4-5-2-1-8-7-6.

After removal of genes 4, 5, 6 and 7, which are already in the first offspring, the remaining genes are:

9-3-2-1-8.

This sequence is placed in the first offspring (starting from the second cut point):

O_1 = (2 1 8 | 4 5 6 7 | 9 3) and similarly the other offspring is:

O_2 = (3 4 5 | 1 8 7 6 | 9 2).

Cycle crossover (CX): Cycle crossover builds an offspring in such a way that each gene (and its position) comes from one of the parents. For example, two parents

P_1 = (1 2 3 4 5 6 7 8 9) and

P_2 = (4 5 2 1 8 7 6 9 3)

would produce the first offspring by taking the first gene from the first parent:

O_1 = (1 x x x x x x x x).

Since every gene in the offspring should be taken from one of its parents (from the same position), the next gene to be considered must be gene 4, as the gene from the parent P_2 just below the selected gene 1. In P_1 this gene is at position '4', thus

O_1 = (1 x x 4 x x x x x).

This, in turn, implies gene 8, as the gene from parent P_2 just below the selected gene 4.

Thus

O_1 = (1 x x 4 x x x 8 x).

Following this rule, the next genes to be included in the first offspring are 3 and 2. However, the selection of gene 2 requires selection of gene 1, which is already on the list. Thus a cycle has been completed

O_1 = (1 2 3 4 x x x 8 x).

The remaining genes are filled from the other parent:

O_1 = (1 2 3 4 7 6 9 8 5).

Similarly,

O_2 = (4 1 2 8 5 6 7 3 9).

3.2.4.2 Adjacency representation

Alternating edges crossover builds an offspring by choosing (at random) an edge from the first parent, then selects an appropriate edge from the second parent, etc. The operator extends the tour by choosing edges from alternating parents. If the new edge (from one of the parents) introduces a cycle into the current (still partial) tour, the operator selects a random edge from the remaining edges which do not introduce cycles. For example, the first offspring from the two parents:

P_1 = (2 3 8 7 9 1 4 5 6) and

P_2 = (7 5 1 6 9 2 8 4 3)

might be

O_1 = (2 5 8 7 9 1 6 4 3),

where the process started from the edge (1,2) from the parent P_1, and the only random edge introduced during the process of alternating edges was (7,6) instead of (7,8), which would have introduced a premature cycle.

Heuristic crossover builds an offspring by choosing a random gene as the starting point for the offspring's tour. Then it compares the two edges (from both parents) leaving this gene and selects the better (shorter) edge. The gene on the other end of the selected edge serves as a starting point in selecting the shorter of the two edges leaving this gene. If, at some stage, a new edge would introduce a cycle into the partial tour,

then the tour is extended by a random edge from the remaining edges which does not introduce cycles.

3.2.4.3 Ordinal representation

The main advantage of ordinal representation is that classical crossover operator works. Any two tours in the ordinal representation, cut after some position and crossed together, would produce two offspring, each of them being a legal tour. For example the two parents

P_1 = (1 1 2 1 | 4 1 3 1 1) and

P_2 = (5 1 5 5 | 5 3 3 2 1)

which correspond to the tours

1-2-4-3-8-5-9-6-7 and

5-1-7-8-9-4-6-3-2

would produce the following offspring:

O1 = (1 1 2 1 5 3 3 2 1) and

O2 = (5 1 5 5 4 1 3 1 1)

corresponding to tours:

1-2-4-3-9-7-8-6-5 and

5-1-7-8-6-2-9-3-4.

As is the case with crossover, classical mutation operator does not perform well with vector representation. In this research, the classical crossover operator has been modified so that it meets the basic goal of mutation (to induce random variation) and also works with the different vector representations discussed earlier. They are described in detail in the following paragraphs.

For path representation, first, randomly generate a number between 1 and n, where n is the number of components to be placed. Replace the gene to be mutated with the randomly generated number, say k. Now, in the tour, find the position where k is, and replace k with the gene that was chosen for mutation. For example, consider a tour

1-2-3-4-5-6-7-8-9

and say the bit to be mutated is '5'. A random number between 1 and 9 is generated. Let the random number generated be 8(= k). In order to perform mutation, just swap the two genes. Thus after mutation, the tour is

1-2-3-4-8-6-7-5-9.

For adjacency representation, the same mutation operator, as described for the path representation is used.

The mutation operator for ordinal representation is similar to classical mutation operator. In classical mutation, the chosen bit changes its value from 0 to 1 or vice versa. In ordinal representation, the chosen bit changes its value to a randomly generated number which lies between 1 and (n-i+1), where n is the total number of components to be placed and i is the position of the current bit chosen for mutation. There is no swapping of bits as is the case with the other representations.

3.3 Implementation

MATLAB and Visual Basic were considered for coding the algorithms. Considering the mathematical complexity of the algorithm, MATLAB 6.5 was chosen. MATLAB has a number of built in mathematical functions that were useful in coding the algorithms. MATLAB also has a number of existing functions for matrix/array manipulations. These functions do not exist in Visual Basic. The implementation of the algorithms in MATLAB is discussed in detail below.

3.3.1 Path Representation

Figure 3 shows the flowchart describing the steps for path representation. Initially the population was randomly generated using the randperm function, which generates random permutations of the number of components to be placed. Each of these

randomly generated tours was evaluated to give its distance. The evaluation function first calculates the distance between two consecutive positions in the tour, and then sums all these distances to give the total distance of the tour.

Probability of selection for each of these tours was calculated next. As previously mentioned, roulette wheel selection was used for this research. In roulette wheel selection the probability of selection is calculated by dividing the evaluation of each tour by the total evaluation (sum of all evaluations). Cumulative probabilities were then calculated by summing the probabilities of selection. Random numbers, between 0 and 1, were then generated. The next highest number in the list of cumulative probabilities, exceeding the random number generated, is the one selected to be in the next generation. The conventional roulette wheel selection is used for maximization problems, as the individuals with higher evaluation have a greater chance of being selected. In this research the objective was to minimize the distance. Thus the selection technique needed to be modified. This was done by calculating the probability of selection by dividing the "reciprocal" of evaluation of each tour by the total evaluation. The rest of the procedure was same as described above.

The next step was to choose individuals for crossover. This was done randomly. A random number, between 0 and 1, was generated for each individual in the population. If the random number was below the probability of crossover, the individual corresponding to the random number is chosen for crossover. Also, for crossover, an even number of individuals are needed. Thus, the next step was to check if the number of chosen individuals was even or not. If it is not even, an extra individual needs to be added or an existing individual needs to be deleted. This was again done randomly. A random number was generated. If the random number was below 0.5, an existing individual from the pool of crossover population was deleted. If the random number was above 0.5, an extra individual was randomly chosen from the original population added to the crossover population.

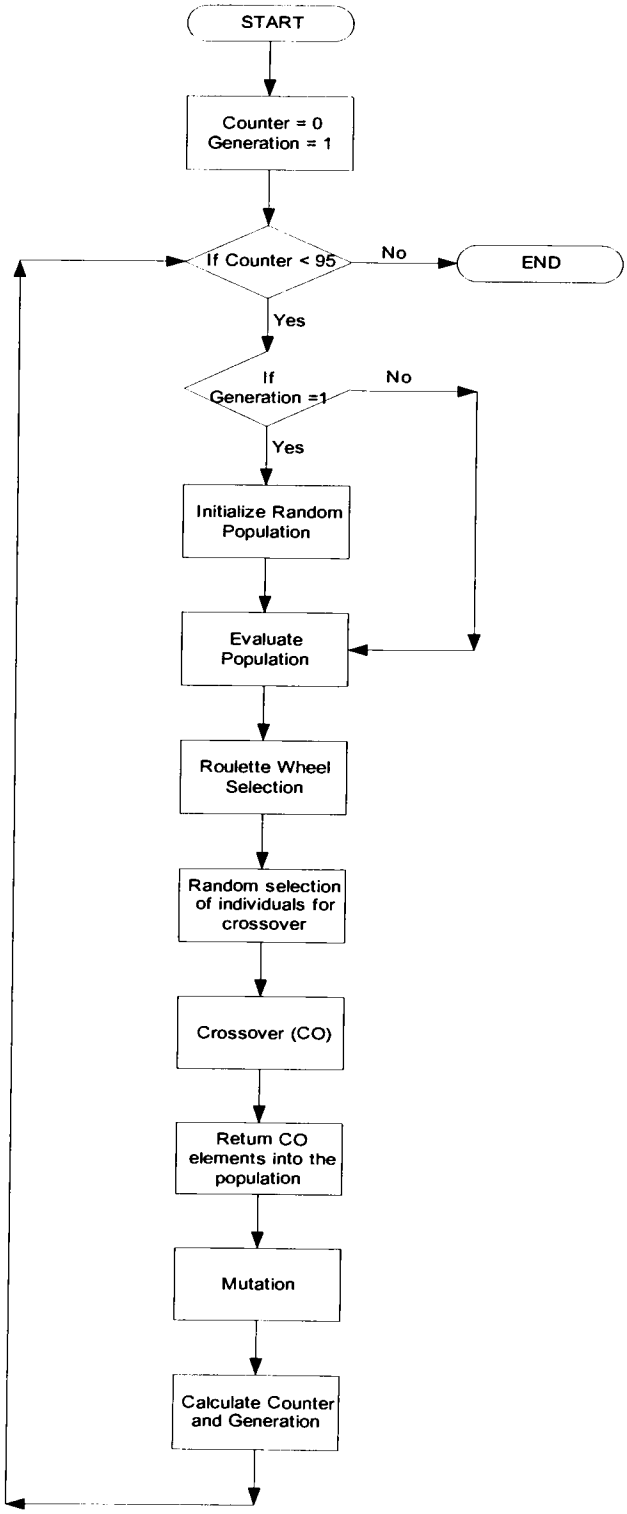


Figure 3: Flowchart for Path Representation

After the crossover population is generated, crossover needs to be performed. For path representation, there are three crossover operators and their implementation is discussed in detail in the following paragraphs.

For Partially Mapped Crossover (PMX), two individuals were chosen at random from the crossover population. Two random numbers, P1 and P2, with values between 2 and 9 were generated. The numbers falling between the minimum and maximum of P1 and P2 in the two individuals were swapped. Then starting from the first number to the minimum of P1 and P2 in the first individual, all the numbers were checked with the numbers between minimum and maximum of P1 and P2. If any of the numbers matched, then the number falls between first number and minimum of P1 and P2 was replaced by the corresponding number that is at the same position as the number it matched, but in the second individual. The same procedure is followed for numbers between the maximum of P1 and P2 to the last number in the individual. Again, the whole procedure is repeated with the second individual. The same procedure is repeated until all the pairs in the crossover population are exhausted.

For Order Crossover (OX), similar to PMX implementation, two individuals were chosen at random from the crossover population. Two random numbers P1 and P2, with values between 2 and 9 were generated. Starting from the number after maximum of P1 and P2, to the end of the individual, and then from the first to the maximum of P1 and P2, all the numbers in the two individuals are stored in that order in two new arrays. The elements of the new array formed by individual two were compared to the numbers between minimum and maximum of P1 and P2 in individual one. If any of the numbers matched, the number in the new array was replaced by zero. Later, all the zeros were deleted to form an updated new array. Now the numbers in the updated new array two were placed in the first individual chosen from the crossover population, starting from the number next to maximum of P1 and P2 to the end of the individual and then again from the first number to the number before minimum of P1

and P2. The same procedure is repeated with all the pairs of individuals in the crossover population.

Cyclic crossover (CX) is different in operation than the other two crossover operators discussed previously since the two random numbers P1 and P2 are not required. In cycle crossover, the crossover population CO was copied to a new population CONew, and the array CONew was filled with zeros. The first element from the first individual in the earlier CO population was copied into the same position in CONew. Then, the first element in the second individual in CO population was compared with all the elements in the first individual until a match was found. If, at the matched position, there is a zero in the CONew population (first individual), then the zero was replaced by the number at the same position in the CO population in the same (first) individual. This procedure is followed until the run does not find a zero in the CONew population when making comparisons. Once this happens, the rest of the zeros in the first individual in the CONew population were replaced by the numbers in the same positions in the second individual in the CO population. The same procedure is repeated with the second individual. The whole process is repeated until all the pairs in the CO population are exhausted and there are no zeros at any position in the CONew population.

Once the crossover operation was performed, the crossover individuals need to be put back into the original population with the other individuals which were not changed. The positions from where the crossover individuals were taken (from the original population) were stored in an array named COPos, and using these, the crossover elements were put back into the population.

Mutation was the next operation performed. For each element, in each individual of the population, a random number was generated. If the random number generated fell below the rate of mutation, then that particular element was mutated. Using the "rem" function, the exact position of the element was determined i.e. individual number and

the element number. A random integer number in the range of 1 to the maximum number of components being placed was generated. The element to be mutated was replaced by this number, and the location where the random number was located in the individual was replaced by whatever the number at the mutation location was.

The termination condition for the algorithm was 95% convergence i.e. the algorithm stops running when 95% of the individuals in the population are the same. Once the crossover and mutation operations are performed on the population it is evaluated to check for convergence and the individual that appears 95% of the times is taken to be the optimal solution. If 95% convergence is not met, then the convergence counter is set back to zero and the generation counter is increased by one and the population is allowed to run through another generation.

3.3.2 Ordinal Representation

Figure 4 shows the flowchart describing the steps for ordinal representation. A reference list was defined at the start of the algorithm and held constant. Initial list population was then generated using the "randint" function. The list population has elements in an individual randomly generated, but the value of each element is in the range of 1 to n-i+1, where n is the total number of components being placed and i is the current position of the element being generated. The list population was then decoded into actual tours. This was done by first copying the reference list to another array and then sequentially using the numbers generated by the randint function to get the appropriate number from the copied reference list. The number that is copied to the decoded population is then deleted from the copied reference list and the procedure is followed until all the elements from the copied reference list are deleted.

