5-31-1991

Genetic algorithm for solving printed circuit board assembly planning problems

Hermean Wong
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Genetic Algorithm for Solving Printed Circuit Board Assembly Planning Problems

by
Hermean Wong

ABSTRACT

The application of the genetic algorithm for solving the planning problems for various printed circuit board assembly machines is presented. The genetic algorithm finds the sequence of component placement/insertion and arrangement of feeders simultaneously, for achieving the shortest assembly time. Three types of assembly planning problems are modeled such that they can be solved by the genetic algorithm. The algorithm uses links (parents) to represent possible solutions and applies genetic operators to generate new links (offspring) in an iterative procedure to obtain the optimal solution. Some examples are provided to illustrate the solutions generated by the genetic algorithm, and these solutions are compared with those from other planning methods. A changing operation rate method is presented for the algorithm improvement.

Genetic Algorithm for Solving Printed Circuit Board Assembly Planning Problems

by
Hermean Wong

A Thesis
Submitted to the Faculty of the Graduate Division of the
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science
Department of Mechanical and Industrial Engineering,
May 1991

APPROVAL PAGE

Genetic Algorithm for Solving Printed Circuit Board Assembly Planning Problems

by

Hermean Wong

Dr. Ming C. Leu, Thesis Advisor
Professsor, Department of Mechanical and Industrial Engineering
Sponsored Chair in Manufacturing/Productivity
N.J.I.T.

Dr. Nouri H. Levy, Committee Member
Associate Professor, Department of Mechanical and Industrial Engineering
N.J.I.T.

Dr. MengChu Zhou, Committee Member
Assistant Professor, Department of Electrical and Computer Engineering,
N.J.I.T.

BIOGRAPHICAL SKETCH

Author : Hermean Wong

Degree : Master of Science in Mechanical Engineering

Date : May, 1991

Date of Birth :

Place of Birth :

Undergraduate and Graduate Education :

- Master of Science in Mechanical Engineering, New Jersey Institute of Technology, Newark, NJ, 1991
- Bachelor of Science in Mechanical Engineering, National Taiwan University, Taipei, Taiwan, R.O.C., 1986

Major : Mechanical Engineering

Presentations and Publications:

- Ji, Z., Ming C. Leu, and Hermean Wong. "Application of Linear Assignment Model for Planning of Robotic Assembly of Printed Circuit Board", Robotics and Automation Research Program Technical Report 022, New Jersey Institute of Technology, Newark, NJ, 1991
- Wong, Hermean. "Application of Genetic Algorithm for PCB Assembly Planning", ASME Graduate Student Conference, New Jersey Institute of Technology, Newark, NJ, Apr. 1991

ACKNOWLEDGEMENT

I will always remain indebted to my advisor, Dr. Ming C. Leu, whose support helped the development of my graduate career. His constant search for new creative ideas, his insistence on practical applications, his constructive criticism and his incessant energy for research served as a role model and created a tremendous impetus for me to complete my research. I will try to use these values in my future career.

I'd like to express my greatest gratitude to my dear wife. The many sacrifices she made for my education can not be repaid in any way.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Rapid development in electronics products, especially in the consumer electronics area, makes the manufacturing productivity of printed circuit boards (PCB) an important issue. Manufacturing of printed circuit boards involves a series of processes including board preparation, component placement/insertion, soldering, and inspection. The planning of placement/insertion of PCB components is the problem focused on in this thesis study.

Components used in printed circuit boards vary from resistors, capacitors, operational amplifiers, integrated circuits (IC), very large scale integrated circuits (VLSI) to all kinds of odd components. Nowadays, more and more efficient machines of different assembly operations are being designed for diversified components.

1.1 Literature Survey

Several mathematical models have been developed for some assembly operations. Ball and Magazine [1] modeled the component sequencing problem for a given feeder arrangement as a rural postman problem. They solved it with a heuristic which guarantees

the solution to be optimum if the assembly head movement is rectilinear. Realizing that devising an optimal arrangement for both feeder assignment and component placement/insertion sequence simultaneously is very difficult, they used the heuristic to separate the problems into two decoupled problems and solve each of them individually.

Leipälä and Nevalainen [2] studied the PCB assembly movement optimization problem for an insertion machine which inserts components sequentially on a PC board fixed to a moving X-Y table, with the components supplied by a moving feeder carriage. The tool head picks up components from a fixed location and places them at another fixed location. They modeled the components sequencing problem as a traveling salesperson problem and the permutation of components in feeders problem as a quadratic assignment problem. The quadratic assignment problem was NP-hard, therefore, a sub-optimal solution was solved with a heuristic approach.

Bard, Clayton and Feo [3] modeled the feeder setup and component placement problem for the Fuji CP II machine, whose operation is similar to that of the Panasert RH, except that the Fuji machine is used for surface mount components. They divided the problem into three sub-problems. One is the movement of the table which is solved with a traveling salesperson heuristic. The other two, the problem of assigning components to feeders and the problem of part retrieval sequence, are solved jointly with a Lagrangian

relaxation scheme for a given placement sequence.

Ji, Leu and Wong [4] decomposed the PCB assembly planning problem with fixed board and feeders into two assignment problems, each solved with the reduced matrix method (Hungarian method). Although the problem solved is similar to that solved by Ball and Magazine [1], the assembly head movement here is not limited to rectilinear.

A common drawback in all of the above algorithms is that each of these algorithms is applicable to only one type of PCB assembly planning problems (for one type of assembly machines). It is very desirable to have one algorithm capable of solving many types of PCB assembly planning problems. An excellent candidate for this purpose is applying the genetic algorithm approach - the focus of this thesis study.

A genetic algorithm uses a stochastic selection process to generate better solutions iteratively from a set of random solutions [5-7]. This approach is effective for solving complex optimization problems, and it is particularly attractive when no conventional techniques are available to solve a given problem. It has been used successfully to solve many difficult problems in job scheduling, machine learning, and pattern recognition [8-12].

1.2 Thesis Structure

The presentation of this thesis will include the following. A

brief overview of assembly of PCB components will first be presented. This will include characteristics of placement/insertion processes, components of various types, assembly machines to accommodate different components and board types. The discussion will lead to the main problem of this thesis study: planning the sequence of component placement/insertion and arrangement of feeders. The implementation of the genetic algorithm for planning PCB assembly for various types of assembly machines will be discussed in detail, with numerical results presented.

Chapter 2

Assembly of Printed Circuit Board Components

The components used in a printed circuit board can be divided mainly into through-hole components (THC) and surface mount components (SMC). In the insertion of through-hole components, the leads of the components are first bent on a special grid, then inserted through the holes into a PCB, and then bent again to prevent falling out during transport of the PCB. In the assembly of surface mount components, the components are placed on a PCB and then are secured to the board by using adhesive or soldering paste.

Leicht, Schraft and Wolf [13] classified the PCB components into ten different types based on shape of component housing; see Table 1. The second column in this table shows the delivery methods for the different types of components.

The assembly equipment for component placement/insertion may be flexible enough to handle many types of components, or may be dedicated to a particular type of components. Low-flexiblility assembly machines, with either single or multiple heads, are

Table 1. Component type for THI*1 and SMT*2

	Type of Component	Method of Delivery
1	Axial	Belt
2	Radial	Belt
3	DIP*3/SIP*4	Stick Magazine
4	Plug, Pins	Roll of Wire, Punch Strips
5	Special Components	Special Means
6	Chip, MELF*5	Belt, Loose Material
7	SOT*6	Belt
8	SOIC*7/SO*8/VSO*9	Belt, Stick Magazine
9	Chip carrier	Belt, Stick Magazine or Surface
10	Micro-Pack (TAB*10)	Film

*1 THI is acronym of Through-Hole Insertion
*2 SMT is acronym of Surface Mount Technique
*3 DIP is acronym of Dual In-line Package
*4 SIP is acronym of Single In-line Package
*5 MELF is acronym of Metal Electrode Leadless Faces
*6 SOT is acronym of Small Outline Transistor
*7 SOIC is acronym of Small Outline Integrated Circuit
*8 SO is acronym of Small Outline
*9 VSO is acronym of Very Small Outline
*10 TAB is acronym of Tape Automated Bonding

generally arranged in a series to form a line of placement stations. Each station places the components designated to it as the PCB moves down the assembly line. These machines are more suitable for products of large quantities. High-flexibility assembly machines often operate as stand-alone machines, with the board components placed/inserted sequentially by a same machine which uses either a moving X-Y table or a moving head system. Detailed information on various commercial assembly machines can be found in the assembly equipment directory publication [13].

When placing surface mount components with a moving X-Y table system, the tackiness of the adhesive or soldering paste may not be sufficient to hold components in place if the table motion is not properly controlled. In general, there are less incidents of component misalignment in stationary board machines, especially in the placement of fine-pitch surface mount components. The components are fed directly to the assembly tool head or picked up by the tool head from feeders at fixed locations.

An important problem in PCB assembly planning is to decide the order in which the components should be placed or inserted. In the case the components are directly fed to the assembly head with a time delay shorter than the traveling time of the PCB, the problem can be formulated as the traveling salesperson problem, because the head visits each component location once (see Figure 1). Most of the assembly machines for the through-hole components belong to this category; for example, Amistar AI-6448, Panasonic Panasert RT, Universal 6287A, and Universal 6241B. The planning problem in this case is to decide the sequence of the assembly head in visiting all component locations such that the total traveling distance or traveling time is minimized. This will be called the traveling salesperson problem (TSP) or the type one problem in the later chapters.

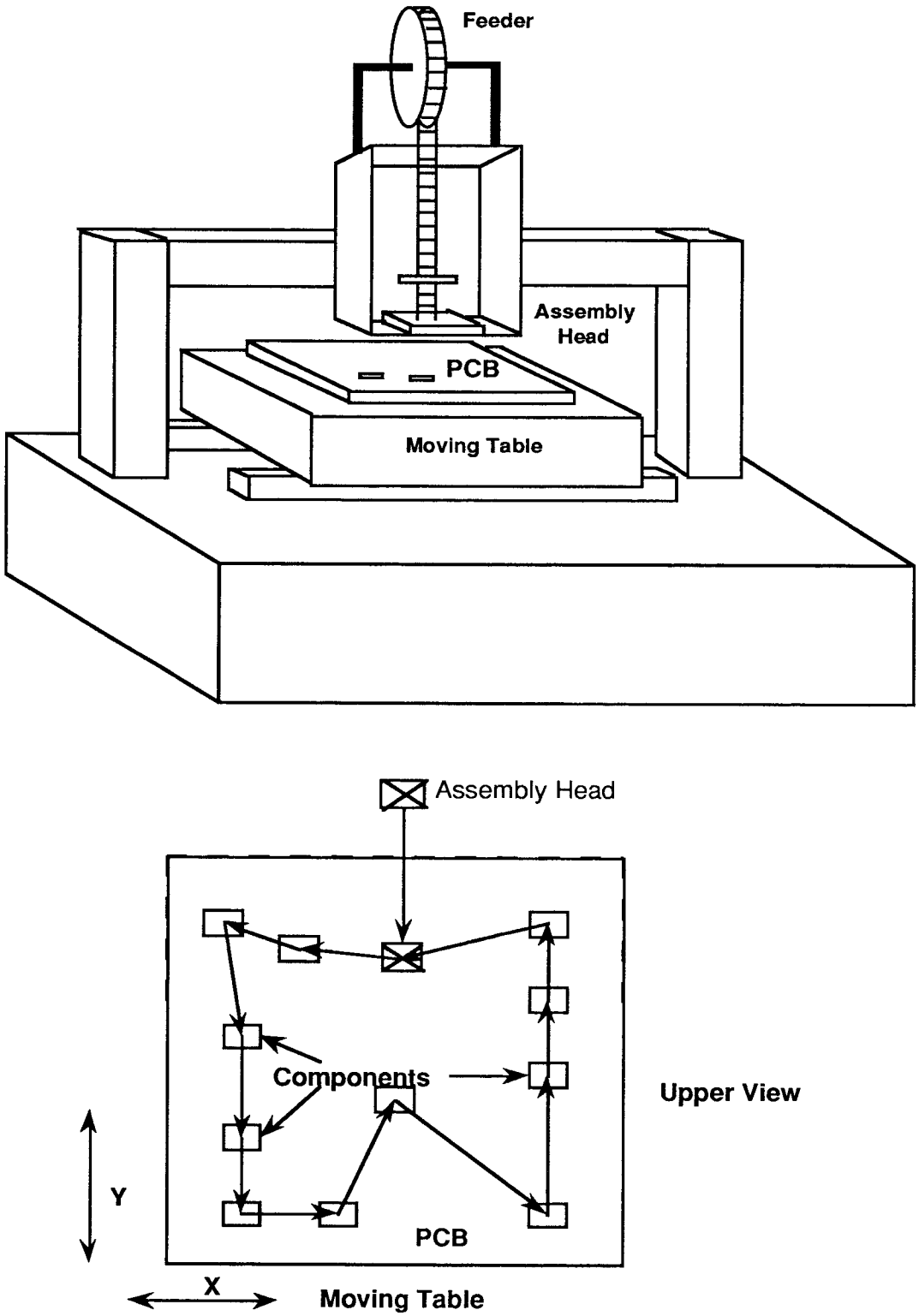


Figure 1. Assembly machine for type one problem. The assembly head is fixed in the (x, y) coordinates while the table moves when performing the assembly of components.

In the case the components are supplied by feeders at fixed locations, the assembly tool needs to pick up components from different feeders and place them on the board. Many of the assembly machines for surface mount components belong to this category, such as Amistar FA-2001, Automelec Autoplace, Fuji EP-20, and Panasonic Panasert MPA. Figure 2 is a schematic for this type of machines. In the bapartite graph shown in Figure 3, c_i (i=1,...,n) represents a component location on a PCB, and f_j (j=1,...,m) represents a feeder location from which a certain type of components is supplied. A line a_{ij} connecting c_i and f_j denotes that the component c_i matches with the feeder f_j. This problem can not be directly modeled as the traveling salesperson problem since the mounting head often needs to visit a feeder location more than once. It was previously modeled as a rural postman problem [1] and recently as an assignment problem [4]. In addition to planning the sequence of placing/inserting components on the board, the planning problem also includes assigning components to feeders. The problem will be called the pick-and-place problem (PPP) or the type two problem in the later chapters.

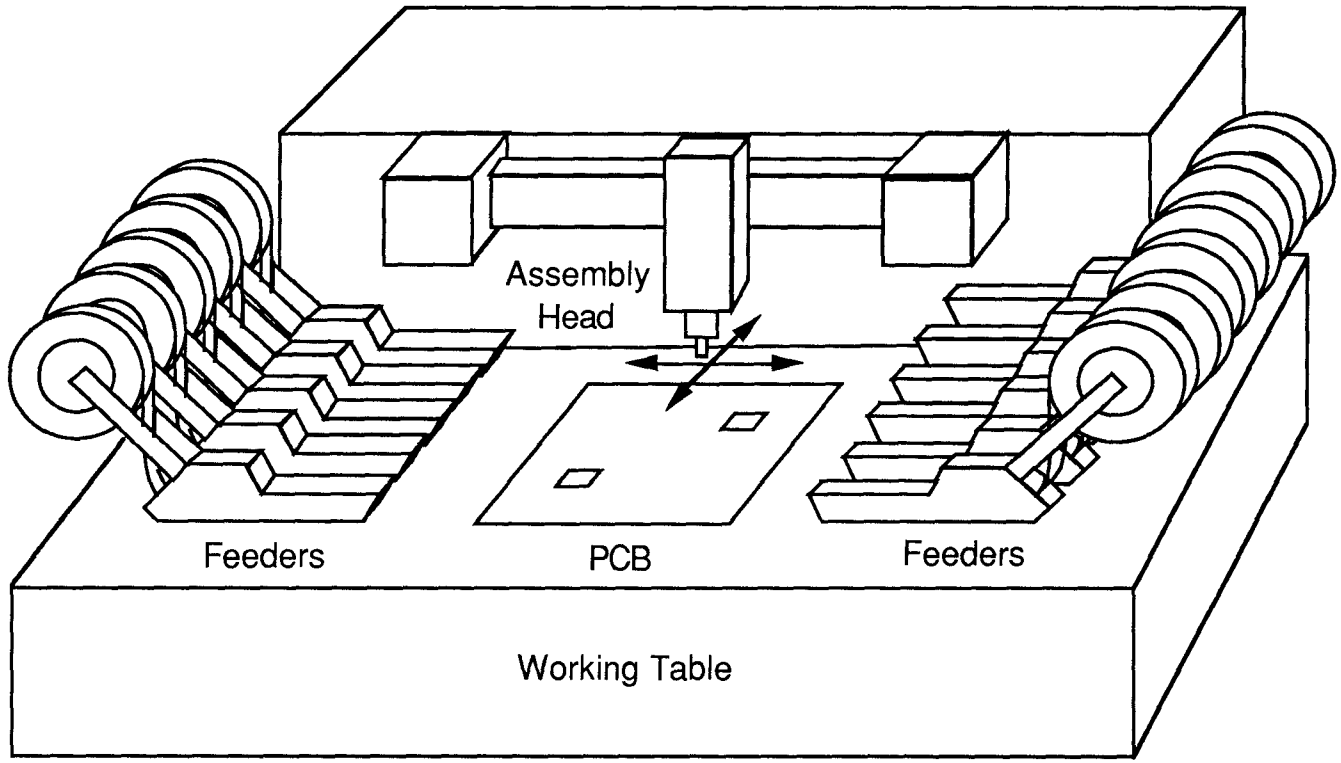


Figure 2. Assembly machine for type two problem. The board and feeders are fixed while the assembly head moves when performing the assembly of components.

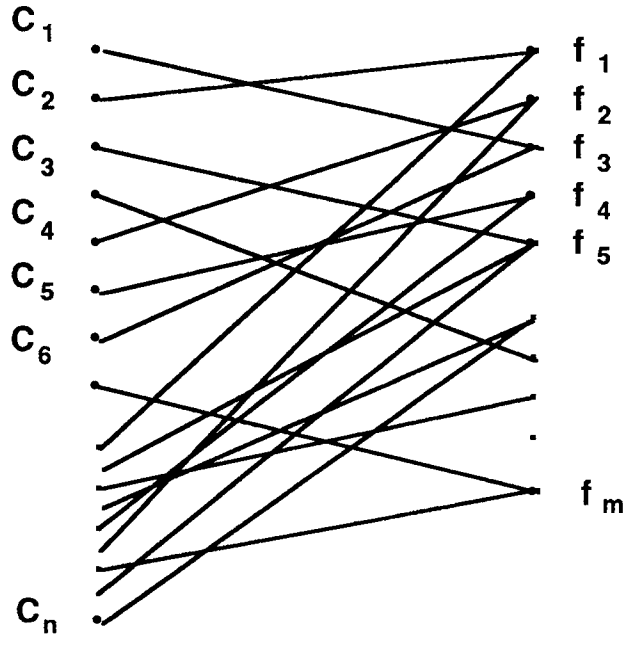


Figure 3. Bapartite graph

In the case the components are directly fed to the assembly head, but with possible time delay longer than the PCB traveling time between two successive components in the sequence, the problem can not be modeled as a traveling salesperson problem. The Panasert RH in [2] and Universal Onserter II 4712B are two commercial machines which belong to this category. Figure 4 shows a schematic for this type of problems. The traveling time of the PCB, or the shifting time of the tool head (the indexing time of the turret in Figure 4), or the traveling time of the moving feeder carrier will be the dominating time of the assembly, depending on which is longer. The summation of the dominating times in the assembly of all board components is the shortest total assembly time needed for a printed circuit board. The problem here is the combination of determining the sequence of the assembly head visiting all components and