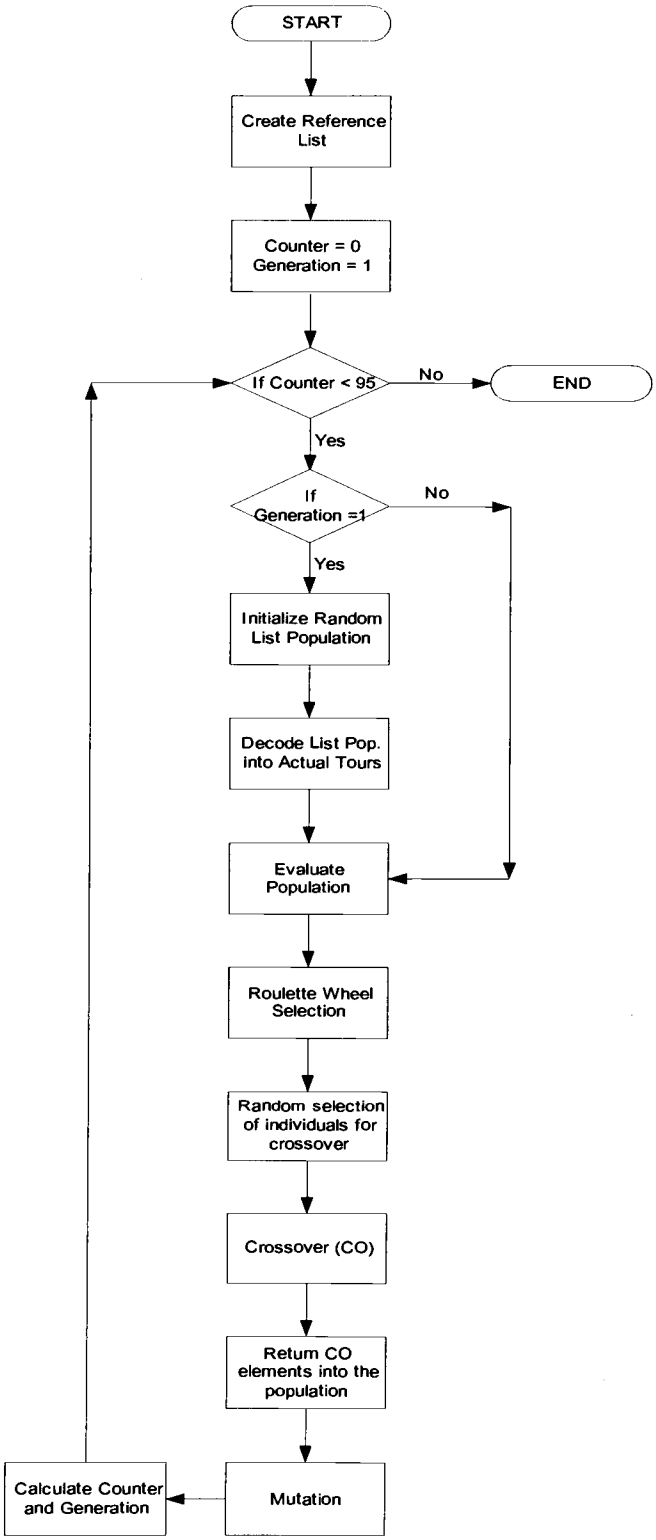


Figure 4: Flowchart for Ordinal Representation

The roulette wheel selection and random selection of individuals for crossover were implemented similar to what was discussed for path representation. Instead of using the actual population, the reference list was used. That was the only difference. Once the crossover individuals are chosen at random, the next step is to perform crossover.

Classical crossover is used with ordinal representation. One point crossover was implemented. A random number P1 was generated in the range of 2 to 9. Two individuals were randomly chosen and all the elements after P1 were swapped between the two individuals.

Implementing mutation is also simpler when compared to the one implemented for path representation. A random number is generated for each of the elements. If the random number is smaller than the mutation rate, then the value of the corresponding element is randomly changed using the randint function, so that it is still in the range. Again the termination condition is the same as discussed for path representation.

3.3.3 Adjacency Representation

Figure 5 shows the flowchart describing the steps for adjacency representation. Adjacency representation was the most difficult to implement. The initial population is generated similar to path representation using the randperm function. The population evaluation, selection probability calculations, roulette wheel selection, random selection of individuals for crossover are all implemented by using the procedure described under path representation. Once the crossover population is generated, the next step is crossover. But before this is done, representation needs to be changed from path to adjacency.

For each individual from the original population, an adjacency equivalent is generated using the following procedure. For all the elements from 1 to n -1 (n is the number of components being placed), the element value in the original population decides the

next position to be filled in the adjacency representation, and the next element on the list in path representation is the value of the element being placed at the selected location in adjacency representation. For the n^{th} element, the value is the value of the first element in path representation. This procedure is repeated for the original population as well as the crossover population.

For alternating edges crossover, a random integer in the range of 1 to the number of components being placed is generated to start the crossover process. This is stored in variable named "StartIndex". An array, "VisitedList", with size equal to the number of components being placed is created, with zeros filled in. Each time a particular element is placed in the offspring population, the position at which the element is placed, the zero in that particular position is replaced to one in the VisitedList array. During the run, if for a particular position, a one is seen at that position in the VisitedList array, then the remaining elements which are yet to be placed are randomly placed in the remaining blank positions in the offspring so that there is no conflict. Once the crossover operation is performed on all the individuals in the crossover population, the individuals are put back into the original population.

Heuristic Crossover is an extension of alternating edges crossover. It has the same initial steps until creating the array "VisitedList". Once the run is started by the randomly selected "StartIndex", the next element to be entered into the offspring individual is chosen depending upon which element from the two parents gives the least distance. This procedure is followed until there is a conflict.

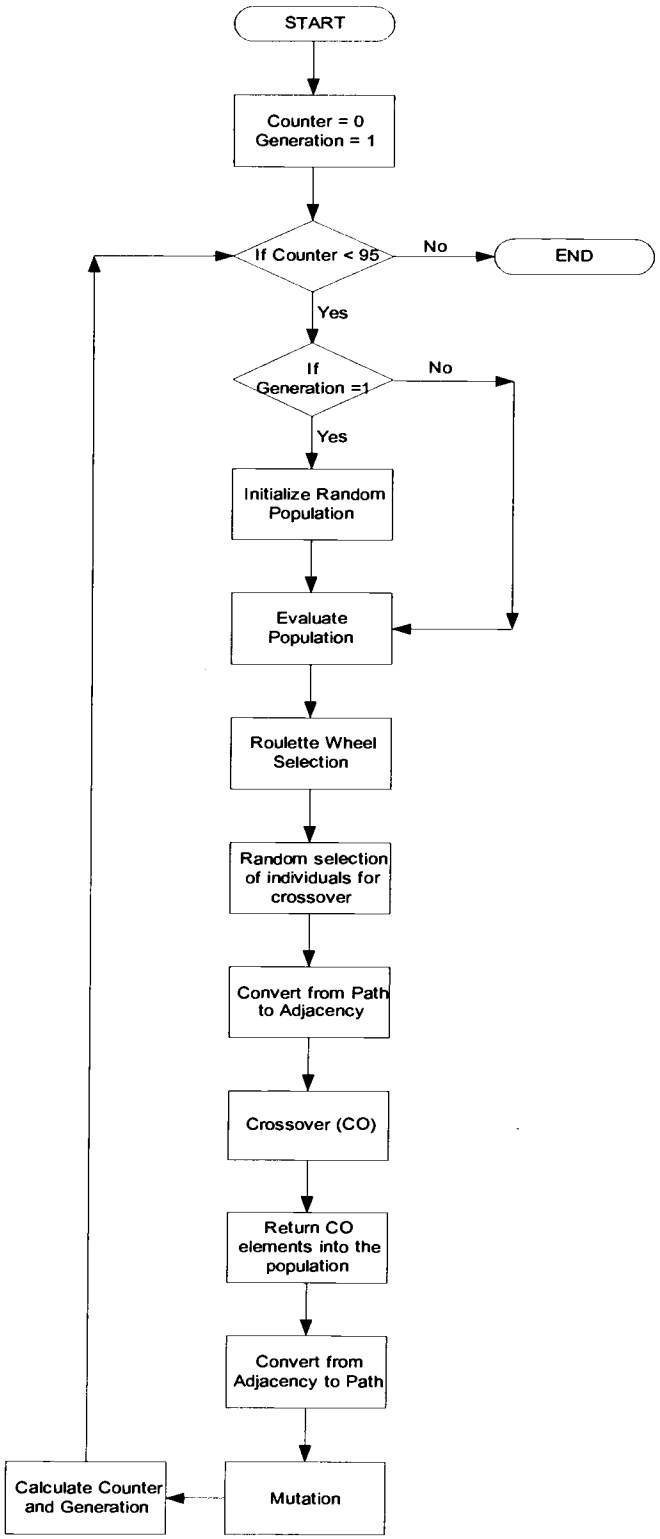


Figure 5: Flowchart for Adjacency Representation

Once the crossover operation is performed, the population is converted back to path representation. This is done by first entering a one at the first position and then the following elements in the tour are filled in starting with the first element that appears in the adjacency individual and then saving the element value in a variable named "Index". The next element in the path representation is the one at the position "Index" in the adjacency representation.

After the population is converted back into path representation the mutation operation is performed as discussed under path representation. The termination criterion is also the same as discussed previously.

4. RESULTS

There are four sets of parameters that are significant for the GA search. These are:

- Representation type.
- A particular type of crossover corresponding to representation chosen.
- Crossover rate (CR).
- Mutation rate (MR).

The first two parameters are combined into one parameter as each type of crossover is exclusive to the type of representation that they are used for. The combined parameter is called 'Method'. Six methods were investigated in this research and are summarized in Table 1.

Table 1: Combination of Representation and Crossover for each Method

Method	Representation + Crossover
1	Path + Partially Mapped
2	Path + Order
3	Path + Cycle
4	Ordinal + Classical
5	Adjacency + Alternating Edges
6	Adjacency + Heuristic

The crossover rates were set at 11 levels and ranged from 0.20 to 0.30, with an interval of 0.01. These values were chosen based on previous research of the algorithm performance (Michalewicz, 1996). Also, the number of levels was chosen with the goal of keeping complexity of the problem under control.

In the initial design the mutation rates were set at 4 levels - 0.005, 0.01, 0.015, 0.020. But with the termination criteria of 95% convergence, the algorithm failed to converge at mutation rates of 0.01, 0.015 and 0.020. Since, in this research the focus is to study effects of the parameters discussed earlier, the termination criteria had to be held at 95% for fair comparisons. Thus the mutation rates were changed and set at 3 levels - 0.0025, 0.005, 0.0075.

For each combination of method, crossover rate and mutation rate, the program was run 25 times. Sample size adequacy was determined by using the formula:

$$n => (z * s / e)^2,$$

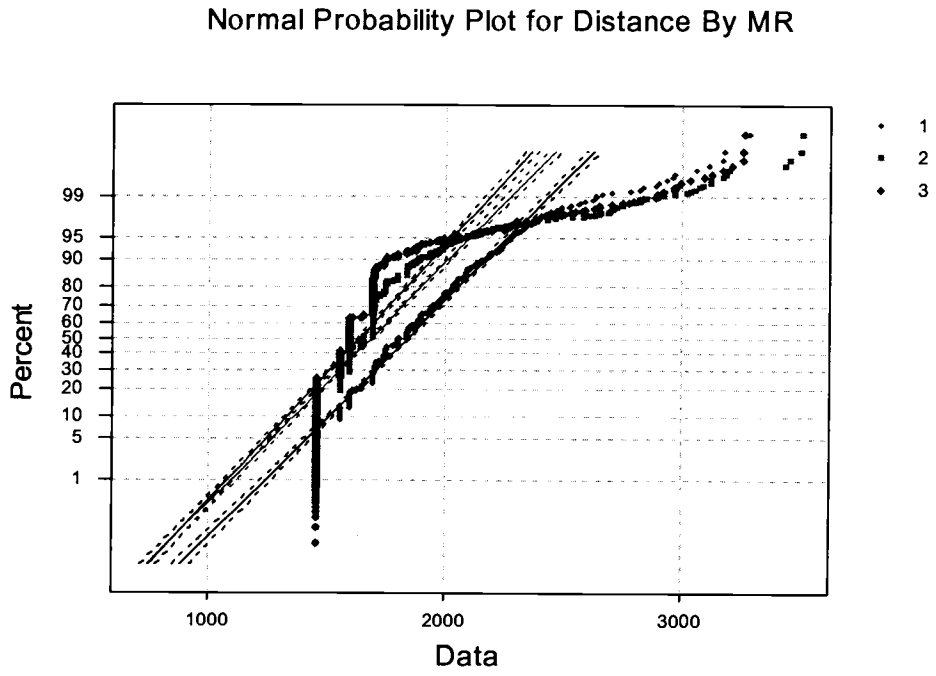
where n is the sample size, s is the population standard deviation and e is the acceptable error. The estimate for standard deviation was 280mm and the acceptable error was set at 150mm (10% of the optimal). This sample size calculation is for a simple random sampling process. This is appropriate because each sequence in the population is generated randomly.

A total of 4950 data points (6 methods * 3 mutation rates * 11 crossover rates * 25 replications) were used for all subsequent analyses. Distance, the number of generations and the time taken for the algorithm to run were recorded as response variables. The program was run for a 10-component placement problem. A small problem size was chosen to have a better control on the algorithm execution.

4.1 Normality Check

Figures 6-11 show the Normal Probability Plots (NPP) for each of the response variables against method and mutation rate. It can be seen that the NPP's are skewed. In general, moderate departures from normality are of little concern in fixed effects ANOVA. Additionally, an error distribution that has considerably thicker or thinner tails then normal is more of a concern than a skewed distribution. Because the F test is only slightly affected, it can be said that ANOVA (and related procedures such as

multiple comparisons) is robust to the normality assumption (Montgomery, 2001). But there is no quantitative measure to determine how much departure from normality or skewness is actually acceptable. However, the skewness observed in the generated data sets does appear to have substantial deviation from normality. As a result, nonparametric tests, which are more conservative test statistics, were used for subsequent analysis. Nonparametric testing produces statistical inferences free from any distributional assumptions.