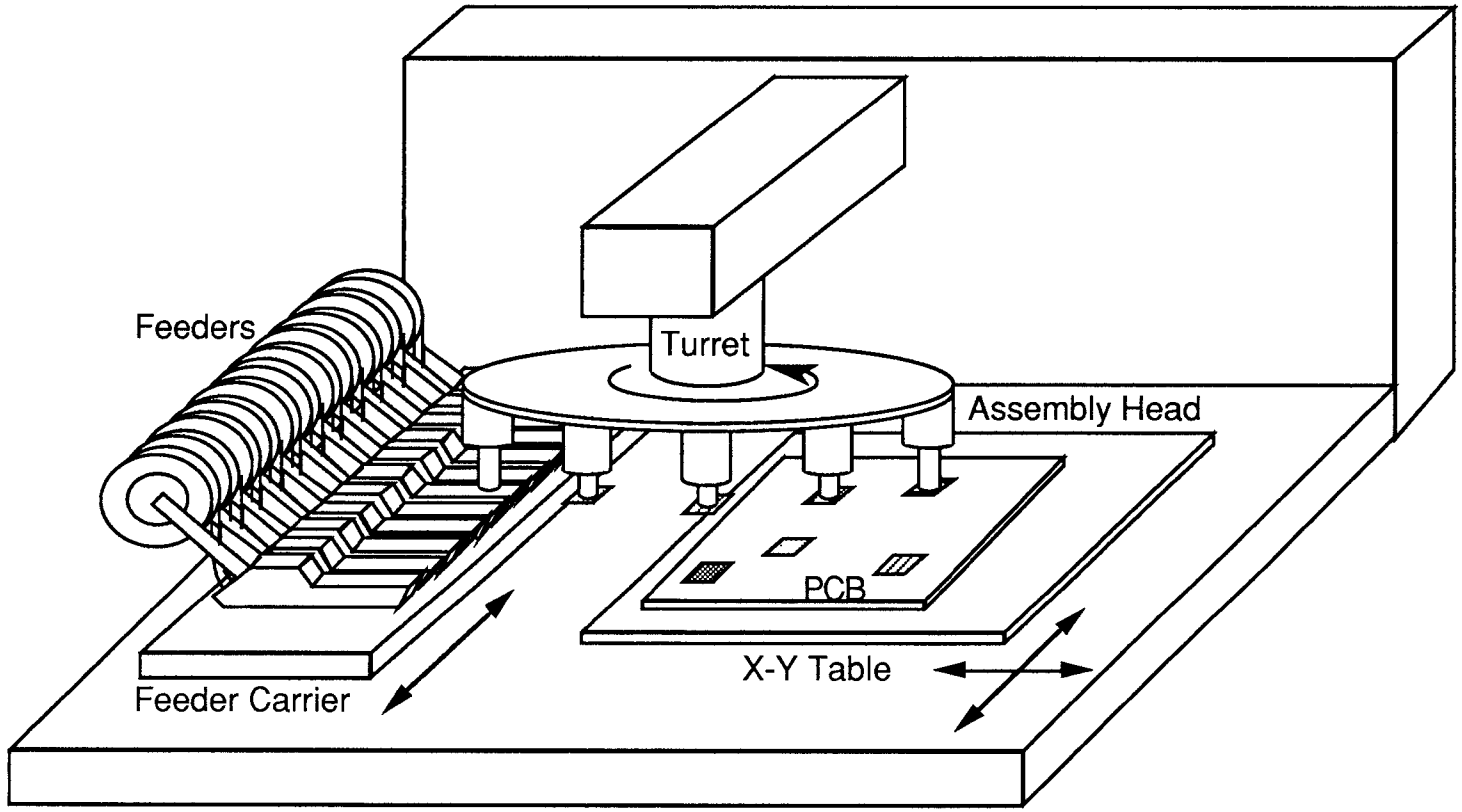


Figure 4. Assembly machine for type three problem

assigning the components to the feeders such that the total assembly time is minimized. This will be called the moving board with time delay problem (MBTDP) or the type three problem in the later chapters. Table 2 shows the relation between the characteristics of assembly machines and the corresponding planning problem types. Examples of commercial assembly machines are also given.

Table 2. Relation between assembly machine characteristics and planning problem types

Problem Type		Assembly Station Characteristics	Examples of Commercial Machines
1	Traveling Salesperson Problem	Stationary head, X-Y table, direct feeding of components to assembly head	Amistar AI-6448, Panasonic Panasert RT, Universal 6287A, Universal 6241B
2	Pick-and-Place Problem	moving head, stationary table, stationary feeders	Amistar FA-2001, Automelec Autoplace, Fuji EP-20, Panasonic Panasert MPA
3	Moving Board with Time Delay Problem	X-Y table, moving feeders, supply of components with a multi-head turret or a moving head between two fixed locations	Fuji CP-II, Panasonic Panasert MQ1, Universal Onserter II 4712B

Chapter 3

Genetic algorithm

Motivated by the success of the genetic algorithm approach in solving difficult optimization problems [8-12], we decided to explore the possibility of developing a genetic algorithm for determining the sequence of component placement/insertion and arrangement of feeders. A new operator called the rotation operator was created for use with this algorithm. This chapter will give an overview of genetic algorithm, including the various genetic operators.

3.1 Overview of Genetic Algorithm

First introduced by John Holland [5] at 1975, a genetic algorithm [5-7] is a general-purpose stochastic optimization algorithm which uses a process similar to biological evolution to improve an initial set of feasible solutions through an iteration process. The metaphor underlying the genetic algorithm is the natural evolution. The building block of a genetic algorithm is the *gene* which represents a certain permutation of the basic elements in solving a problem. A sequence of the genes that represents certain

physical meaning of the problem is one of the solutions and is called a *link*. The first step in developing a genetic algorithm is to identify genes and with which to establish links for a given the problem.

For the particular assembly planning problems of our study, for example, the genes are formed by components and feeders. A link is a sequence that indicates an order in which the components are placed or inserted or a permutation that refers to the order of assigning component types to feeders.

The second step in developing a genetic algorithm is to generate some initial links randomly. These links are used as the parents on which various *genetic operators* (to be discussed in chapter 3.2) are applied to generate the *offspring* (new links). The offspring, together with their parents, are evaluated by an objective function. The good ones are collected as new parents. This process is iterated until a certain criterion, such as a specified number of iterations, is reached. During the iterative process, the genes in the parent links are adaptively recombined by the genetic operators.

The number of offspring generated by each operator is measured by the operation rate, which is the ratio between the number of parents and their offspring to the number of parents. That is, if the number of parents is m and the number of offspring is n, then the operation rate is $\frac{m+n}{m}$. Due to the stochastic nature of the selection process, the fitter parents are likely to produce more offspring, and the less fit parents are likely to produce less offspring.

Thus, the fitness of parents is expected to improve from generation to generation. This is the basic mechanism of optimization in the genetic algorithm. It provides a near-optimal solution in a complex search space.

Generally speaking, a genetic algorithm has the following components:

1. a representation of problem solution in the form of links,
2. a way of creating initial solutions,
3. an evaluation function which rates solutions in terms of their "fitness",
4. a way of applying genetic operators for generating new solutions in the iterative process.

The main structure of the genetic algorithm used in this thesis is shown in Figure 5. It uses four operators: crossover operator, mutation operator, inversion operator, and rotation operator.

3.2 Genetic Operators

3.2.1 Crossover Operator

The most important genetic operator is the crossover operator. This operator is a key to genetic algorithm's power. It operates on

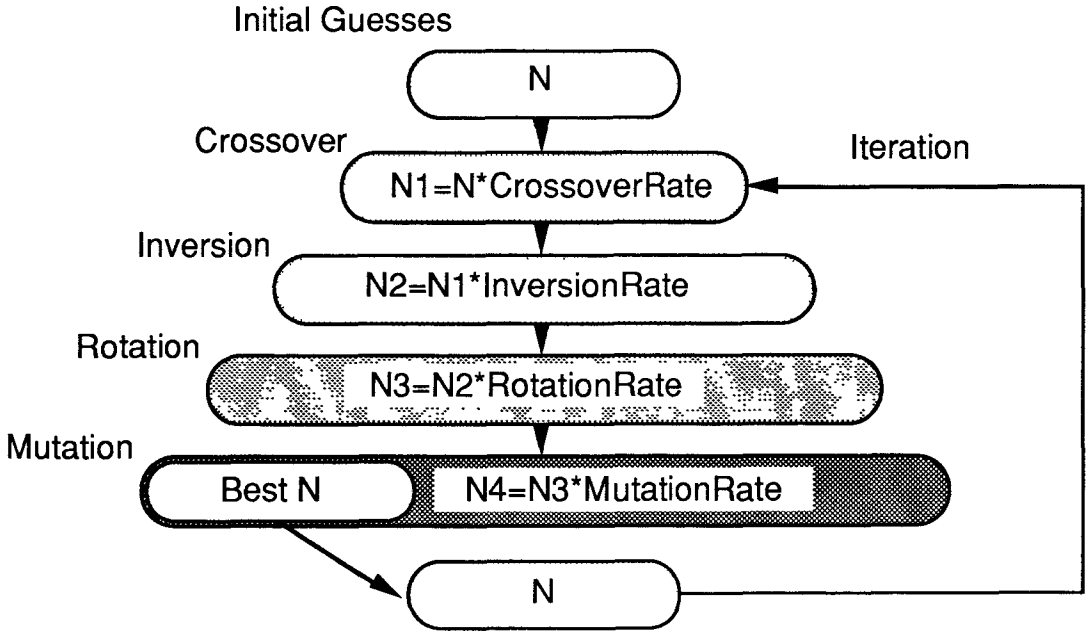


Figure 5. Main structure of a genetic algorithm

two parent links and generates an offspring link by combining partial links of the parents. A way to achieve a crossover is to choose a cut point randomly anywhere in the two parent links, then merge them using the first portion of one parent link and the second portion of another parent link to form the offspring. A simple merged link in this way, however, produces some repeated genes and some missing genes in the offspring for the assembly planning application. Therefore, the crossover rule is adjusted so that proper link structure is preserved. Depending on how the merge is performed, there can be different crossover operators. The crossover operator used in this thesis is the "order crossover" [14]. Figure 6 illustrates how this operator works. A cut point is chosen at random

location (e.g. the middle as shown in Figure 6). The left segment of the first parent link is then copied to the offspring. The remaining part of the offspring is filled with genes that are not already in the offspring from the second parent link, thus maintaining their relative order in the original link.

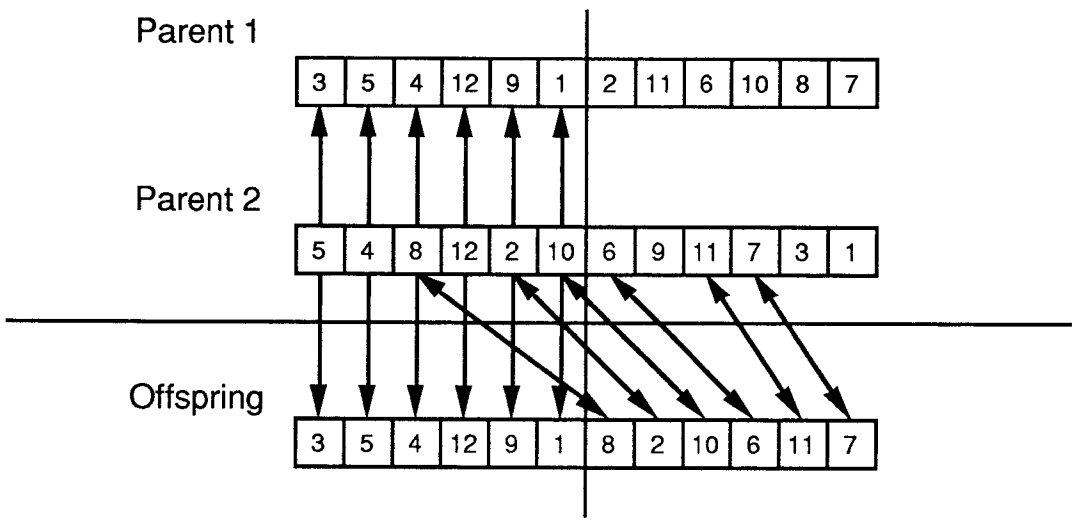


Figure 6. Crossover operator

3.2.2 Mutation Operator

The mutation operator provides the background variation and it occasionally introduces benefit to an offspring link. This can be done by exchanging two genes in a parent link. An example is illustrated in Figure 7.

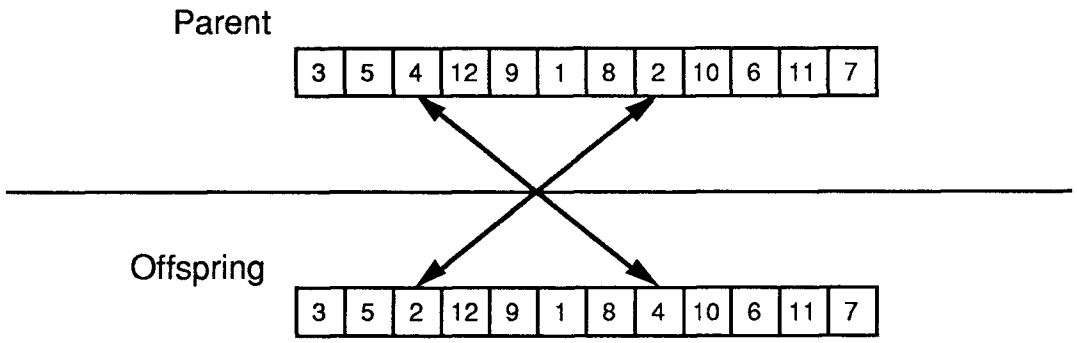


Figure 7. Mutation operator

3.2.3 Inversion Operator

The inversion operator takes a segment from a parent link and flips it to form a new link, as shown in Figure 8. This operation increases the probability of moving together certain genes that are co-adapted to cluster during the crossover operation. Therefore, it provides a better chance to have some advantageous properties inheriting to the offspring.

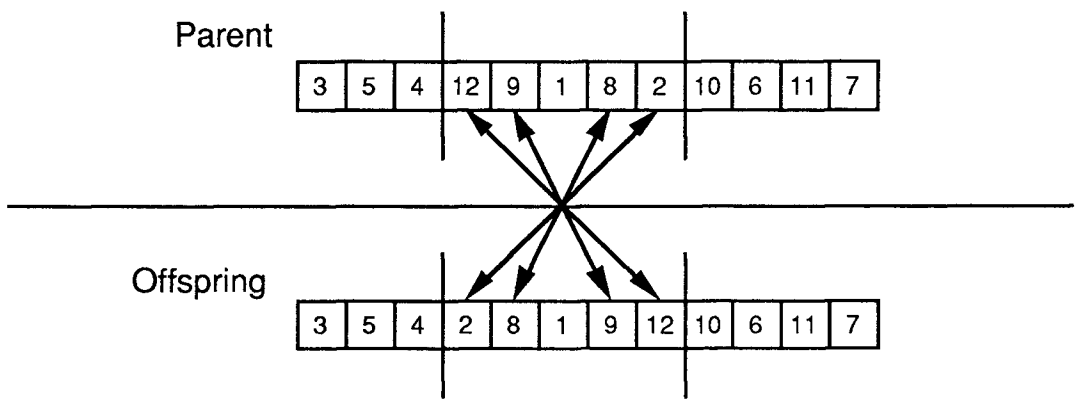


Figure 8. Inversion operator

3.2.4 Rotation Operator

The rotation operator rotates a segment of a parent link to the right or left to create an offspring. An example is shown in Figure 9, where offspring 1 is obtained through a right rotation of the middle segment of the parent, while offspring 2 is obtained through a left rotation of the same segment of the parent. This rotation operator is created from the observation that the rotation of a middle portion of an assembly sequence might generate a better one.

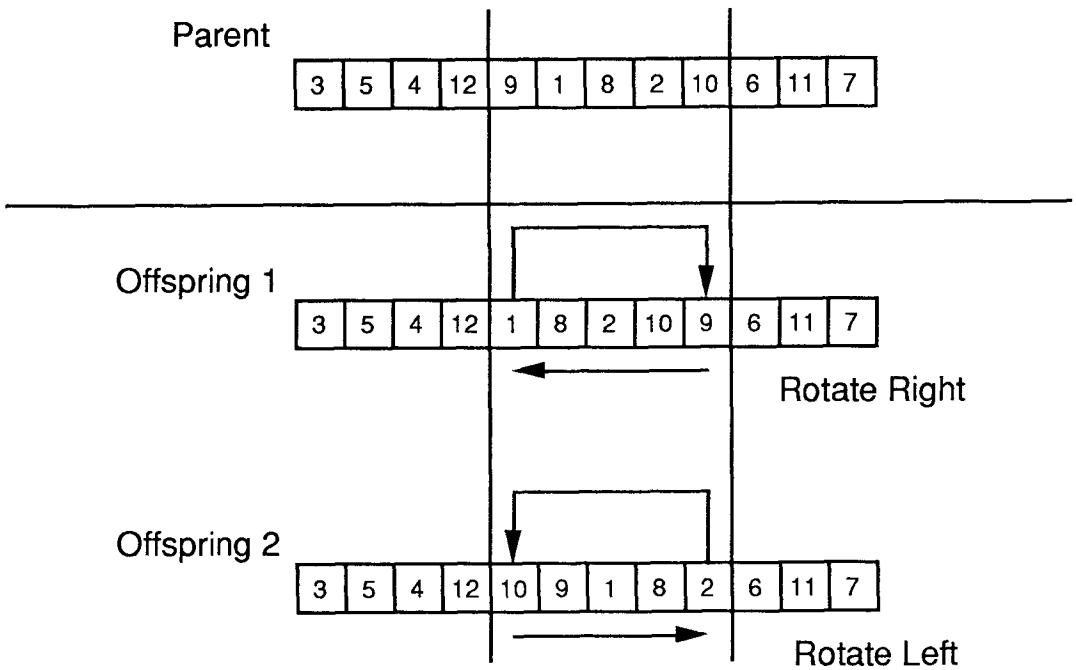


Figure 9. Rotation operator

3.3 Determining When to Stop Iterations

Although the genetic algorithm approach has been developed for more than ten years, there exists no good method for deciding when to stop iterations. Due to the random process in the algorithm, there is no guarantee for progress in each iteration. Sometimes there may be no progress for hundreds of iterations but then it starts to converge again. Terminating the iteration if the progress stands still for some small number of iterations is not a suitable criterion. One way to stop iteration is to keep monitoring the results, and if it seems to converge very slowly then terminate the iterations. Another way is to terminate the iterative process when "a number of iterations have been achieved." A better way might be to use a cost reduction associated with each iteration. If a genetic approach starts from a random generation of parent links, the former iterations should have better progress than the latter iterations. This means that the cost reduction for a former iteration should be better than that for a latter iteration. If we can find a way to evaluate the cost of each iteration, we can terminate the iteration if the cost is less than some value based on a certain criterion.

3.4 Operation Rates

Another important issue in using a genetic algorithm is about operation rates. What operation rates should be used so that the

genetic algorithm can perform well? It is a problem to consider when starting the genetic approach. Grefenstte [15] digitized all genetic parameters, including the operation rates, into some distinct values. He used a binary link to choose the parameter values from these values and used the genetic algorithm itself to optimize the binary link.

In the thesis, an easy way was devised to search better sets of operation rates beside using the fixed rates approach. We always fix the number of parent links and the number of offspring links in each iteration. With a given set of operation rates, the times of each operator's effect in some iterations are counted during the calculation. If the times of effect counted of an operator is greater than other operators at the end of these iterations, the operation rate of that operator is raised so that hopefully it will act better in the latter iterations. If some operation rates are raised, other operation rates will be reduced since the total number of offspring is fixed. Chapter 8 will discuss more in detail.

Chapter 4

Application of Genetic Algorithm to PCB Assembly Planning

The planning problems associated with most of PCB assembly machines fall into two categories. One is the sequence of components assembly, and the other is the assignment of components to feeders. If we consider several machines in an assembly line, we also need to consider which machine to assign to for each board component. The components assembly sequence can be represented as a link. The feeder assignment can also be represented as a link. It is obvious that the genetic algorithm is a good candidate for solving this kind of problems.

To implement a genetic algorithm for planning of PCB assembly, the link representation needs to be first established. All the three types of problems described in Chapter 2 involve the planning of components assembly sequence. For a PCB having n components, a link is a list of n component numbers, each between 1 and n, representing the placement/insertion sequence of components. In the example shown in Figure 10, the number "8" in

the third position of the link means that the component number 8 on the PCB is the third component in the assembly sequence, after component numbers 2 and 6.