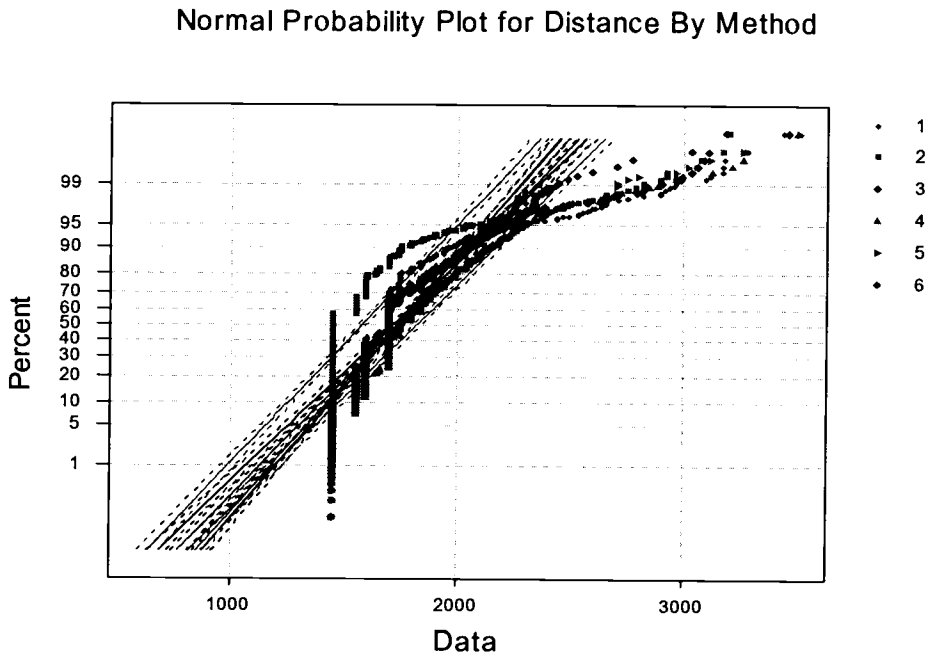


MR 1 = 0.0025

MR 2 = 0.0050

MR 3 = 0.0075

Figure 6: NPP for Distance by MR



Method 1 = Path + Partially mapped

Method 2 = Path + Order

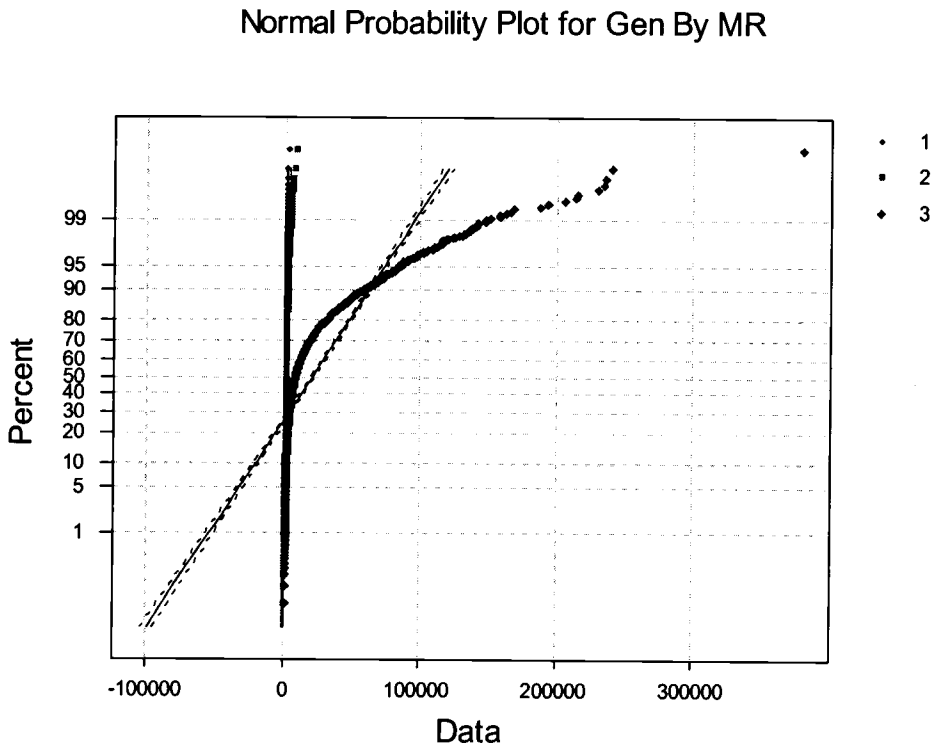
Method 3 = Path + Cycle

Method 4 = Ordinal + Classical

Method 5 = Adjacency + Alternating edges

Method 6 = Adjacency + Heuristic

Figure 7: NPP for Distance by Method

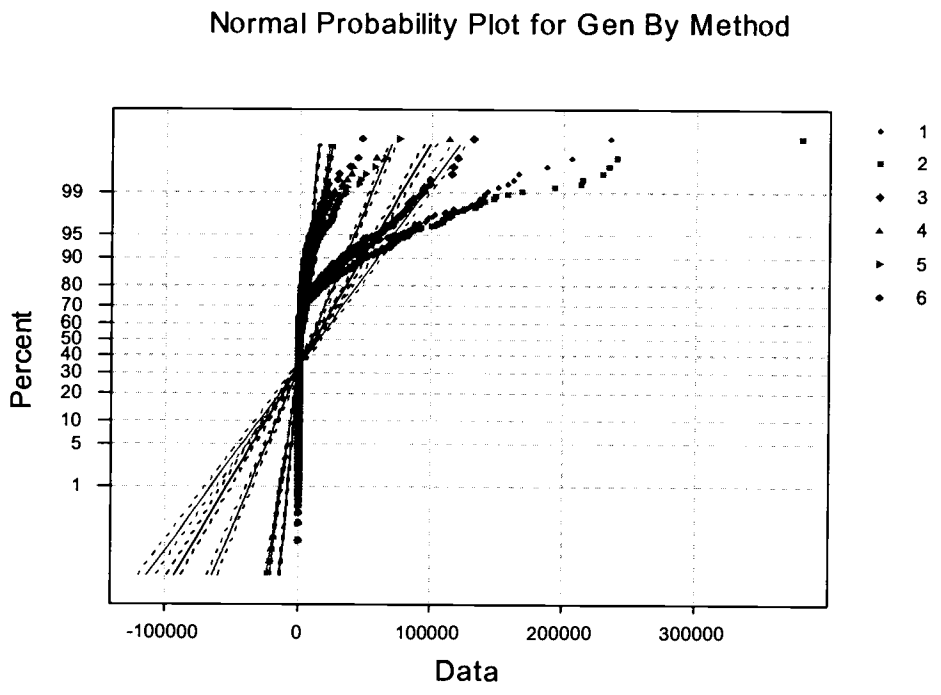


MR 1 = 0.0025

MR 2 = 0.0050

MR 3 = 0.0075

Figure 8: NPP for Generation by MR



Method 1 = Path + Partially mapped

Method 2 = Path + Order

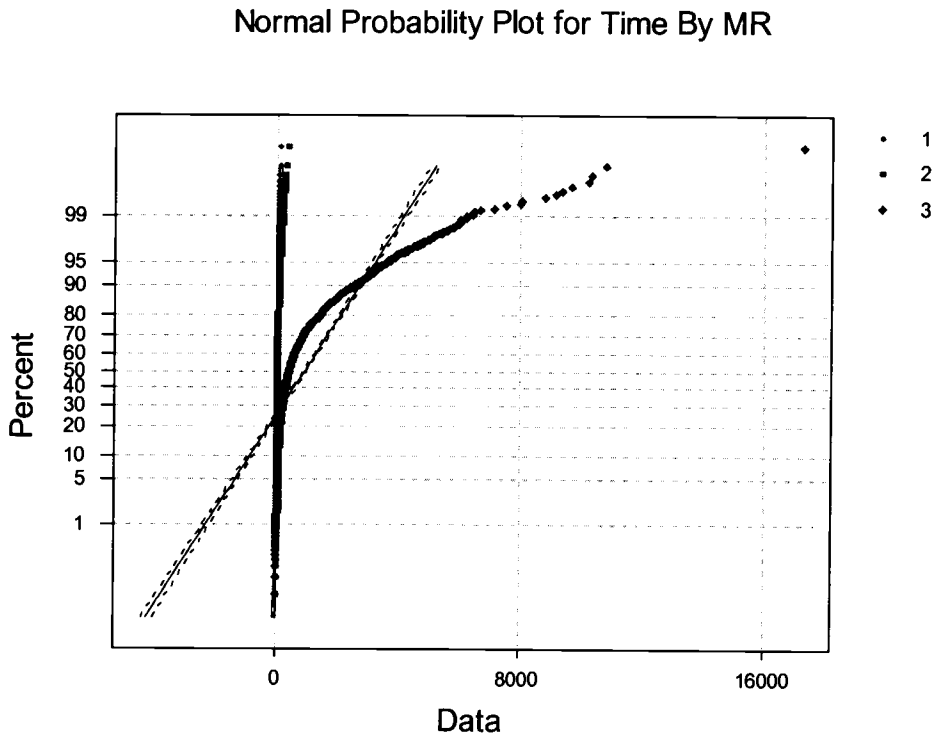
Method 3 = Path + Cycle

Method 4 = Ordinal + Classical

Method 5 = Adjacency + Alternating edges

Method 6 = Adjacency + Heuristic

Figure 9: NPP for Generation by Method

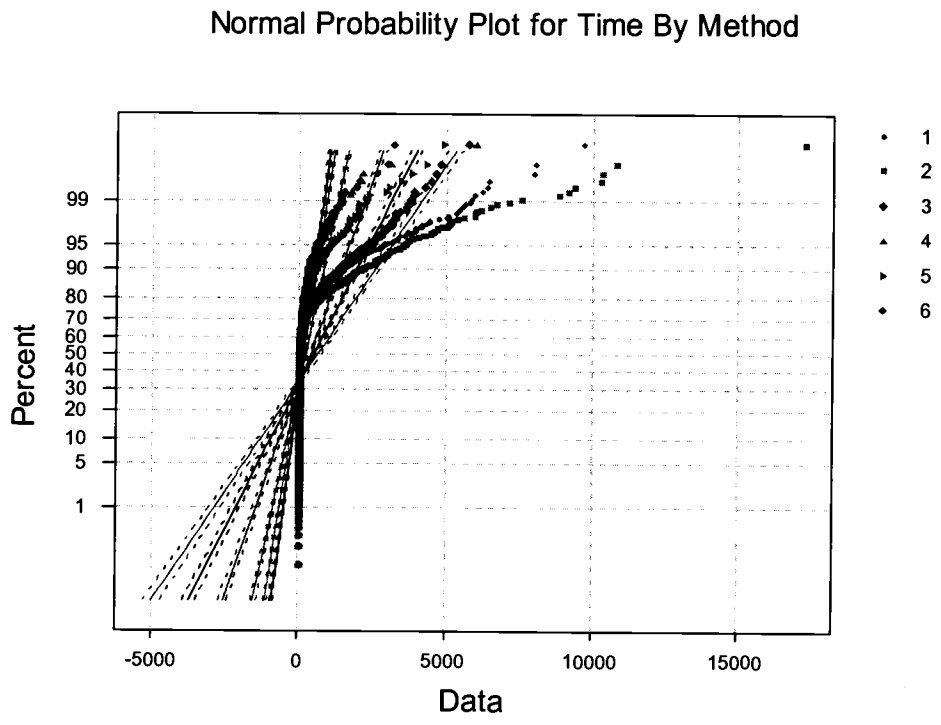


MR 1 = 0.0025

MR 2 = 0.0050

MR 3 = 0.0075

Figure 10: NPP for Time by MR



Method 1 = Path + Partially mapped

Method 2 = Path + Order

Method 3 = Path + Cycle

Method 4 = Ordinal + Classical

Method 5 = Adjacency + Alternating edges

Method 6 = Adjacency + Heuristic

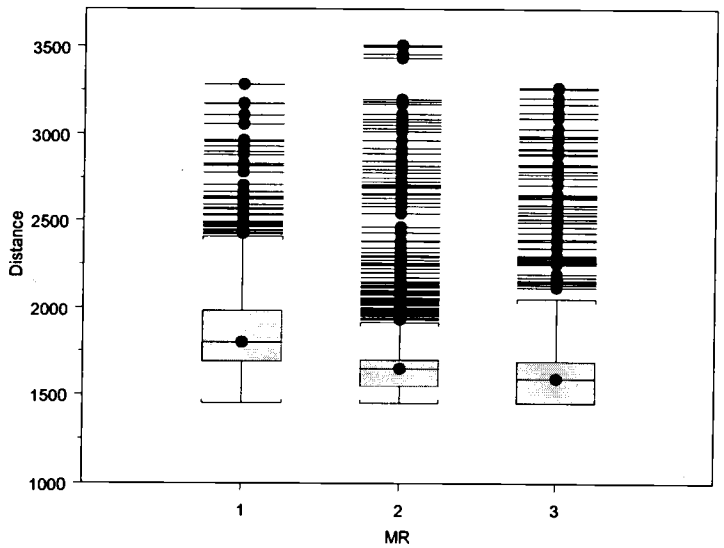
Figure 11: NPP for Time by Method

4.2 Mood's Median Test

Mood's median test is used to test the equality of medians from two or more populations and provides a nonparametric alternative to the one-way ANOVA. Mood's median test is also called a median test or sign scores test.

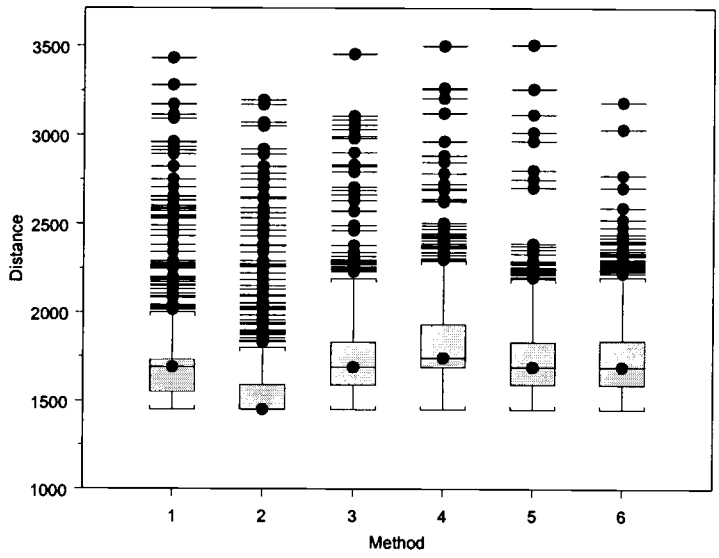
4.2.1 Distance Analysis

Figures 12-14 show the box plots for response variable Distance by each of the factors. Figures 15-17 shows the Mood's median test for response variable Distance with factors CR, MR and method respectively. The p-values for the factors in that order are 0.981, <0.005 and <0.005. The figures also show a plot of median values and the 95% confidence interval for each of the factors. Also, the interaction plots for response variable Distance are shown in figures 18-20.