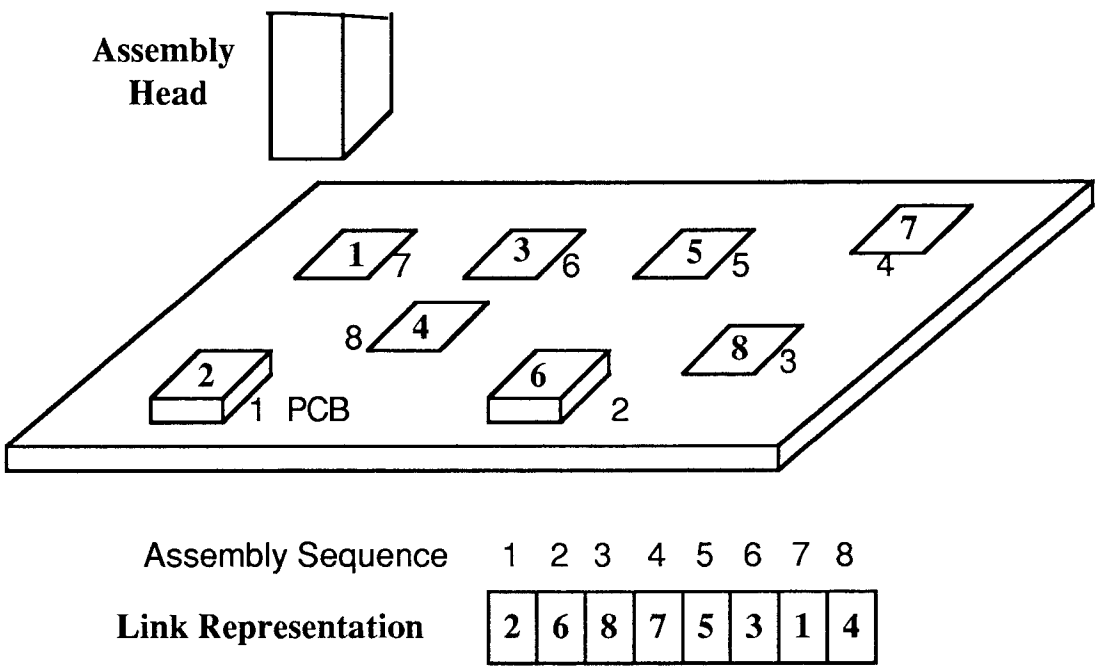


Figure 10. Link representation of assembly sequence

In both type two and type three problems, instead of using the usual single-link method, a multi-link method is developed for the combined component placement/insertion and feeder arrangement problem. Each feeder is assigned with either one type of components or empty. Another link is thus created for the feeder assignment. For the problem with m feeders and p types of components, where $m \geq p$, the second link is a list of feeders, each between 1 and m, representing the assignment of component types to feeders. The

feeders assigned with a number larger than p will not be used. The above two links can be combined together as a group and optimized simultaneously. In the optimization process the genetic operators are applied to each link separately, but the two links are evaluated together.

The initial estimates of solutions in the iterative process are randomly generated to provide a goodness test of the implementation. For actual application, however, it may be more expedient to initialize with more directed methods [2]. For example, we can find the initial estimate from the setup of a professional technician. Combined with some random initial estimates, we can improve the solution.

The evaluation functions for different types of problems may be different. The objective of PCB assembly planning is usually to minimize the total assembly time. The total assembly distance can be used as the objective instead of total assembly time, if the speed of assembly head is constant.

The movements of the assembly head or boards may have different patterns for different assembly machines. For the machines which control the X direction motion and Y direction motion separately without coordination (the Chebyshev metric), the evaluation function for traveling distance is $max(|x_1 - x_2|, |y_1 - y_2|)$ where $max(a, b)$ is the larger of a and b, and (x_1, y_1), (x_2, y_2) are the locations of these two components. For the machines which move

from one location to another in a straight line, the evaluation function for traveling distance between two component locations is $\sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$ (the Euclidean metric). If the assembly head is a robot with revolute joints, instead of using assembly distance as the evaluation function, we should consider the assembly time whose evaluation needs to involve the robot's kinematics.

Chapter 5

Genetic Algorithm for Traveling Salesperson Problem

5.1 Representation of PCB Assembly Planning problem

As described in Chapter 2, in this problem the assembly head visits each component location once. This problem is to determine the assembly sequence so as to minimize the total traveling distance. For a PCB having n components, the assembly sequence is represented by a link with n genes. Figure 11 is an illustration of this problem. The link designated by 5-4-2-1-3-6 represents an assembly sequence in which the assembly head goes through the sequence of component 5 $\Rightarrow$ component 4 $\Rightarrow$ component 2 $\Rightarrow$ component 1 $\Rightarrow$ component 3 $\Rightarrow$ component 6 $\Rightarrow$ component 5. After the sequence ends, it repeats again and again.

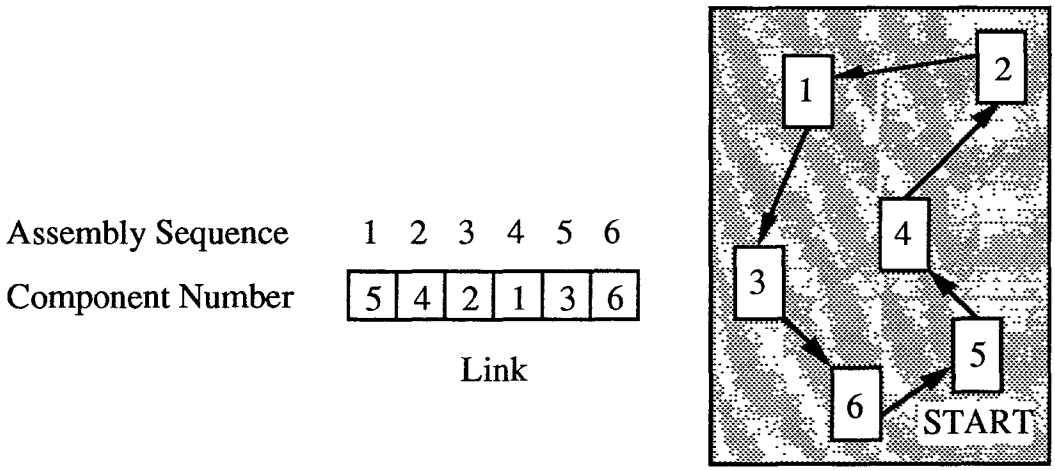


Figure 11. Link representation for type one problem

The assembly distance between component i and component j is

$$d_{i,j} = \begin{cases} max(|x_j-x_i|, |y_j-y_i|) & \text{for Chebyshev metric} \\ \sqrt{|x_j-x_i|^2+|y_j-y_i|^2} & \text{for Euclidean metric} \end{cases}$$

where $1 \le i \le n$, $1 \le j \le n$.

The evaluation function is:

$$\left[\sum_{i=1}^{i=n-1} d_{i,i+1} \right] + d_{n,1}$$

which is the total traveling distance, i.e. the total distance of visiting each component once.

The above travel distance formulation can be extended straightforwardly to solve minimal travel time problems.

5.2 Results and Discussion

A PCB of thirty components is used as an example in solving the traveling salesperson problem. The locations of the components are listed in Table 3. Thirty links are generated randomly as the initial estimates. Only Euclidean metric is assumed in the assembly process of this PCB. Fixed operation rates are used and they are listed in Table 4. The assembly process produced after solving the problem is illustrated in Figure 12. The total traveling distance is 256.8 mm. The computation time is 75 seconds for 200 iterations on the Compaq 386/20e PC.

Table 3. Locations of components (mm)

	X	Y		X	Y		X	Y
1	16.0	18.1	11	78.3	41.2	21	60.7	18.1
2	8.3	41.2	12	77.5	35.6	22	9.7	26.8
3	4.4	16.0	13	13.5	40.8	23	28.7	24.0
4	81.0	41.2	14	86.1	28.9	24	3.5	35.8
5	1.8	23.8	15	87.1	41.2	25	7.8	35.6
6	5.7	41.2	16	17.6	29.8	26	36.3	35.0
7	11.2	35.6	17	40.1	35.5	27	20.8	29.4
8	55.2	41.2	18	32.3	41.2	28	74.8	41.8
9	80.7	22.7	19	84.2	41.2	29	86.4	22.3
10	63.9	27.3	20	34.3	18.1	30	17.7	24.0

Table 4. Operation rates

Operator	Operation Rate
Crossover Operator	1.50
Inversion Operator	1.22
Rotation Operator	1.22
Mutation Operator	1.54

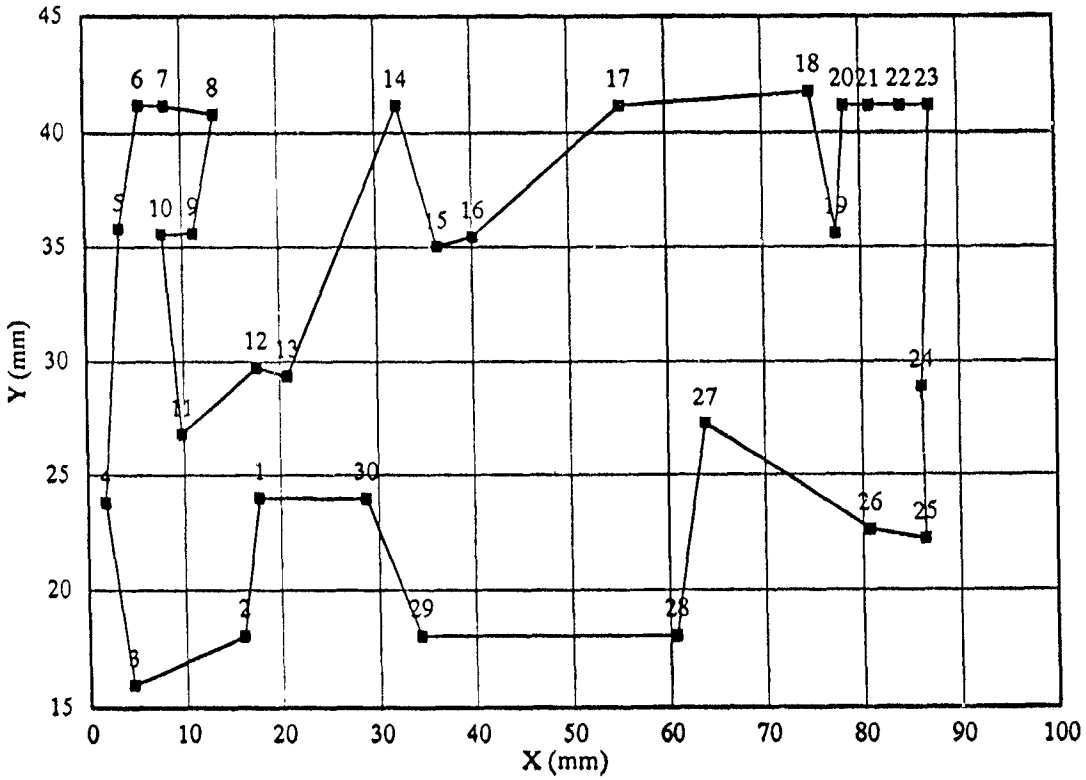


Figure 12. Optimal assembly sequence based on genetic algorithm

Several trials had been made to start each iteration from a different random initial guess. Figure 13 shows the results of total travel distance reduction for five different randomly generated initial estimates, each having 200 iterations.

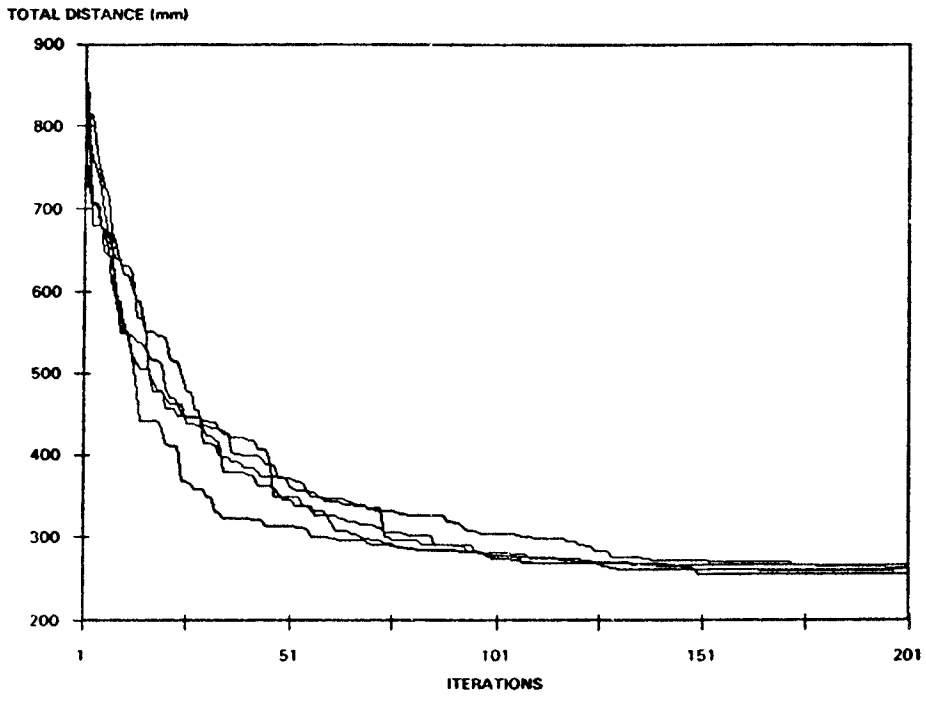


Figure 13. Solutions from genetic algorithm with different initial guesses

There is an heuristic based solution algorithm from STORM [16], which is an optimal scheduling software package run on the IBM PC/XT/AT. The algorithm used by STORM to solve the traveling salesperson problem is based on least distance insertion or maximum angle insertion to determine the travel sequence. The total assembly distance obtained by STORM for this problem is 268.2 mm, which is 4.4% worse than the result of the genetic algorithm. The computer time used is 285 seconds which is almost three times larger than the result from the genetic algorithm. The solution obtained from STORM is shown in Figure 14.

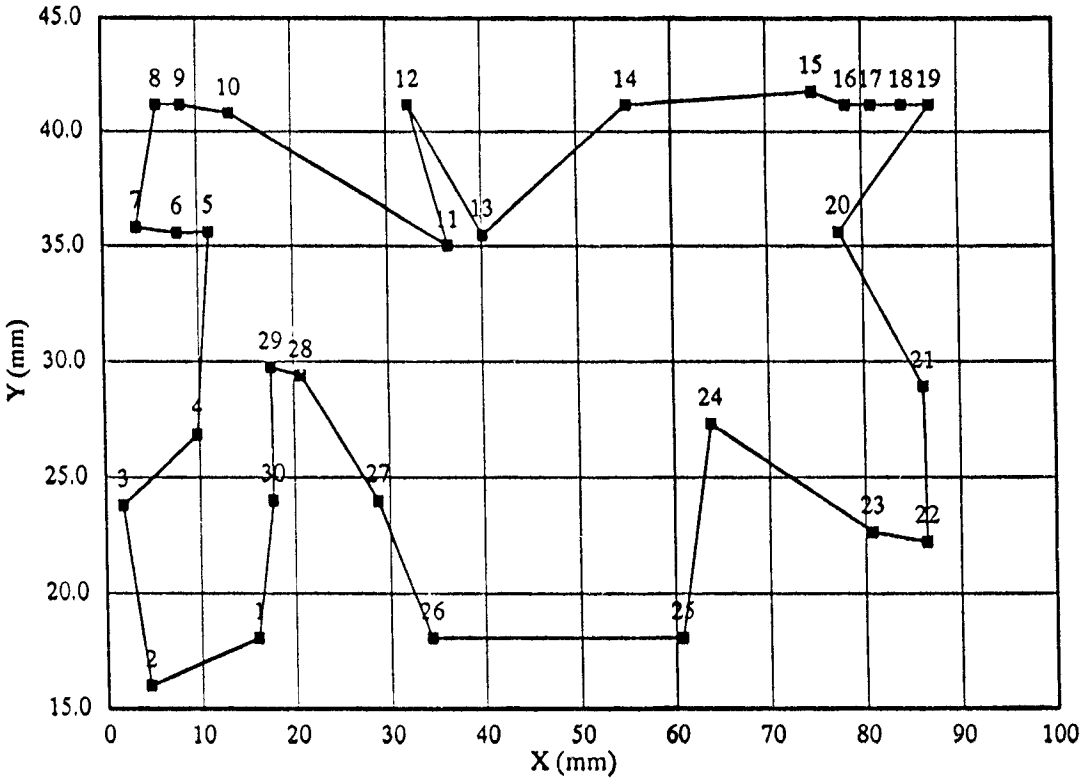


Figure 14. Optimal assembly sequence from STORM

Chapter 6

Genetic Algorithm for Pick and Place Problem

6.1 Representation of PCB Assembly Planning Problems

For a pick-and-place machine, the assembly tool picks up components from feeders and place them on the board. The optimization problem needs to determine the minimum traveling distance; however, in addition to the assembly sequence, it also needs to determine the assignment of components to feeders. We assume that the PCB has n components in p types, and the components are provided by m feeders, where $m \geq p$. In applying the genetic algorithm approach to solve this problem we used two links: one is the link of assembly sequence, with n genes, and the other is the link of feeder assignment, with m genes.

We assume that the assembly head will rest at a particular location in the transition between two printed circuit boards. The rest location is the "starting point" and also the "end point" in assembling each board since the assembly process will start at this point and the assembly head will go back to this point after a PCB

has been assembled. In the following the rest location will be called the "starting point."

Figure 15 is an illustration of this problem. The component sequence and feeder arrangement are designated by link 1 and link 2, respectively. In this illustration, the assembly head goes through the following locations: starting point ⇒ feeder 2 ⇒ component 3 ⇒ feeder 3 ⇒ component 5 ⇒ feeder 3 ⇒ component 6 feeder 2 ⇒ component 2 ⇒ feeder 1 ⇒ component 1 ⇒ feeder 1 ⇒ component 4 ⇒ starting point.

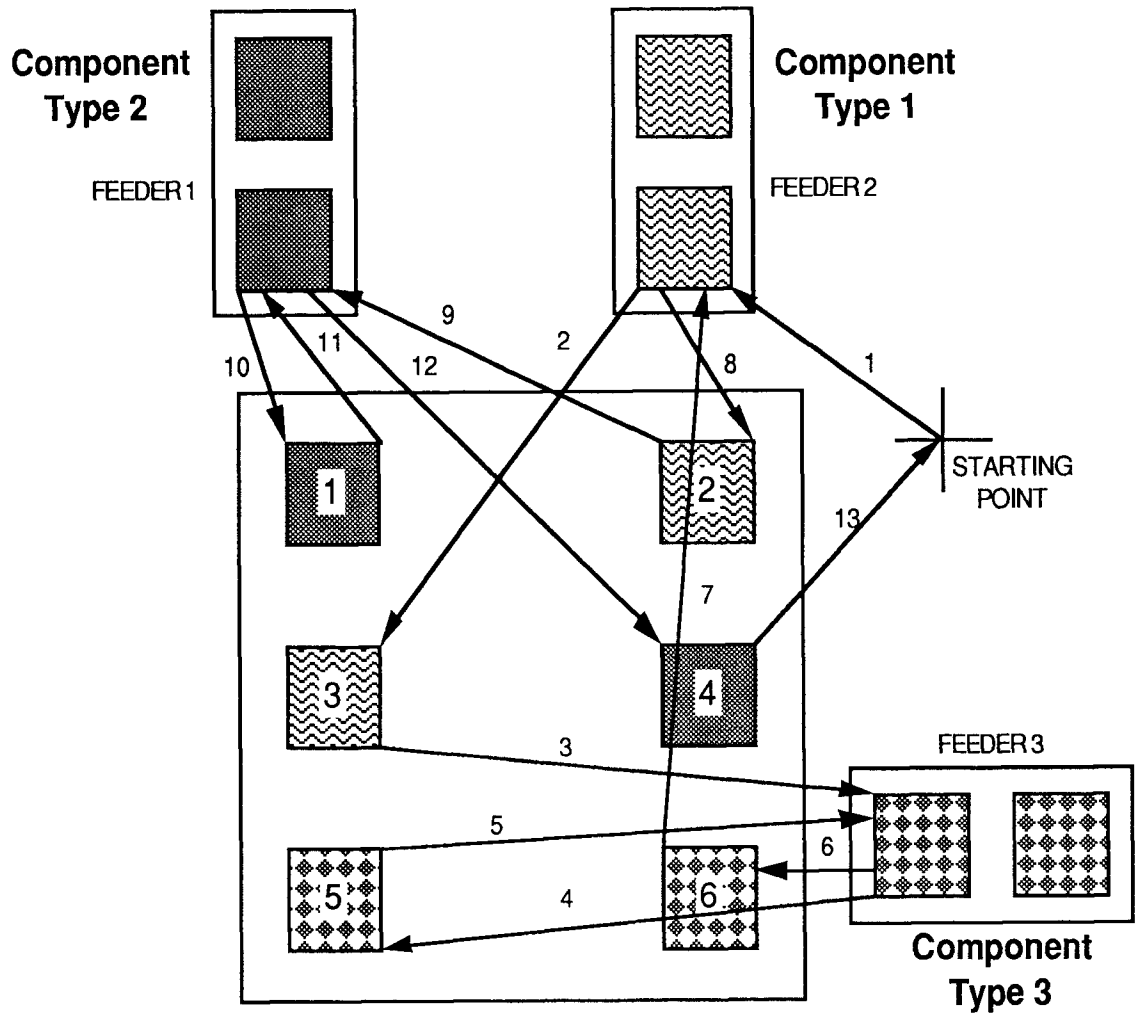
The assembly distance between feeder i (x_i, y_i) and component j (x_j, y_j) is

$$d_{i,j} = \begin{cases} max(|x_j-x_i|, |y_j-y_i|) & \text{for Chebyshev metric} \\ \sqrt{|x_j-x_i|^2+|y_j-y_i|^2} & \text{for Euclidean metric} \end{cases}$$

where $1 \le i \le m$, $1 \le j \le n$.