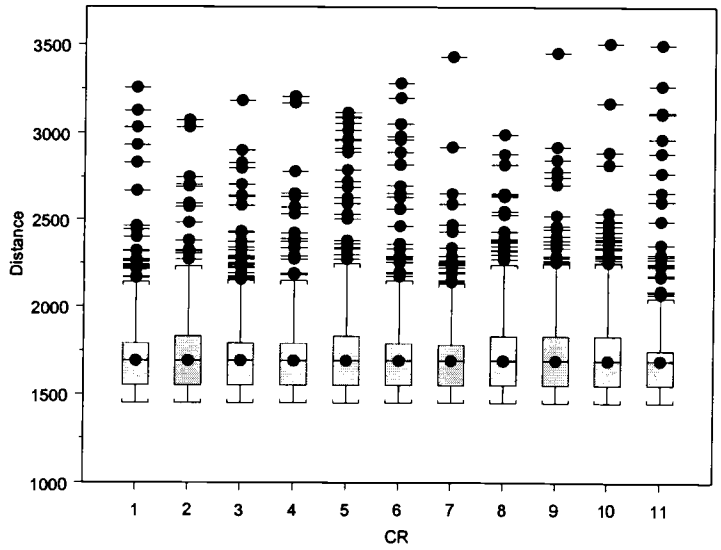
n = 1650 for each MR

Figure 12: Box plot for Distance by MR



n = 825 for each Method

Figure 13: Box plot for Distance by Method



n = 450 for each CR

Figure 14: Box plot for Distance by CR

```
Chi-Square = 3.03   DF = 10   P = 0.981

                                        Individual 95.0% CIs
CR         N<=    N>   Median    Q3-Q1  -----+---------+---------+---------+-
0.20       275   175   1690.5    240.5           (-------------------+
0.21       268   182   1690.5    281.3            (-------------------+
0.22       277   173   1690.5    241.5                               +
0.23       271   179   1690.5    240.5            (-------------------+
0.24       267   183   1690.5    283.1                               +
0.25       273   177   1690.5    240.5            (-------------------+
0.26       273   177   1690.5    233.1            (-------------------+
0.27       263   187   1690.5    282.1  (-----------------------------+
0.28       275   175   1690.5    282.1            (-------------------+
0.29       263   187   1690.5    282.1            (-------------------+
0.30       281   169   1690.5    206.2                               +
                                        -----+---------+---------+---------+-
                                          1640      1660      1680      1700
Overall median = 1690.5
```

Figure 15: Mood's median test for Distance by CR

```
Chi-Square = 1104.10   DF = 2   P = 0.000

                                        Individual 95.0% CIs
MR         N<=    N>   Median    Q3-Q1  ---+---------+---------+---------+---
0.0025     477  1173     1800      294                                 (+----)
0.0050    1126   524     1650      150           +-----)
0.0075    1383   267     1592      241  +
                                        ---+---------+---------+---------+---
                                        1610      1680      1750      1820
Overall median = 1691
```

Figure 16: Mood's median test for Distance by MR

```
Chi-Square = 449.46   DF = 5   P = 0.000

                                        Individual 95.0% CIs
Method     N<=    N>   Median    Q3-Q1  -----+---------+---------+---------+-
1          547   278     1691      183                          +
2          700   125     1450      142  +
3          504   321     1691      242                          +
4          289   536     1742      242                              (+----)
5          490   335     1691      241                          +
6          456   369     1691      250                          +
                                        -----+---------+---------+---------+-
                                          1500      1600      1700      1800
Overall median = 1691
```

Figure 17: Mood's median test for Distance by Method

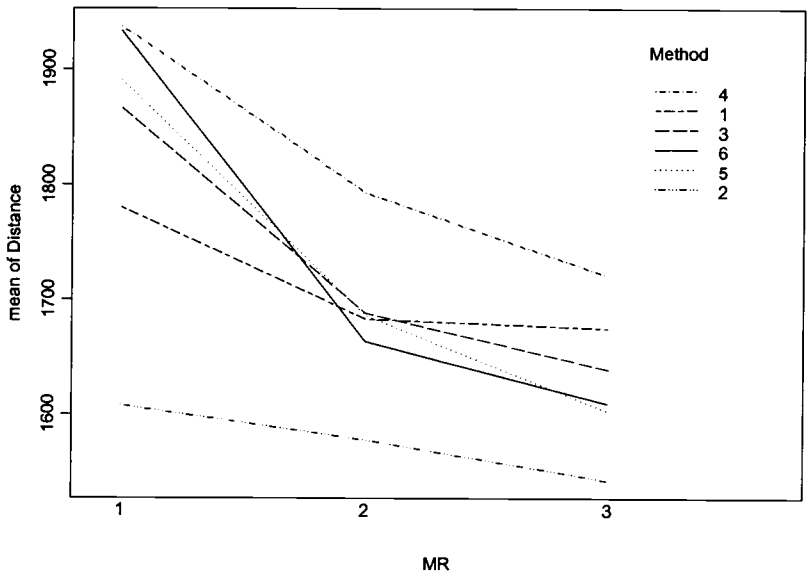


Figure 18: Interaction plot for Distance for MR by Method

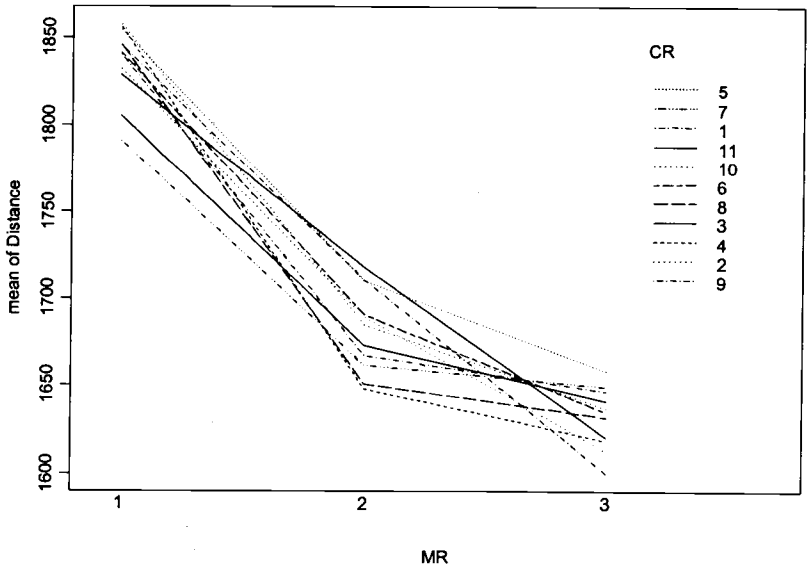


Figure 19: Interaction plot for Distance for MR by CR

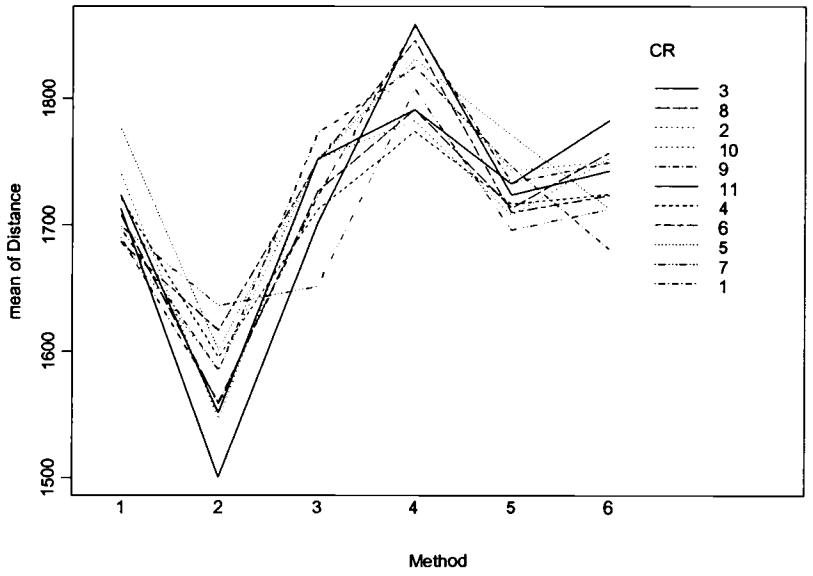
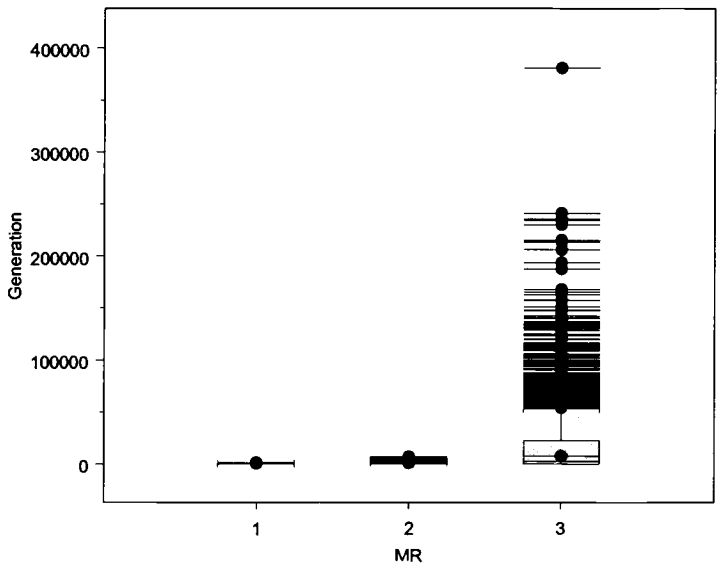


Figure 20: Interaction plot for Distance for Method by CR

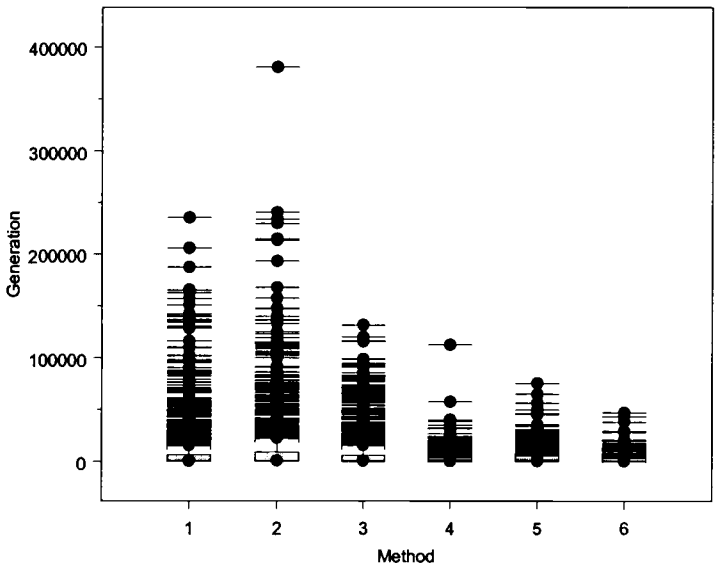
4.2.2 Generation Analysis

Figures 21-23 show the box plots for response variable Generation by each of the factors. Figures 24-26 shows the Mood's median test for response variable Generation with factors CR, MR and method respectively. The p-values for the factors in that order are 0.910, <0.005 and <0.005. The figures also show a plot of median values and the 95% confidence interval for each of the factors. Also, the interaction plots for response variable Generation are shown in figures 27-29.



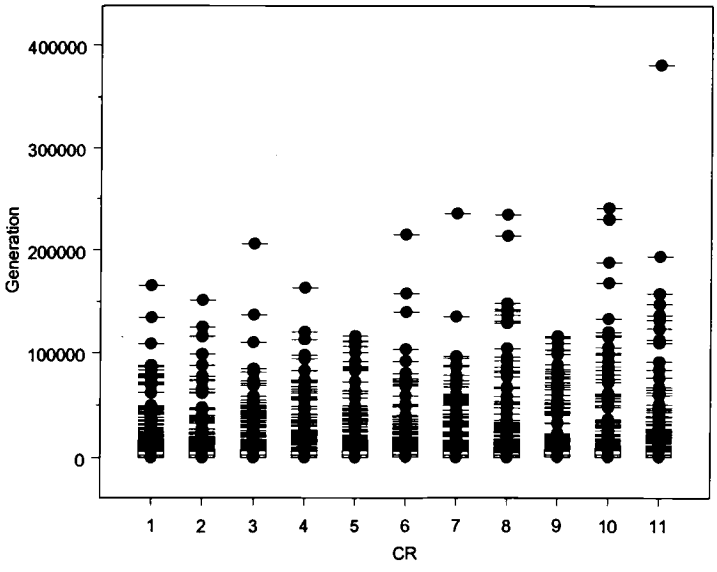
n = 1650 for each MR

Figure 21: Box plot for Generation by MR



n = 825 for each Method

Figure 22: Box plot for Generation by Method



n = 450 for each CR

Figure 23: Box plot for Generation by CR

```
Chi-Square = 4.71   DF = 10   P = 0.910

                                          Individual 95.0% CIs
CR         N<=    N>   Median    Q3-Q1  ----+---------+---------+---------+--
0.20       233   217      487     2669        (------+---------)
0.21       233   217      492     2640       (-------+---------)
0.22       222   228      533     2675          (--------+-------)
0.23       227   223      509     2554        (--------+---------)
0.24       222   228      541     2752          (---------+-------)
0.25       232   218      495     2332        (-------+---------)
0.26       226   224      503     2396       (--------+--------)
0.27       234   216      460     2276   (--------+----------)
0.28       217   233      548     2365              (-----+----------)
0.29       214   236      577     2759           (-----------+----------)
0.30       217   233      542     2367           (--------+---------)
                                         ----+---------+---------+---------+--
                                           400       500       600       700

Overall median = 522
```

Figure 24: Mood's median test for Generation by CR

```
Chi-Square = 2787.03   DF = 2   P = 0.000

                                          Individual 95.0% CIs
MR         N<=    N>   Median    Q3-Q1  ----------+---------+---------+------
0.0025    1593    57      175      147   +
0.0050     807   843      535      583    +
0.0075      77  1573     7572    20401                              (--+--)
                                         ----------+---------+---------+------
                                                2500      5000      7500

Overall median = 522
```

Figure 25: Mood's median test for Generation by MR

```
Chi-Square = 84.34   DF = 5   P = 0.000

                                          Individual 95.0% CIs
Method     N<=    N>   Median    Q3-Q1  ----+---------+---------+---------+--
1          385   440      579     6145           (--+-----)
2          331   494      801     8731                    (----+--------)
3          376   449      642     6105              (---+----)
4          487   338      370     1282  (--+)
5          434   391      484     2244        (-+--)
6          464   361      422     1205      (+--)
                                         ----+---------+---------+---------+--
                                           400       600       800      1000

Overall median = 522
```

Figure 26: Mood's median test for Generation by Method

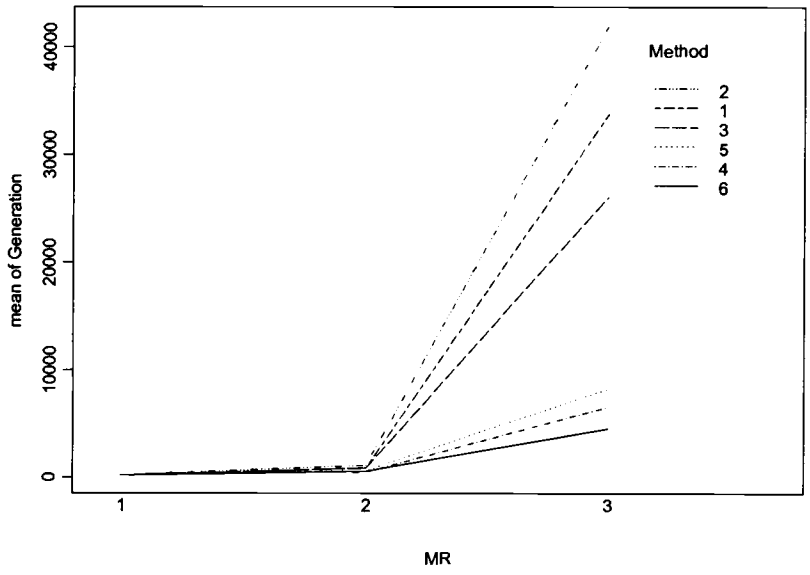


Figure 27: Interaction plot for Generation for MR by Method

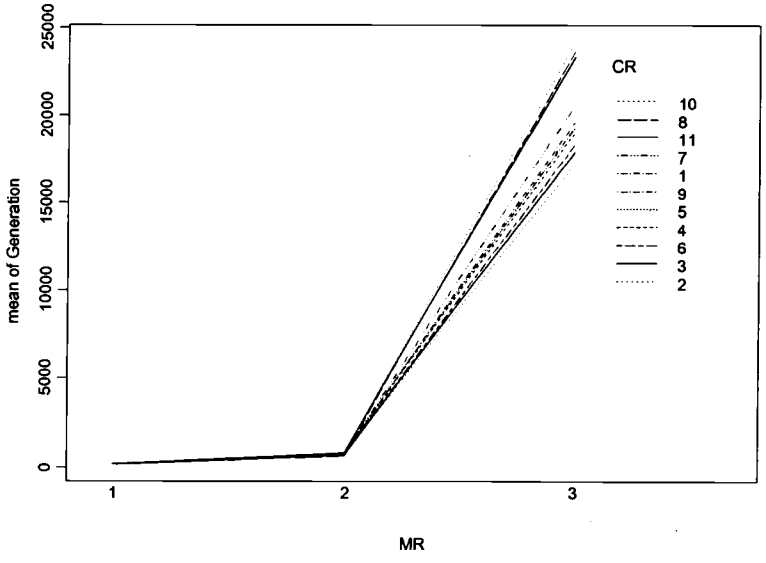


Figure 28: Interaction plot for Generation for MR by CR

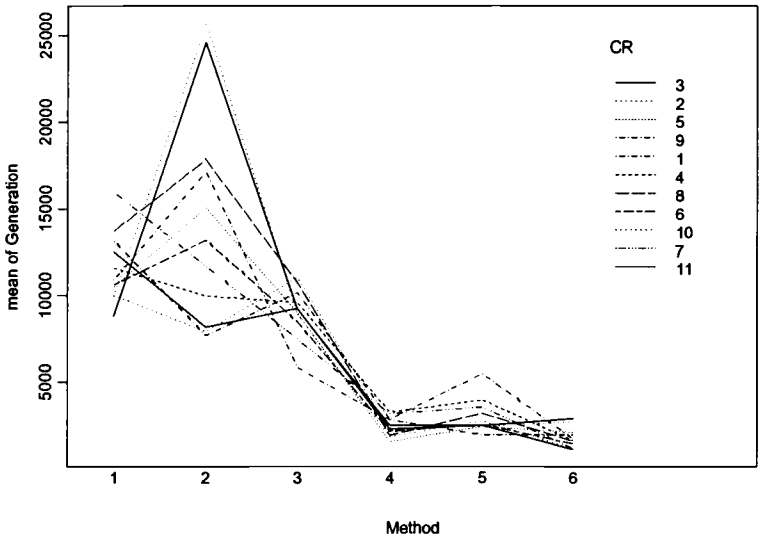
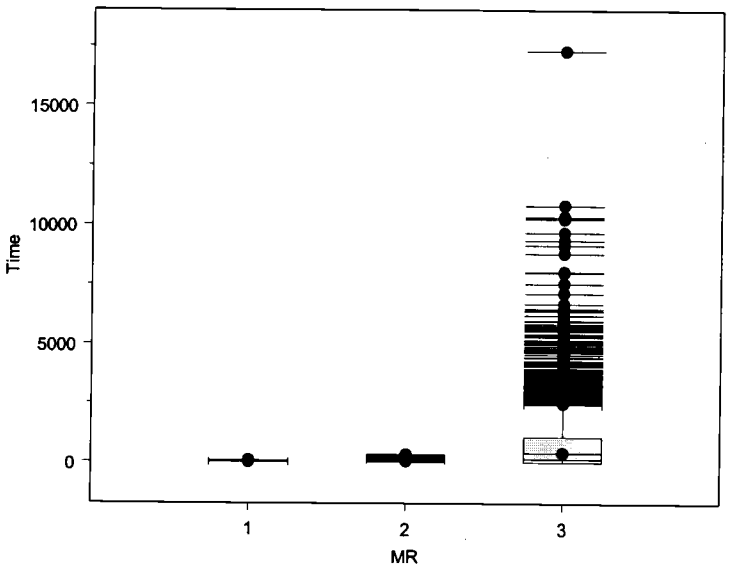


Figure 29: Interaction plot for Generation for Method by CR

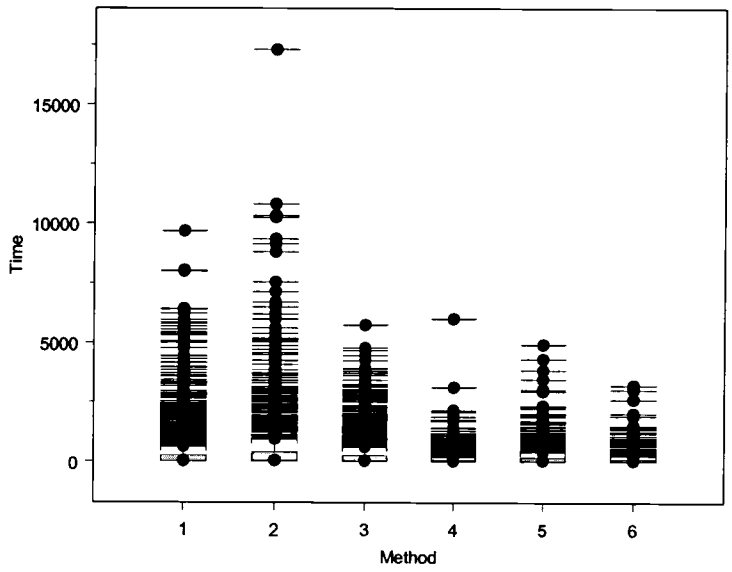
4.2.3 Time Analysis

Figures 30-32 show the box plots for response variable Time by each of the factors. Figures 33-35 shows the Mood's median test for response variable Time with factors CR, MR and method respectively. The p-values for the factors in that order are 0.693, <0.005 and <0.005. The figures also show a plot of median values and the 95% confidence interval for each of the factors. Also, the interaction plots for response variable Time are shown in figures 36-38.



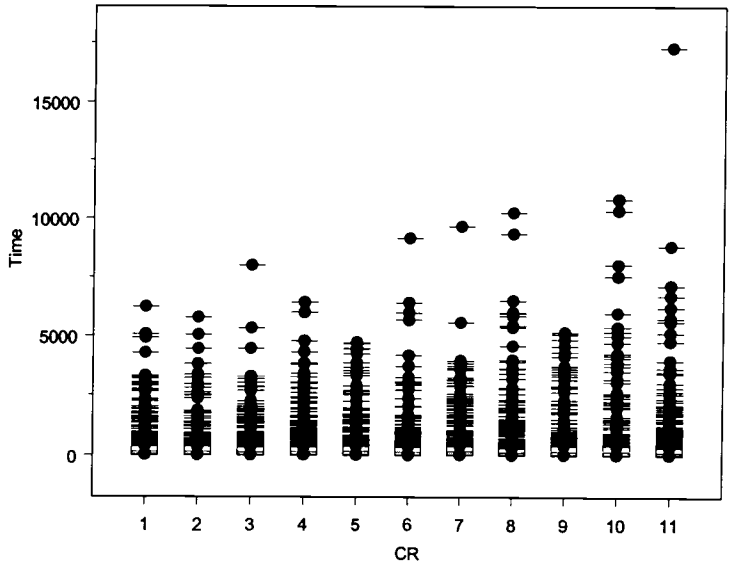
n = 1650 for each MR

Figure 30: Box plot for Time by MR



n = 825 for each Method

Figure 31: Box plot for Time by Method



n = 450 for each CR

Figure 32: Box plot for Time by CR

```
Chi-Square = 7.34   DF = 10   P = 0.693

                                          Individual 95.0% CIs
CR       N<=    N>   Median    Q3-Q1  --------+---------+---------+--------
0.20     229   221     29.2    148.6      (-----------+--------)
0.21     228   222     28.9    143.1   (--------------+-----------------)
0.22     235   215     26.5    130.4      (------+-----------)
0.23     223   227     30.6    137.4          (----------+-----------)
0.24     214   236     32.7    155.3             (-----------+--------)
0.25     237   213     27.7    140.1       (-------+-----------)
0.26     226   224     29.7    156.3       (-----------+-----------)
0.27     236   214     27.3    121.8      (--------+----------)
0.28     210   240     32.7    154.7                 (-------+--------------)
0.29     219   231     32.6    160.7               (---------+-----------)
0.30     218   232     31.8    133.3                (-------+----------)
                                        --------+---------+---------+--------
                                               25.0      30.0      35.0

Overall median = 30.0
```

Figure 33: Mood's median test for Time by CR

```
Chi-Square = 2812.06   DF = 2   P = 0.000

                                          Individual 95.0% CIs
MR       N<=    N>   Median    Q3-Q1  ----------+---------+---------+------
0.0025  1603    47        9        8   +
0.0050   791   859       31       33    (+
0.0075    81  1569      399      936                                    (--+--)
                                        ----------+---------+---------+------
                                                120       240       360

Overall median = 30
```

Figure 34: Mood's median test for Time by MR

```
Chi-Square = 67.76   DF = 5   P = 0.000

                                          Individual 95.0% CIs
Method   N<=    N>   Median    Q3-Q1  -----+---------+---------+---------+-
1        439   386     24.8    246.0        (---+-------)
2        385   440     35.5    377.3                    (-------+--------)
3        427   398     27.7    244.9           (-----+----)
4        497   328     20.3     69.4   (--+--)
5        350   475     38.7    153.3                            (-----+------
)
6        377   448     34.1     88.5                       (---+-----)
                                        -----+---------+---------+---------+-
                                          21.0      28.0      35.0      42.0

Overall median = 30.0
```

Figure 35: Mood's median test for Time by Method

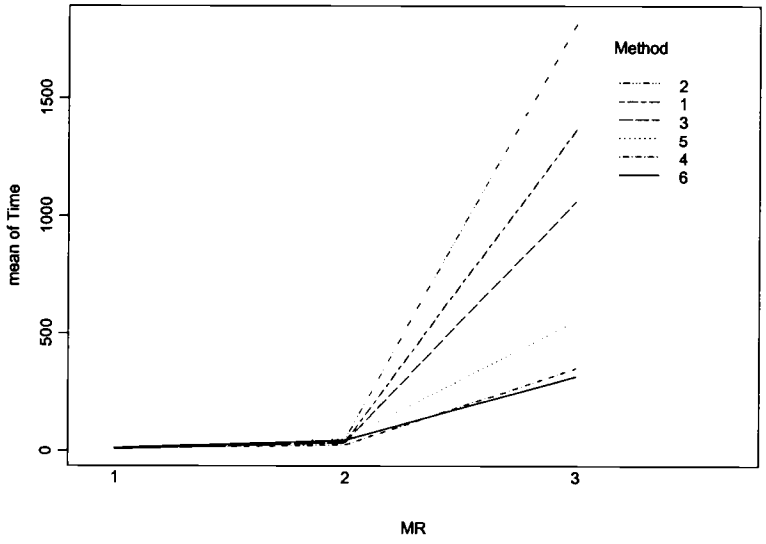


Figure 36: Interaction plot for Time for MR by Method

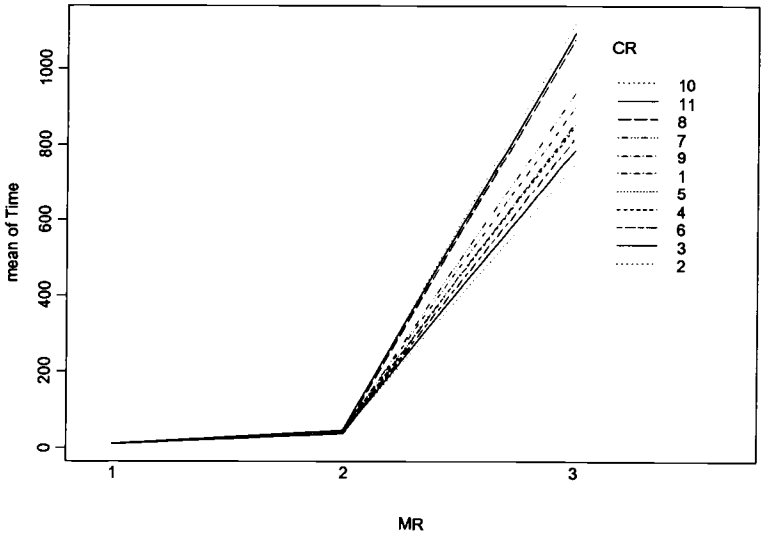


Figure 37: Interaction plot for Time for MR by CR

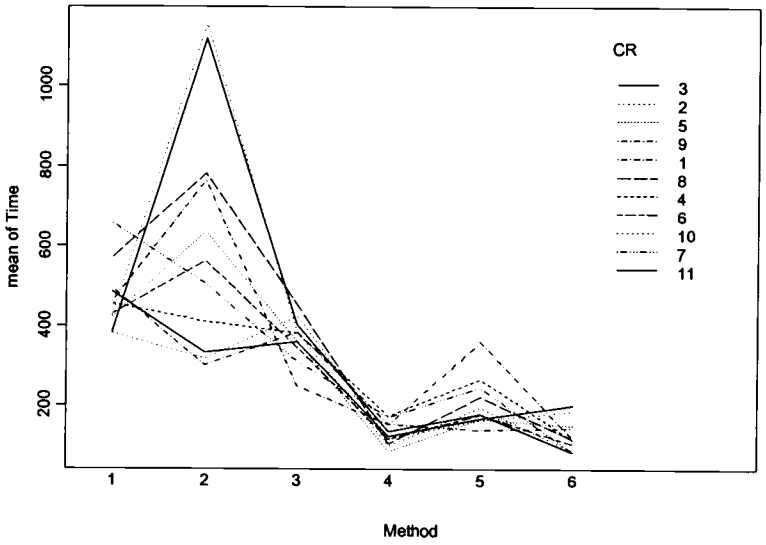


Figure 38: Interaction plot for Time for Method by CR

5. CONCLUSIONS AND FUTURE RESEARCH

5.1 Discussion

A genetic algorithm was developed for solving a traveling salesman problem to effectively determine a near-optimal solution for the sequencing problem on a component placement machine. Three vector representation schemes, along with the crossover and mutation operators' exclusive to a representation scheme, were compared. Mood's median test was performed to identify the significant factors for each of the response variables.

Mood's median test identified mutation rate and method as the significant factors for response variable distance. At a mutation rate of 0.0075, the median distance was close to optimal and the confidence interval was small, indicating that this particular mutation rate consistently yields better solutions. For a mutation rate of 0.0025, the median was high and the confidence interval extended towards the higher size, indicating that it is not as consistent. Comparison of methods indicated that method 2 (path + order) had the best performance, with optimal value for median and very tight confidence interval. All the other methods, except for method 4 (ordinal + classical), performed moderately and there was not a significant difference between those methods. Method 4 (ordinal + classical) had the worst performance and also had a larger confidence limit.

For response variable generation and time, mutation rate and method were the significant factors and the results were similar. For both generation and time, the responses were low and had tight confidence for mutation rates of 0.0025 and 0.005. For a mutation rate of 0.0075, the number of generations and time increased significantly. Also, the first three methods took a larger number of generations to solve as compared to the other three methods and had large confidence intervals. However,

for response variable time, except method 4 (ordinal + classical), there was not a significant difference in times for all the other methods. The confidence intervals on all these methods were large. From the comparison of different methods for response variables generation and time, it can be concluded that though it takes significantly fewer generations to get to the near optimal solution using methods 5 (adjacency + alternating edges) and 6 (adjacency + heuristic) when compared to methods 1 (path + partially mapped), 2 (path + order) and 3 (path + cycle), the time taken for the algorithm to run is not significantly different for the methods mentioned.