Let (x_s, y_s) denote the starting point, $d_{s,i}$ (or $d_{s,j}$) be the distance between the starting point and feeder i (or component j). Let $a[k]$ denote the feeder to which type k components are assigned and $b[j]$ denote the component type for component j, then $a[b[j]]$ represents the feeder for component j, where $1 \le k \le p$ and $1 \le b[j] \le p$ and $1 \le a[k] \le m$. The total assembly distance is:

$$d_{s,\,a[b[1]]} + \left[\sum_{j=1}^{j=n-1} (d_{a[b[j]],j} + d_{a[b[j+1]],j}) \right] + d_{a[b[n]],n} + d_{s,n}$$



Assembly Sequence 1 2 3 4 5 6

Component Number | 3 | 5 | 6 | 2 | 1 | 4 |

LINK 1

Component Type 1 2 3

Feeder Number | 2 | 1 | 3 |

LINK 2

Figure 15. Link representation for type two problem

6.2 Results and Discussion

A sample PCB with 200 components in 10 different types, as in Figure 16, is used as an example in solving the pick-and-place problem. Figure 16 (a) shows the locations of the components and feeders, and Figure 16 (b) gives the types of the components.

The starting point for this process is (0, 0). One hundred links are generated randomly as the initial estimates. Fixed operation rates are used and they are listed in Table 5. Figure 17 shows the assembly distance versus the number of iterations. The total assembly distance is reduced by 11.84% in 6,150 iterations and is still decreasing.

Table 5. Operation rates

Operator	Operation Rate
Crossover Operator	1.5
Inversion Operator	1.3
Rotation Operator	1.3
Mutation Operator	1.3