Based on the results obtained, mutation rate of 0.005 and Method 2 (path + order) is recommended for solving the TSP. Since higher mutation rates lead to larger number of generations and higher solving times, its use in the industry may not be practical.

The same problem was also solved by using the algorithm used by the component placement machine. The problem was solved multiple times, but yielded the same result every time. The sequence was solved to find the distance traveled by the head, and was found to be 2282mm. Compared to the optimal value of 1450mm, it can be seen that the performance of the component placement machines' algorithm is very poor.

5.2 Future Research

This research can be further extended to investigate the effects of other GA parameters on the execution of the algorithm. Potential areas could include comparing different selection criteria, tightening or relaxing the termination conditions or measuring algorithm performance for other response variables. For example, comparing the different selection schemes, like tournament selection or elitism with the roulette wheel selection could help find the one that works the best for the component placement problem. Also, devising a composite response variable which measures the

distance traveled by the placement head and time taken to solve, would help in drawing fairer conclusions.

Feeder optimization is another area of extension which can be simultaneously solved with the sequencing problem. The feeder optimization not only affects the way the sequencing is done, but also has an effect on the setup of the machine. But the feeder optimization is equipment specific as the feeder locations and types change from machine to machine.

Most of the component placement machines in use have multiple placement heads. Thus, solving the sequencing problem for a multiple headed placement machine is also another area of future research. Multiple sequences, one for each head, need to be simultaneous monitored and improved to get an optimal sequence. As the number of heads increases, the complexity of the problem increases exponentially.

BIBLIOGRAPHY

Brandeau, M. L., Billington, C. A., 1991, Design of manufacturing cells: operation assignment in printed circuit board manufacturing. *Journal of Intelligent Manufacturing*, Vol. 2, 95-106.

Buckles, B. P., Petry, F. E., 1992, Genetic algorithms. *IEEE Computer Society Press Technology Series*.

Burke, E. K., Cowling, P. I., Keuthen, R., 1999, New models and heuristics for component placement in printed circuit board assembly. *Proceedings of the IEEE International Conference on Information, Intelligence and Systems (ICIIS)*, 133-140.

Capps, C. H., 1998, Setup reduction in PCB assembly: A group technology application using genetic algorithms. *Master's Thesis, Oregon State University*.

Crama, Y., Flippo, O. E., Klundert, J., Spieksma, F. C. R., 1996, The component retrieval problem in printed circuit board assembly. *International Journal of Flexible Manufacturing Systems*, Vol. 8, 287-312.

Crama, Y., Flippo, O. E., Klundert, J., Spieksma, F. C. R., 1997, The assembly of printed circuit boards: a case with multiple machines and multiple board types. *European Journal of Operations Research*, Vol. 98, 457-472.

Crama, Y., Klundert, J., Spieksma, F. C. R., 2002, Production planning problems in printed circuit board assembly. *Discrete Applied Mathematics*, Vol. 123, 339-361.

Davis, L., 1987, Genetic algorithms and simulated annealing. *Pitman Publishing*.

Davis, L., 1991, Handbook of genetic algorithms. *Van Nostrand Reinhold.*

Garetti, M., Pozzetti, A., Tavecchio, R., 1996, Production scheduling in SMT electronic boards assembly. *Production Planning and Control*, Vol. 7, 197-204.

Gen, M., Cheng, R., 2000, Genetic algorithms and engineering optimization. *John Wiley & Sons, Inc.*

Goldberg, D. E., 1989, Genetic algorithms in search, optimization and machine learning. *Addison-Wesley Publishing Company, Inc.*

Gunther, H. O., Gronalt, M., Zeller, R., 1998, Job sequencing and component set-up on a surface mount placement machine. *Production Planning and Control*, Vol. 9, 201-211.

Jeevan, K., Parthiban, A., Seetharamu, K. N., Azid, I. A., Quadir, G. A., 1985, Optimization of PCB component placement using genetic algorithms. *Journal of Electronics Manufacturing*, Vol. 11, 69-79.

Ji, P., Wan, Y. F., 2001, Planning for printed circuit board assembly: The state-of-the-art-review. *International Journal of Computer Applications in Technology*, Vol. 14, 136-144.

Johnsson, M., Smed, J., 2001, Observations on PCB assembly optimization. *Electronics Packaging and Production*, Vol. 41, 38-42.

Khoo, L. P., Ng, T. K., 1998, A genetic algorithm-based planning system for PCB component placement. *International Journal of Production Economics*, Vol. 54, 321-332.

Lee, S. H., Lee, B. H., Park, T. H., 1999, A hierarchical method to improve the productivity of a multi-head surface mounting machine. *Proceedings of the IEEE International Conference on Robotics and Automation.*

Leu, M. C., Wong, H., Ji, Z., 1993, Planning of component placement/insertion sequence and feeder setup in PCB assembly using genetic algorithms. *Transactions of the ASME*, Vol. 115, 424-432.

Magyar, G., Johnsson, M., Nevalainen, O., 1999, On solving single machine optimization problems in electronic assembly. *Journal of Electronics Manufacturing*, Vol. 9, 249-267.

Michalewicz, Z., 1996, Genetic algorithms + Data structures = Evolution programs. *Springer.*

Mitchell, M., 1996, An introduction to genetic algorithms. *The MIT Press.*

Montgomery, D. C., 2001, Design and analysis of experiments. *John Wiley & Sons, Inc.*

Prasad, R. P., 1997, Surface mount technology: Principles and practices. *Chapman and Hall.*

Rawlins, G. J. E., 1991, Foundations of genetic algorithms. *Morgan Kaufmann Publishers.*

Reinelt, G., 1994, The traveling salesman – Computational solutions. *Springer-Verlag.*

Sakawa, M., 2002, Genetic algorithms and fuzzy multiobjective optimization. *Kluwer Academic Publishers.*

Sanchez, J. M., Priest, J. W., 1991, Optimal component-insertion sequence planning methodology for the semi automatic assembly of printed circuit boards. *Journal of Intelligent Manufacturing*, Vol. 2, 177-188.

Su, C., Ho, L., Fu, H., 1998, A novel tabu search approach to find the best placement sequence and magazine assignment in dynamic robotics assembly. *Integrated Manufacturing Systems*, Vol. 9, 366-376.

Su, Y. Srihari, K., 1996, Placement sequence identification using artificial neural networks in surface mount PCB assembly. *International Journal of Advanced Manufacturing Technology*, Vol. 11, 285-299.

Traister, J. E., 1990, Design guidelines for surface mount technology. *Academic Press, Inc.*

Van Laarhoven, P. J. M., Zijm, W. H. M., 1993, Production planning and numeric control in PCB assembly. *The International Journal of Flexible Manufacturing Systems*, Vol. 5, 187-207.

Wang, W., Nelson, P. C., Tirpak, T. M., 1999, Optimization of high-speed multi-station SMT placement machines using evolutionary algorithms. *IEEE Transactions on Electronics Packaging Manufacturing*, 1-10.

APPENDICES

APPENDIX A

MATLAB CODE FOR PLACEMENT SEQUENCE OPTIMIZATION

```
% Clear command window and remove variables from memory
clear
clc

% Read data from file
Y=dlmread('Assyinfo.txt');

% Prompt user to input Representation and Crossover methods
disp('Choose a Representation Scheme');
disp('Enter 1 for Path');
disp('Enter 2 for Ordinal');
disp('Enter 3 for Adjacency');
Representation = input('Enter number:');

if Representation == 1
   disp('Choose a Crossover operator');
   disp('Enter 1 for PMX');
   disp('Enter 2 for OX');
   disp('Enter 3 for CX');
   Crossover = input('Enter number:');
end

if Representation == 3
   disp('Choose a Crossover operator');
   disp('Enter 1 for Alternating Edges');
   disp('Enter 2 for Heuristic');
   Crossover = input('Enter number:');
```

```
end
% Promt user to input Crossover and Mutation rates
COProb = input('Enter Crossover Probability: ');
MUProb = input('Enter Mutation Probability: ');

% Start time
tic

% Path Representation
if Representation == 1
   Counter=0;
   Generation = 1;
   while Counter < 95
% Random generation of Initial Population
      if Generation == 1
         for i=1:100
            X(:,:,i)=randperm(10);
         end
      end

% Evaluation Function which calculates the distance travelled between 2 points for
each sequence
      for i=1:100
         for j=2:10
            Z(i,j-1) = sqrt((Y(X(1,j,i),1)-Y(X(1,j-1,i),1))^2+(Y(X(1,j,i),2)-Y(X(1,j-
1,i),2))^2);
         end
      end

% Calculating the total distance travelled for each sequence and also the
```

```
% grand total of all the distances
    for i=1:100
      Total(i)=sum(Z(i,:));
      TotalReci(i)=1/Total(i);
    end
    GrandTot = sum(TotalReci(:));
    Average(Generation) = mean(Total);
    GenPlot(Generation) = Generation;

% Calculating the probability of selection for each sequence
    for  i=1:100
      P(i)=TotalReci(i)/GrandTot;
    end

% Roulette wheel selection
    for i=1:100
      RandomNum = rand(1);
      K = 0;
      for j=1:100
        K = K + P(j);
        if K >= RandomNum
          NewX(:,:,i)=X(:,:,j);
          break
        end
      end
    end

% Randomly selecting individuals for Crossover
    K = 1;
    for i=1:100
```

```
        RandomNum = rand(1);
        if RandomNum < COProb
          CO(:,:,K) = NewX(:,:,i);
          COPos(K) = i;
          K = K + 1;
        end
      end

% Check if the number of chosen sequences are even or odd
      if rem(K,2) == 0
        RandomNum = rand(1);
        if RandomNum < .5
          CO(:,:,K-1) = [];
          COPos(K-1)=[];
          K = K-2;
        else
          COPos(K) = round((rand(1)*99)+1);
          CO(:,:,K) = NewX(:,:,COPos(K));
        end
      end

% PMX Crossover operator
      if Crossover == 1
        CONew = CO;
        if rem(K,2) ~= 0
          K = K-1;
        end
        RandomCO = randperm(K);
        for i=1:2:K
          P1 = round(rand(1)*8 + 1);
```

```
P2 = round(rand(1)*8 + 1);
if P1==1
   P1=P1+1;
end
if P2==1
   P2=P2+1;
end
for j=min(P1,P2):max(P1,P2)
   Temp = CO(:,j,RandomCO(i));
   CONew(:,j,RandomCO(i)) = CO(:,j,RandomCO(i+1));
   CONew(:,j,RandomCO(i+1)) = Temp;
end
for n=1:(min(P1,P2)-1)
   m=min(P1,P2);
   while m <= max(P1,P2)
      if CONew(:,n,RandomCO(i)) == CONew(:,m,RandomCO(i))
         CONew(:,n,RandomCO(i)) = CONew(:,m,RandomCO(i+1));
         m=min(P1,P2);
      else
         m=m+1;
      end
   end
end
for n=(max(P1,P2)+1):10
   m=min(P1,P2);
   while m <= max(P1,P2)
      if CONew(:,n,RandomCO(i)) == CONew(:,m,RandomCO(i))
         CONew(:,n,RandomCO(i)) = CONew(:,m,RandomCO(i+1));
         m=min(P1,P2);
      else
```

```
            m=m+1;
          end
        end
      end
      for n=1:(min(P1,P2)-1)
        m=min(P1,P2);
        while m <= max(P1,P2)
          if CONew(:,n,RandomCO(i+1)) == CONew(:,m,RandomCO(i+1))
            CONew(:,n,RandomCO(i+1)) = CONew(:,m,RandomCO(i));
            m=min(P1,P2);
          else
            m=m+1;
          end
        end
      end
      for n=(max(P1,P2)+1):10
        m=min(P1,P2);
        while m <= max(P1,P2)
          if CONew(:,n,RandomCO(i+1)) == CONew(:,m,RandomCO(i+1))
            CONew(:,n,RandomCO(i+1)) = CONew(:,m,RandomCO(i));
            m=min(P1,P2);
          else
            m=m+1;
          end
        end
      end
    end
  end

% OX Crossover operator
```

```
if Crossover == 2
   CONew = CO;
   if rem(K,2) ~= 0
      K = K-1;
   end
   RandomCO = randperm(K);
   for i=1:2:K
      P1 = round(rand(1)*8 + 1);
      P2 = round(rand(1)*8 + 1);
      if P1==1
         P1=P1+1;
      end
      if P2==1
         P2=P2+1;
      end
      q=1;
      for n=(max(P1,P2)+1):10
         Temp(q)=CO(:,n,RandomCO(i+1));
         q=q+1;
      end
      for n=1:max(P1,P2)
         Temp(q)=CO(:,n,RandomCO(i+1));
         q=q+1;
      end
      for n=1:10
         for m=min(P1,P2):max(P1,P2)
            if Temp(n) == CO(:,m,RandomCO(i))
               Temp(n) = 0;
            end
         end
```

```
end
d=1;
for n=1:10
   if Temp(n)~= 0
      Temp1(d)=Temp(n);
      d=d+1;
   end
end
q=1;
for n=(max(P1,P2)+1):10
   CONew(:,n,RandomCO(i))=Temp1(q);
   q=q+1;
end
for n=1:(min(P1,P2)-1)
   CONew(:,n,RandomCO(i))=Temp1(q);
   q=q+1;
end
q=1;
for n=(max(P1,P2)+1):10
   Temp(q)=CO(:,n,RandomCO(i));
   q=q+1;
end
for n=1:max(P1,P2)
   Temp(q)=CO(:,n,RandomCO(i));
   q=q+1;
end
for n=1:10
   for m=min(P1,P2):max(P1,P2)
      if Temp(n) == CO(:,m,RandomCO(i+1))
         Temp(n) = 0;
```

```
            end
          end
        end
        d=1;
        for n=1:10
          if Temp(n)~= 0
            Temp1(d)=Temp(n);
            d=d+1;
          end
        end
        q=1;
        for n=(max(P1,P2)+1):10
          CONew(:,n,RandomCO(i+1))=Temp1(q);
          q=q+1;
        end
        for n=1:(min(P1,P2)-1)
          CONew(:,n,RandomCO(i+1))=Temp1(q);
          q=q+1;
        end
      end
    end

% CX Crossover operator
    if Crossover == 3
      CONew = CO;
      CONew(:,:,:)=0;
      if rem(K,2) ~= 0
        K = K-1;
      end
      RandomCO = randperm(K);
```

```
for i=1:2:K
  q=1;
  CONew(:,q,RandomCO(i))=CO(:,q,RandomCO(i));
  for n=1:10
    for m=1:10
      if CO(:,q,RandomCO(i+1)) == CO(:,m,RandomCO(i))
        if CONew(:,m,RandomCO(i))~= 0
          break
          break
        else
          CONew(:,m,RandomCO(i)) = CO(:,m,RandomCO(i));
          q=m;
          break
        end
      end
    end
  end
  for n=1:10
    if CONew(:,n,RandomCO(i))==0
      CONew(:,n,RandomCO(i))=CO(:,n,RandomCO(i+1));
    end
  end
end
for i=1:2:K
  q=1;
  CONew(:,q,RandomCO(i+1))=CO(:,q,RandomCO(i+1));
  for n=1:10
    for m=1:10
      if CO(:,q,RandomCO(i)) == CO(:,m,RandomCO(i+1))
        if CONew(:,m,RandomCO(i+1))~= 0
```

```
                    break
                    break
                else
                    CONew(:,m,RandomCO(i+1)) = CO(:,m,RandomCO(i+1));
                    q=m;
                    break
                end
            end
        end
    end
    for n=1:10
        if CONew(:,n,RandomCO(i+1))==0
            CONew(:,n,RandomCO(i+1))=CO(:,n,RandomCO(i));
        end
    end
  end
end

% Putting the Crossovered Elements back into the population
  for i=1:K
    NewX(:,:,COPos(RandomCO(i))) = CONew(:,:,RandomCO(i));
  end

% Randomly choosing bits and performing Mutation
  for i=1:1000
    RandomNum = rand(1);
    if RandomNum < MUProb
      if i >= 10 && rem(i,10)~=0
        quotient = ((i-rem(i,10))/10)+1;
        Coloumn=rem(i,10);
```

```
        elseif i>= 10 && rem(i,10) == 0
            quotient = i/10;
            Coloumn=10;
        else
            quotient = 1;
            Coloumn=rem(i,10);
        end
        NewElement = round((rand(1)*9 + 1));
        for j=1:10
            if NewX(:,j,quotient) == NewElement
                NewX(:,j,quotient) = NewX(:,Coloumn,quotient);
                NewX(:,Coloumn,quotient) = NewElement;
            end
        end
    end
end
X=NewX;

Counter=0;
for i=1:6
    for j=1:100
        if X(:,:,i) == X(:,:,j)
            Counter = Counter+1;
            LeastPos=j;
        end
    end
    if Counter >= 95
        break
        break
    else
```

```
            Counter=0;
          end
        end
        Generation = Generation+1;
    end
end

% Ordinal Representation
if Representation == 2
    Reference = [1 2 3 4 5 6 7 8 9 10];
    Counter=0;
    Generation = 1;

    while Counter < 95
% Generating list L of references on first run
      if Generation == 1
                          for i=1:100
          for j=1:10
            L(:,j,i)=round(rand(1)*(10-j) + 1);
          end
                          end
      end

% Decoding the reference list to actual tours
      for i=1:100
        RefCopy=Reference;
        for j=1:10
          X(:,j,i)= RefCopy(L(:,j,i));
          RefCopy(L(:,j,i))=[];
        end
```

```
    end

% Evaluation Function
    for i=1:100
      for j=2:10
        Z(i,j-1) = sqrt((Y(X(1,j,i),1)-Y(X(1,j-1,i),1))^2+(Y(X(1,j,i),2)-Y(X(1,j-
1,i),2))^2);
      end
    end

% Calculating the total distance travelled for each sequence and also the
% grand total of all the distances
    for i=1:100
      Total(i)=sum(Z(i,:));
      TotalReci(i)=1/Total(i);
    end
    GrandTot = sum(TotalReci(:));
    Average(Generation) = mean(Total);
    GenPlot(Generation) = Generation;

% Calculating the probability of selection for each sequence
    for  i=1:100
      P(i)=TotalReci(i)/GrandTot;
    end

% Roulette wheel selection
    for i=1:100
      RandomNum = rand(1);
      K = 0;
      for j=1:100
```

```
            K = K + P(j);
            if K >= RandomNum
               NewL(:,:,i)=L(:,:,j);
               break
            end
         end
      end

% Randomly selecting individuals for Crossover
      K = 1;
      for i=1:100
         RandomNum = rand(1);
         if RandomNum < COProb
            CO(:,:,K) = NewL(:,:,i);
            COPos(K) = i;
            K = K + 1;
         end
      end

% Check if the number of chosen sequences are even or odd
      if rem(K,2) == 0
         RandomNum = rand(1);
         if RandomNum < .5
            CO(:,:,K-1) = [];
            COPos(K-1)=[];
            K = K-2;
         else
            COPos(K) = round((rand(1)*99)+1);
            CO(:,:,K) = NewL(:,:,COPos(K));
         end
```

```
    end

% Classical Crossover operator
    CONew = CO;
    if rem(K,2) ~= 0
      K = K-1;
    end
    RandomCO = randperm(K);
    for i=1:2:K
      P1 = round(rand(1)*7 + 2);
      for j=P1:10
        CONew(:,j,RandomCO(i)) = CO(:,j,RandomCO(i+1));
        CONew(:,j,RandomCO(i+1)) = CO(:,j,RandomCO(i));
        NewL(:,j,COPos(RandomCO(i)))=CO(:,j,RandomCO(i+1));
        NewL(:,j,COPos(RandomCO(i+1)))=CO(:,j,RandomCO(i));
      end
    end

% Randomly selecting individual bits and performing Mutation
    for i=1:1000
      RandomNum = rand(1);
      if RandomNum < MUProb
        if i >= 10 && rem(i,10)~=0
          quotient = ((i-rem(i,10))/10)+1;
          Coloumn=rem(i,10);
        elseif i>= 10 && rem(i,10) == 0
          quotient = i/10;
          Coloumn=10;
        else
          quotient = 1;
```

```
            Coloumn=rem(i,10);
        end
        NewL(:,Coloumn,quotient) = round((rand(1)*(10-Coloumn) + 1));
      end
    end
    L=NewL;

    Counter=0;
    for i=1:6
      for j=1:100
        if L(:,:,i) == L(:,:,j)
          Counter = Counter+1;
          LeastPos=j;
        end
      end
      if Counter >= 95
        break
        break
      else
        Counter=0;
      end
    end
    Generation = Generation+1;
  end
end

% Adjacency Representation
if Representation == 3
  Counter=0;
  Generation = 1;
```

```
while Counter<95
  if Generation == 1
    for i=1:100
      X(:,:,i)=randperm(10);
    end
  end

  for i=1:100
    for j=2:10
      Z(i,j-1) = sqrt((Y(X(1,j,i),1)-Y(X(1,j-1,i),1))^2+(Y(X(1,j,i),2)-Y(X(1,j-
1,i),2))^2);
    end
  end

% Calculating the total distance travelled for each sequence and also the
% grand total of all the distances
  for i=1:100
    Total(i)=sum(Z(i,:));
    TotalReci(i)=1/Total(i);
  end
  GrandTot = sum(TotalReci(:));
  Average(Generation) = mean(Total);
  GenPlot(Generation) = Generation;

% Calculating the probability of selection for each sequence
  for  i=1:100
    P(i)=TotalReci(i)/GrandTot;
  end
```

```
% Roulette wheel selection
    for i=1:100
      RandomNum = rand(1);
      K = 0;
      for j=1:100
        K = K + P(j);
        if K >= RandomNum
          NewX(:,:,i)=X(:,:,j);
          break
        end
      end
    end

% Randomly selecting individuals for Crossover
    K = 1;
    for i=1:100
      RandomNum = rand(1);
      if RandomNum < COProb
        CO(:,:,K) = NewX(:,:,i);
        COPos(K) = i;
        K = K + 1;
      end
    end

% Check if the number of chosen sequences are even or odd
    if rem(K,2) == 0
      RandomNum = rand(1);
      if RandomNum < .5
        CO(:,:,K-1) = [];
        COPos(K-1)=[];
```

```
            K = K-2;
        else
            COPos(K) = round((rand(1)*99)+1);
            CO(:,:,K) = NewX(:,:,COPos(K));
        end
    end

% Converting from Path to Adjacent
    for i=1:100
        for t=1:size(X,2)-1
                    Adj(:,X(:,t,i),i)=X(:,t+1,i);
            end
            Adj(:,X(:,size(X,2),i),i)=X(1,1,i);
    end
    Adj1=Adj;
    for i=1:size(CO,3)
        for t=1:size(CO,2)-1
            COAdj(:,CO(:,t,i),i)=CO(:,t+1,i);
        end
        COAdj(:,CO(:,size(CO,2),i),i)=CO(1,1,i);
    end

% Perform Alternating Edges Crossover
    if Crossover == 1
        CONew = zeros(size(COAdj));
        if rem(K,2) ~= 0
            K = K-1;
        end
        RandomCO = randperm(K);
        for i=1:K
```

```
Cols=size(COAdj,2);
                    StartIndex=round(rand(1)*(Cols-1) + 1);
                    ChangingIndex=StartIndex;
                    CONum=0;
                    VisitedList=zeros(1,Cols);
                    VisitedList(ChangingIndex)=1;
                    Comp=1;

                    while Comp<Cols
  if rem(i,2)==0
    Seed = i-2+CONum+1;
  else
    Seed = i-1+CONum+1;
  end

NewComp=COAdj(:,ChangingIndex,RandomCO(:,Seed));

                        if VisitedList(NewComp)==1
                            AllowedList=zeros(1,Cols-Comp);
                            z=1;
                            for t=1:size(VisitedList,2)
                                if VisitedList(t)==0
                                    AllowedList(z)=t;
                                    z=z+1;
                                end
                            end

NewComp=AllowedList(round(rand(1)*(size(AllowedList,2)-1)+1));
                        end
```

```
                                CONew(:,ChangingIndex,i)=NewComp;
                                ChangingIndex=NewComp;
                                Comp=Comp+1;
                                VisitedList(ChangingIndex)=1;
                                CONum=1-CONum;
                            end
                            CONew(:,ChangingIndex,i)=StartIndex;
        end

        for i=1:K
            Adj(:,:,COPos(RandomCO(i))) = CONew(:,:,i);
        end
    end

% Perform Heuristic Crossover
    if Crossover == 2
        CONew = zeros(size(COAdj));
        if rem(K,2) ~= 0
            K = K-1;
        end
        RandomCO = randperm(K);
        for i=1:K
            Cols=size(COAdj,2);
                            StartIndex=round(rand(1)*(Cols-1) + 1);
                            ChangingIndex=StartIndex;
                            CONum=0;
                            VisitedList=zeros(1,Cols);
                            VisitedList(ChangingIndex)=1;
                            Comp=1;
            if rem(i,2)==0
```

```
        Seed = i-1;
    else
        Seed = i;
    end

    while Comp<Cols

        Temp1 = COAdj(:,ChangingIndex,RandomCO(:,Seed));
        Temp2 = COAdj(:,ChangingIndex,RandomCO(:,Seed+1));

        Dist1 = sqrt((Y(ChangingIndex,1)-
Y(Temp1,1))^2+(Y(ChangingIndex,2)-Y(Temp1,2))^2);
        Dist2 = sqrt((Y(ChangingIndex,1)-
Y(Temp2,1))^2+(Y(ChangingIndex,2)-Y(Temp2,2))^2);

        if Dist1 < Dist2
            NewComp=COAdj(:,ChangingIndex,Seed);
        else
            NewComp=COAdj(:,ChangingIndex,Seed+1);
        end

                    if VisitedList(NewComp)==1
                        AllowedList=zeros(1,Cols-Comp);
                        z=1;
                        for t=1:size(VisitedList,2)
                            if VisitedList(t)==0
                                AllowedList(z)=t;
                                z=z+1;
                            end
                        end
```

```
        NewComp=AllowedList(round(rand(1)*(size(AllowedList,2)-1)+1));
                                    end

                                    CONew(:,ChangingIndex,i)=NewComp;
                                    ChangingIndex=NewComp;
                                    Comp=Comp+1;
                                    VisitedList(ChangingIndex)=1;
                                    CONum=1-CONum;
                                end
                                CONew(:,ChangingIndex,i)=StartIndex;
            end

            for i=1:K
               Adj(:,:,COPos(RandomCO(i))) = CONew(:,:,i);
            end
        end

% Converting from Adjacent to Path
        for i=1:100
           index=1;
                    X(:,:,i) = zeros(size(Adj(:,:,i)));
                    X(:,1,i) = 1;
                    for j=2:10
                            X(:,j,i) = Adj(index);
                            index = X(:,j,i);
                    end
        end

% Randomly choosing bits and performing Mutation
```

```
for i=1:1000
   RandomNum = rand(1);
   if RandomNum < MUProb
      if i >= 10 && rem(i,10)~=0
         quotient = ((i-rem(i,10))/10)+1;
         Coloumn=rem(i,10);
      elseif i>= 10 && rem(i,10) == 0
         quotient = i/10;
         Coloumn=10;
      else
         quotient = 1;
         Coloumn=rem(i,10);
      end
      NewElement = round((rand(1)*9 + 1));
      for j=1:10
         if NewX(:,j,quotient) == NewElement
            NewX(:,j,quotient) = NewX(:,Coloumn,quotient);
            NewX(:,Coloumn,quotient) = NewElement;
         end
      end
   end
end
X=NewX;

Counter=0;
for i=1:6
   for j=1:100
      if Adj(:,:,i) == Adj(:,:,j)
         Counter = Counter+1;
         LeastPos=j;
```

```
            end
        end
        if Counter >= 95
            break
            break
        else
            Counter=0;
        end
    end
    Generation = Generation+1;
  end
end

% Stop time
TimeElapsed = toc;

% Storing results in an array
Results(1)=Generation-1;
Results(2)=TimeElapsed;
Results(3)=Total(LeastPos);

% Generating plots
for i=1:10
  XAxis(i)=Y((X(1,i,LeastPos)),1);
  YAxis(i)=Y((X(1,i,LeastPos)),2);
end
subplot(1,2,1);plot(XAxis,YAxis,'b-s')
subplot(1,2,2);plot(GenPlot,Average)
```

APPENDIX B

ANOVA

A few researchers in the past have used multi-factor ANOVA for analyzing similar problems. But the normal probability plots for this research show that the data is highly skewed and thus non-parametric tests are better suited to analyze this data. But for the sake of completeness, multi factor ANOVA was performed on each of the response variables and the results and conclusions are presented within this appendix. All results in this appendix should be interpreted carefully due to the deviations observed from the normality assumptions underlying ANOVA.

Distance Analysis

Table B1 shows the ANOVA for the response variable Distance. Method, MR, two-factor interaction MR-method and three factor interactions MR-method-CR have p-values less than 0.05. The same analysis was applied for the significant factors in order to eliminate the variability arising from the nuisance sources. Table B2 shows the new ANOVA table, confirming that the factors are indeed significant.

Table B1: ANOVA Summary for response variable Distance for all factors

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	37420709	18710354	339.0791	0.0000000
Method	5	25190972	5038194	91.3049	0.0000000
CR	10	570237	57024	1.0334	0.4118971
MR:Method	10	8241101	824110	14.9350	0.0000000
MR:CR	20	1420176	71009	1.2869	0.1753868
Method:CR	50	3476125	69522	1.2599	0.1041129
MR:Method:CR	100	7051428	70514	1.2779	0.0336016
Residuals	4752	262214911	55180		

Table B2: ANOVA Summary for response variable Distance for significant factors

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	37420709	18710354	339.0791	0.00000000
Method	5	25190972	5038194	91.3049	0.00000000
MR:Method	10	8241101	824110	14.9350	0.00000000
MR:Method:CR	180	12517966	69544	1.2603	0.01171657
Residuals	4752	262214911	55180		

A Tukey test was performed on significant factors to identify the level(s) which are significantly different. Tables B3 and B4 and Figures B1 and B2 show the results of the Tukey test for significant factors MR and Method respectively.

Table B3: Summary of Tukey test comparing MR for response variable Distance

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method
intervals excluding 0 are flagged by '****'

	Estimate	Std.Error	Lower Bound	Upper Bound		
1-2	154.0	8.18	135.0	173.0	****	MR1 = 0.0025
1-3	204.0	8.18	185.0	224.0	****	MR2 = 0.005
2-3	50.3	8.18	31.2	69.5	****	MR3 = 0.0075

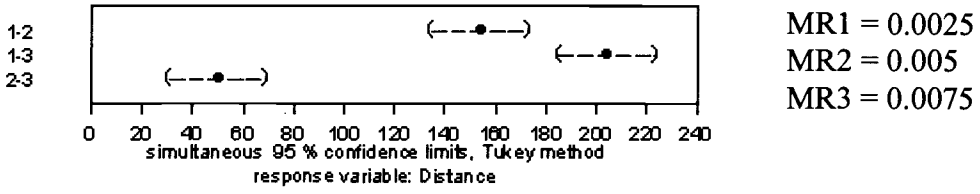


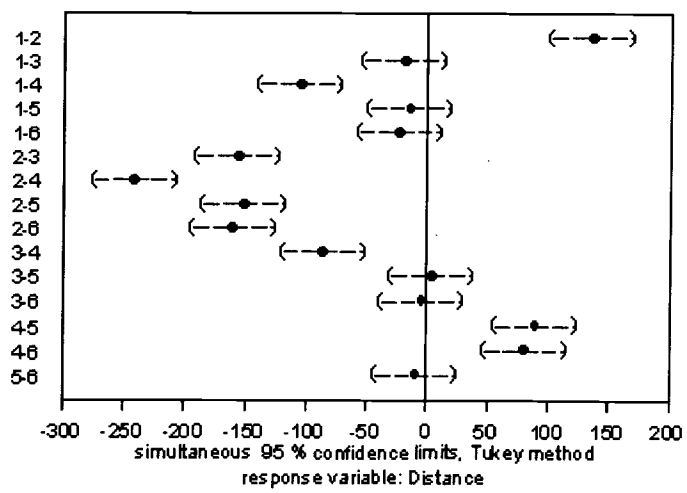
Figure B1: Tukey test comparing MR for response variable Distance

Table B4: Summary of Tukey test comparing Methods for response variable Distance

95 % simultaneous confidence intervals for specified linear combinations, by the Tukey method

intervals excluding 0 are flagged by '****'

	Estimate	Std.Error	Lower Bound	Upper Bound	
1-2	137.00	11.6	104.0	170.0	****
1-3	-18.60	11.6	-51.6	14.3	
1-4	-104.00	11.6	-137.0	-71.2	****
1-5	-13.80	11.6	-46.8	19.1	
1-6	-22.70	11.6	-55.6	10.3	
2-3	-155.00	11.6	-188.0	-122.0	****
2-4	-241.00	11.6	-274.0	-208.0	****
2-5	-151.00	11.6	-184.0	-118.0	****
2-6	-159.00	11.6	-192.0	-127.0	****
3-4	-85.50	11.6	-119.0	-52.6	****
3-5	4.78	11.6	-28.2	37.8	
3-6	-4.05	11.6	-37.0	28.9	
4-5	90.30	11.6	57.3	123.0	****
4-6	81.50	11.6	48.5	114.0	****
5-6	-8.83	11.6	-41.8	24.1	



Method 1 = Path + PMX
Method 2 = Path + OX
Method 3 = Path + CX
Method 4 = Ordinal + Classical
Method 5 = Adjacency + Alternating edges
Method 6 = Adjacency + Heuristic

Figure B2: Tukey test comparing Method for response variable Distance

Generation Analysis

Table B5 shows the ANOVA for a response variable Generation. Method, MR, two-factor interactions MR-method and Method-CR and three factor interactions MR-method-CR have p-values less than 0.05. The same analysis was applied for the significant factors and is shown in Table B6, confirming that the factors are indeed significant.

Table B5: ANOVA Summary for response variable Generation for all factors

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	429451906942	214725953471	798.9736	0.0000000
Method	5	120021699168	24004339834	89.3177	0.0000000
CR	10	2962934839	296293484	1.1025	0.3558626
MR:Method	10	229188463044	22918846304	85.2787	0.0000000
MR:CR	20	5910203141	295510157	1.0996	0.3414847
Method:CR	50	33537094230	670741885	2.4958	0.0000000
MR:Method:CR	100	64646056688	646460567	2.4054	0.0000000
Residuals	4752	1277110626602	268752236		

Table B6: ANOVA Summary for response variable Generation for significant factors

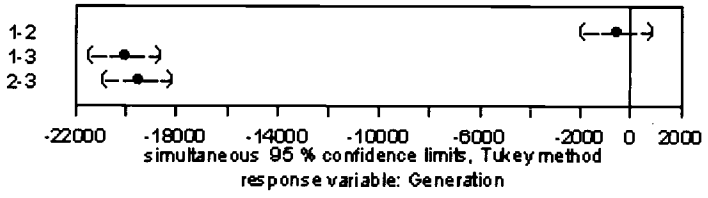
	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	429451906942	214725953471	798.9736	0.000000e+000
Method	5	120021699168	24004339834	89.3177	0.000000e+000
MR:Method	10	229188463044	22918846304	85.2787	0.000000e+000
Method:CR	60	36500029069	608333818	2.2635	1.146583e-007
MR:Method:CR	120	70556259829	587968832	2.1878	3.400000e-012
Residuals	4752	1277110626602	268752236		

A Tukey test was performed on significant factors to identify the level(s) which are significantly different. Tables B7 and B8 and Figures B3 and B4 show the results of the Tukey test for significant factors MR and Method respectively.

Table B7: Summary of Tukey test comparing MR for response variable Generation

95 % simultaneous confidence intervals for specified linear combinations, by the Tukey method
intervals excluding 0 are flagged by '****'

	Estimate	Std.Error	Lower Bound	Upper Bound	
1-2	-529	571	-1870	809	
1-3	-20000	571	-21400	-18700	****
2-3	-19500	571	-20800	-18200	****



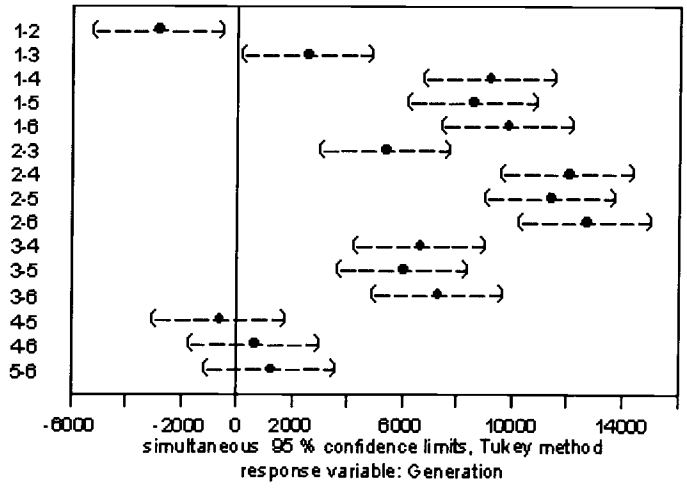
MR1 = 0.0025
MR2 = 0.005
MR3 = 0.0075

Figure B3: Tukey test comparing MR for response variable Generation

Table B8: Summary of Tukey test comparing Method for response variable Generation

95 % simultaneous confidence intervals for specified linear combinations, by the Tukey method
intervals excluding 0 are flagged by '****'

	Estimate	Std.Error	Lower Bound	Upper Bound	
1-2	-2850	807	-5150	-546	****
1-3	2540	807	243	4850	****
1-4	9200	807	6900	11500	****
1-5	8590	807	6290	10900	****
1-6	9850	807	7550	12200	****
2-3	5390	807	3090	7690	****
2-4	12000	807	9750	14300	****
2-5	11400	807	9130	13700	****
2-6	12700	807	10400	15000	****
3-4	6650	807	4350	8960	****
3-5	6040	807	3740	8340	****
3-6	7310	807	5010	9610	****
4-5	-612	807	-2910	1690	
4-6	653	807	-1650	2950	
5-6	1270	807	-1040	3570	



Method 1 = Path + PMX
Method 2 = Path + OX
Method 3 = Path + CX
Method 4 = Ordinal + Classical
Method 5 = Adjacency + Alternating edges
Method 6 = Adjacency + Heuristic

Figure B4: Tukey test comparing Method for response variable Generation

Time Analysis

Table B9 shows the ANOVA for a response variable Time. Method, MR, two-factor interactions MR-method, Method-CR and MR-CR and three factor interactions MR-method-CR have p-values less than 0.05. The same analysis was applied for the significant factors and is shown in Table B10, confirming that the factors are indeed significant.

Table B9: ANOVA Summary for response variable Time for all factors

	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	866186988	433093494	856.1999	0.00000000
Method	5	169557324	33911465	67.0409	0.00000000
CR	10	8482375	848238	1.6769	0.07999447
MR:Method	10	338086905	33808690	66.8378	0.00000000
MR:CR	20	16470133	823507	1.6280	0.03815025
Method:CR	50	72474857	1449497	2.8656	0.00000000
MR:Method:CR	100	139450090	1394501	2.7568	0.00000000
Residuals	4752	2403714779	505832		

Table B10: ANOVA Summary for response variable Time for significant factors

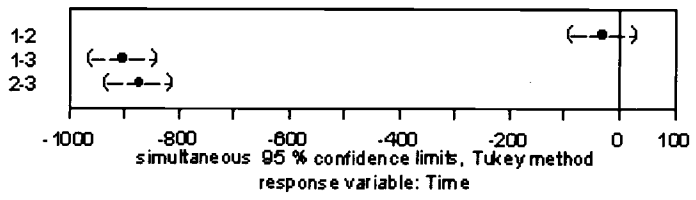
	Df	Sum of Sq	Mean Sq	F Value	Pr(F)
MR	2	866186988	433093494	856.1999	0.00000000
Method	5	169557324	33911465	67.0409	0.00000000
MR:Method	10	338086905	33808690	66.8378	0.00000000
MR:CR	30	24952508	831750	1.6443	0.01492652
Method:CR	50	72474857	1449497	2.8656	0.00000000
MR:Method:CR	100	139450090	1394501	2.7568	0.00000000
Residuals	4752	2403714779	505832		

A Tukey test was performed on significant factors to identify the level(s) which are significantly different. Tables B11 and B12 and Figures B5 and B6 show the results of the Tukey test for significant factors MR and Method respectively.

Table B11: Summary of Tukey test comparing MR for response variable Time

```
95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method
intervals excluding 0 are flagged by '****'

    Estimate Std.Error Lower Bound Upper Bound
1-2      -30      24.8       -88.1          28
1-3     -902      24.8      -960.0        -844 ****
2-3     -872      24.8      -930.0        -814 ****
```



MR1 = 0.0025
MR2 = 0.005
MR3 = 0.0075

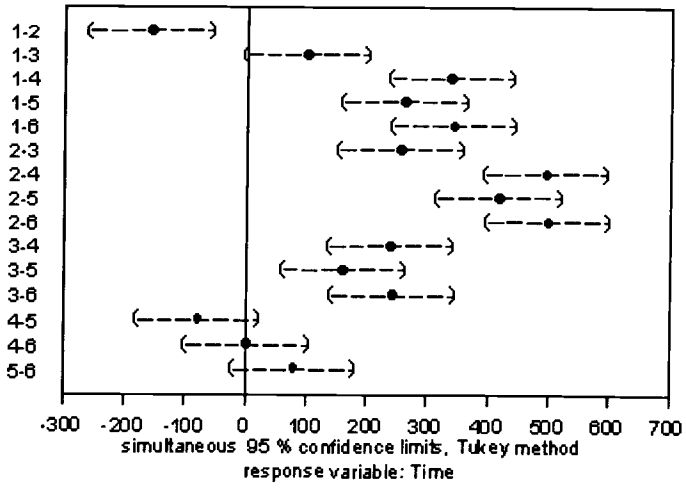
Figure B5: Tukey test comparing MR for response variable Time

Table B12: Summary of Tukey test comparing Method for response variable Time

95 % simultaneous confidence intervals for specified linear combinations, by the Tukey method

intervals excluding 0 are flagged by '****'

	Estimate	Std.Error	Lower Bound	Upper Bound	
1-2	-156.00	35	-256.00	-55.9	****
1-3	101.00	35	1.23	201.0	****
1-4	341.00	35	241.00	441.0	****
1-5	263.00	35	164.00	363.0	****
1-6	345.00	35	245.00	444.0	****
2-3	257.00	35	157.00	357.0	****
2-4	497.00	35	397.00	596.0	****
2-5	419.00	35	319.00	519.0	****
2-6	500.00	35	400.00	600.0	****
3-4	240.00	35	140.00	340.0	****
3-5	162.00	35	62.60	262.0	****
3-6	243.00	35	144.00	343.0	****
4-5	-77.40	35	-177.00	22.4	
4-6	3.66	35	-96.20	103.0	
5-6	81.10	35	-18.80	181.0	



Method 1 = Path + PMX
Method 2 = Path + OX
Method 3 = Path + CX
Method 4 = Ordinal + Classical
Method 5 = Adjacency + Alternating edges
Method 6 = Adjacency + Heuristic

Figure B6: Tukey test comparing Method for response variable Time

Conclusions

For response variable distance, mutation rate and method were the significant factors. A Tukey test comparing the different levels of mutation rates shows that all three levels are significantly different from each other. Also, from Figure B1, it can be seen that between mutation rates of 0.0025-0.005 and of 0.0025-0.0075 there is a significant difference in the means, but between mutation rates 0.005-0.0075 the difference is closer to zero. Comparing the different levels of methods, it was found that methods 2 (path + order) and 4 (ordinal + classical) were significantly different from methods 1, 3, 5 and 6. Figure B2 shows that method 2 (path + order) performed significantly better compared to all the other methods and method 4 (ordinal + classical) was the worst. From the interaction plots, shown in Figure 15, it can again be inferred that the method 2 (path + order) performed better than the rest, but the improvement in the solution with an increase in mutation rate was not as significant as was with the other methods.

Again for response variables generation and time, mutation rate and method were the significant factors. Tukey test for both these responses yielded similar results. No significant difference was found between mutation rates of 0.0025 and 0.005, but there was a significant difference when rates 0.0025 and 0.005 were compared with 0.0075. Also, methods 4 (ordinal + classical), 5 (adjacency + alternating edges) and 6 (adjacency + heuristic) were not significantly different. The interaction plots, Figures 27 and 36, show that the number of generations and time increases drastically for the first three methods when the mutation rate changes from 0.005 to 0.0075. For all the methods there is not a significant increase in the number of generations and time taken when mutation rate is increased from 0.0025-0.005.