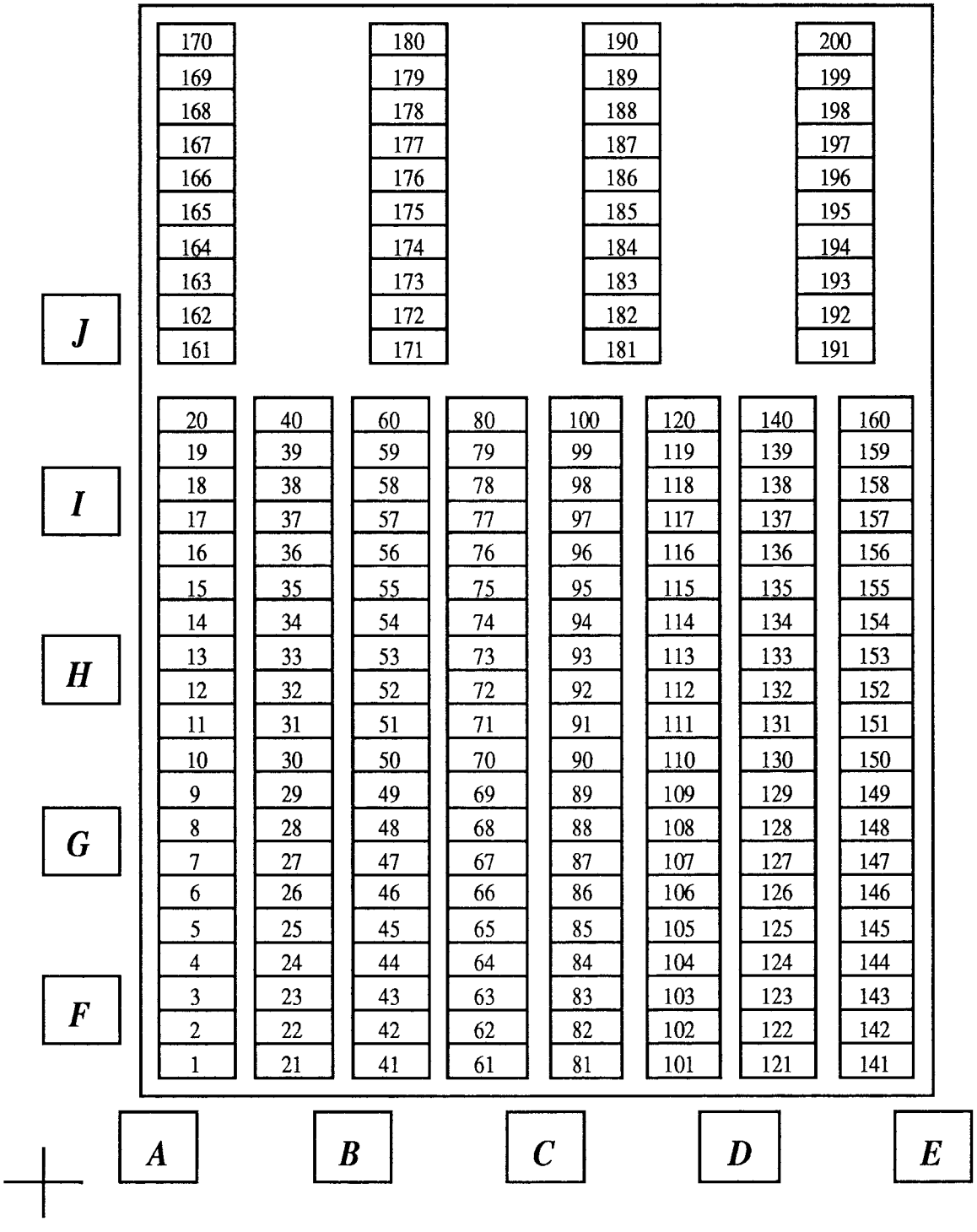


Figure 16 (a). Components and feeder locations

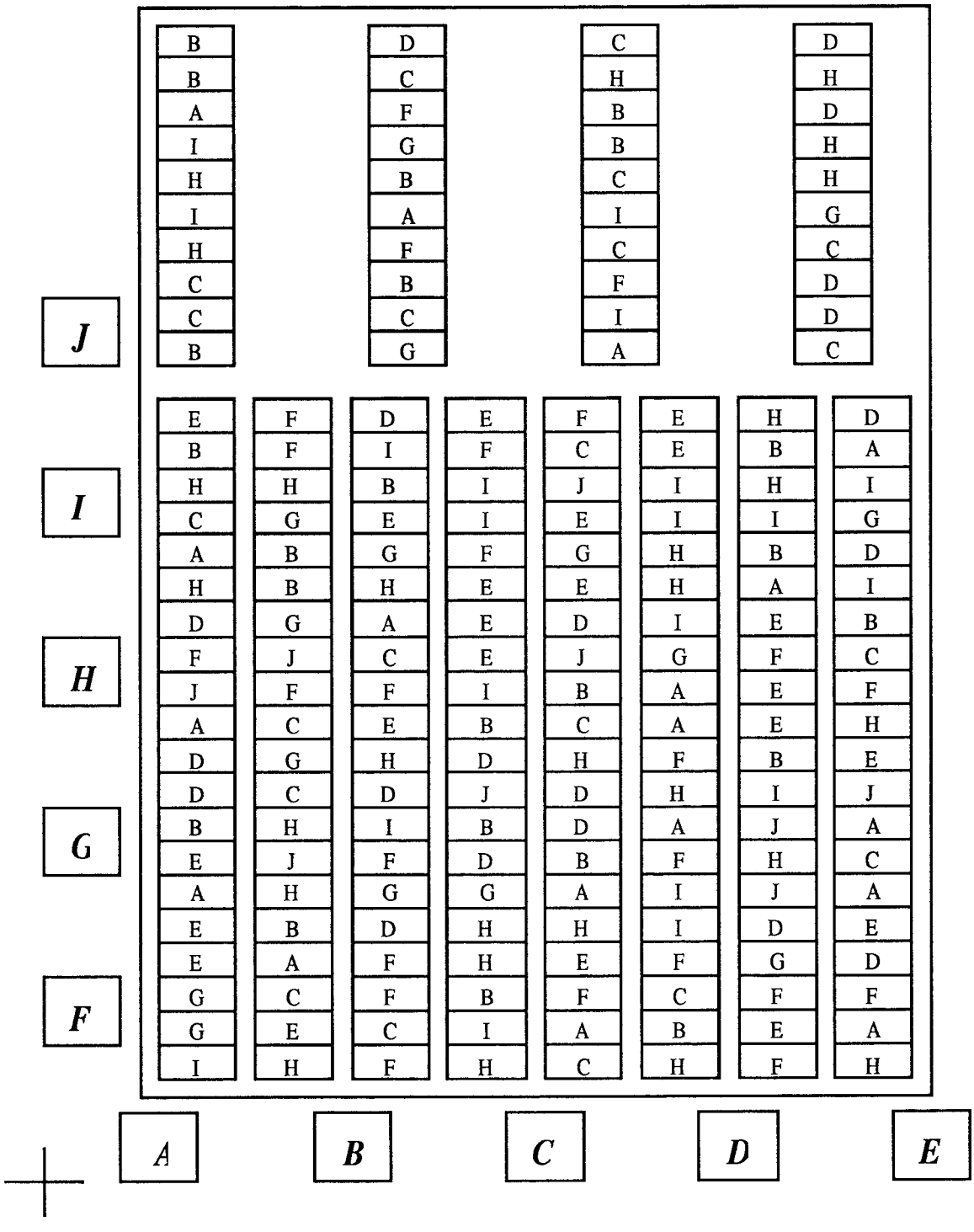


Figure 16 (b). Types of the components

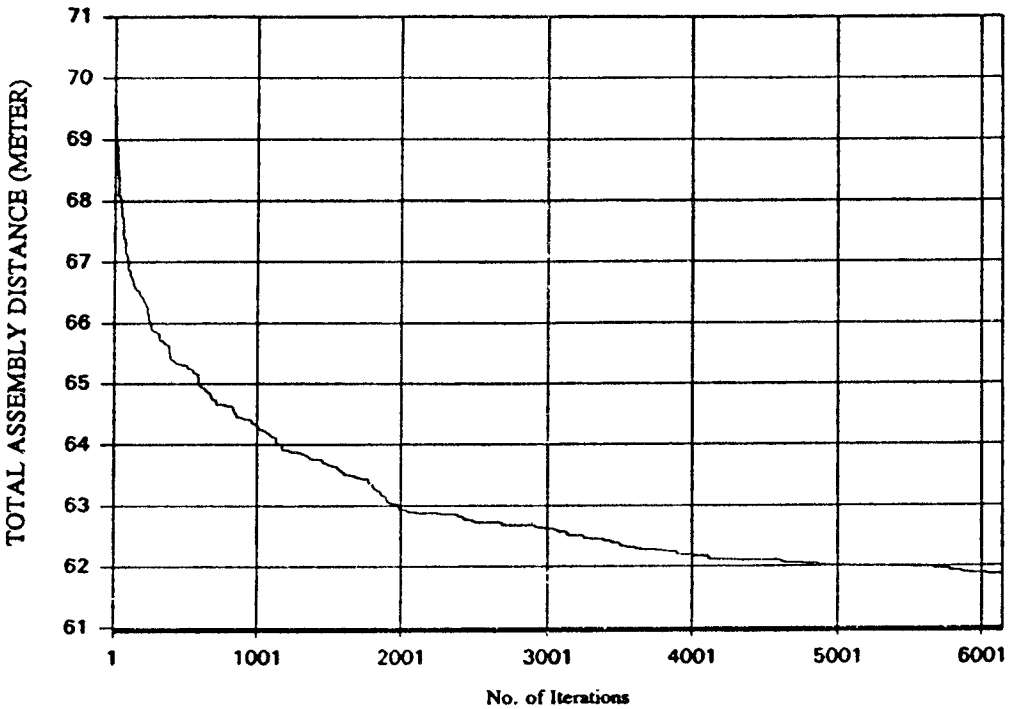


Figure 17. Solution from genetic algorithm

The feeder locations and assembly sequence obtained after the 6,150 iterations are illustrated in Figure 18. The total assembly distance is 61,861 mm. The squares on the bottom row and the left column represent the locations of the feeders. All the others represent the locations of the components. The number to the right of each feeder designates the type of components assigned to the feeder. The number to the right of each component designates the order of the assembly sequence for the component. Figure 19 shows the trajectory for assembling the components for the whole board, and Figure 20 shows the trajectory for assembling the first 20 components.

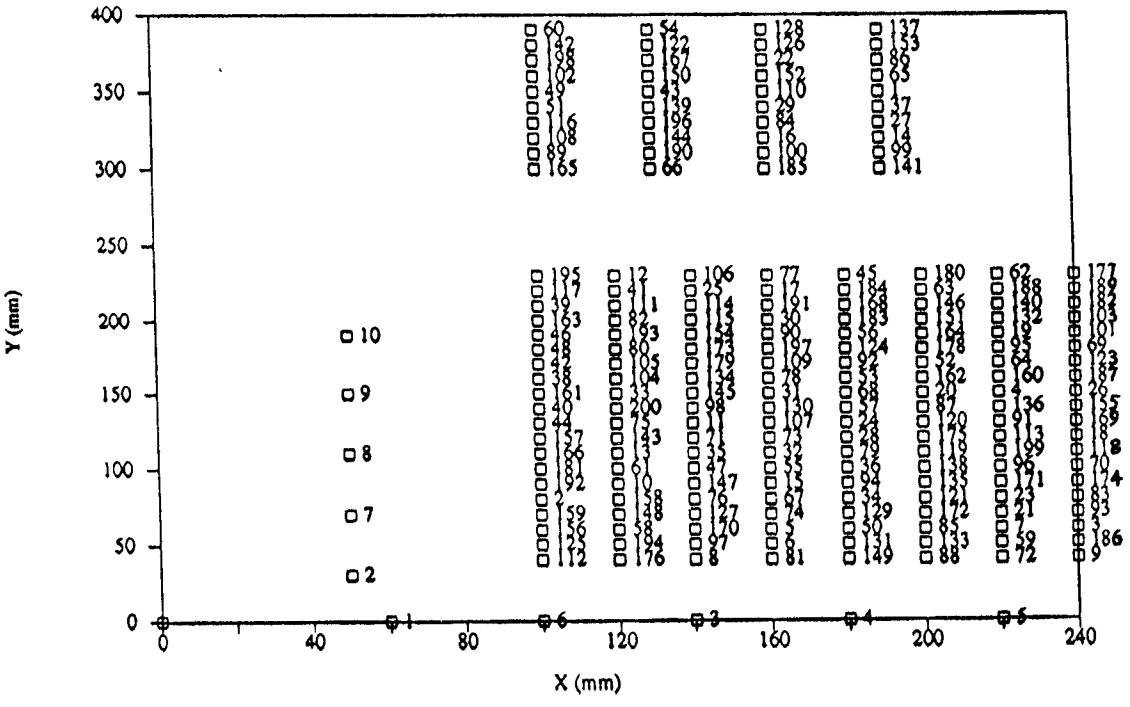


Figure 18. Optimal assembly sequence and feeder locations based on genetic algorithm

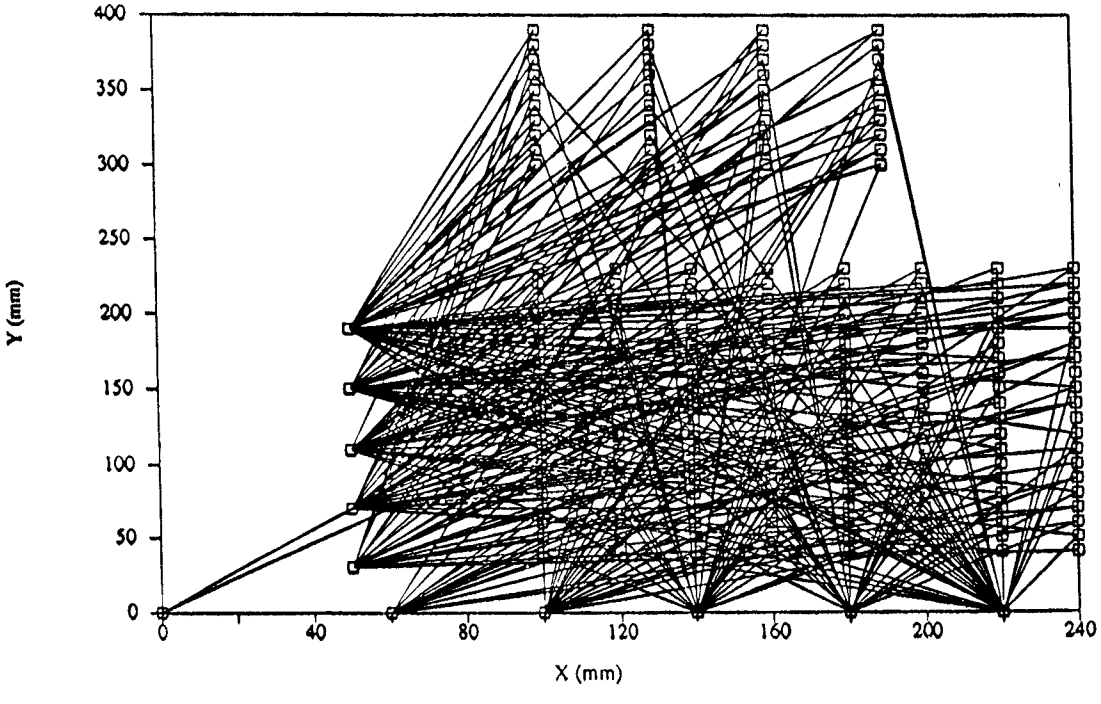


Figure 19. Optimal assembly sequence based on genetic algorithm

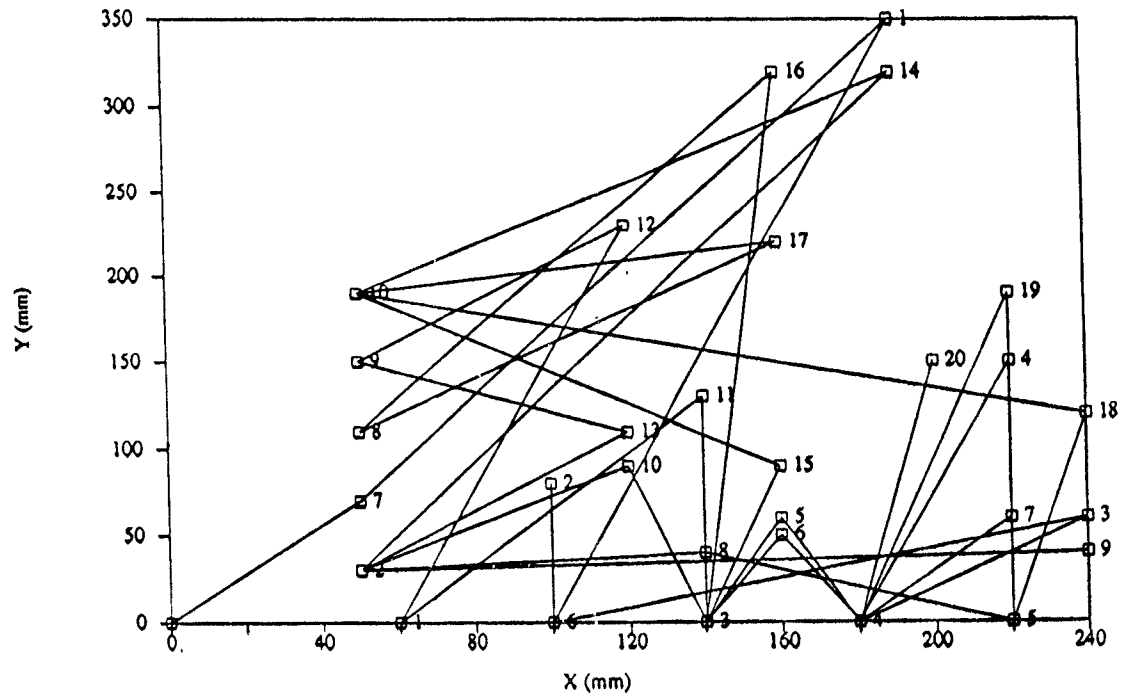


Figure 20. First 20 steps of assembly sequence

Another PCB which has fifty components was tested to compare the result with that from the dual reduced matrix method in [4]. The location and type of each component are listed in Table 6. Also listed in Table 6 are the locations of the feeders. The starting point of this assembly process is (0,0). Fifty groups, each having two links, are generated randomly as the initial estimates. The first link has fifty genes representing the assembly sequence. The second link has ten genes representing the feeder assignment.

Table 6. Locations of components and feeders and type of components (mm). The feeders are designated by F1-F10.

	X	Y	Type		X	Y	Type		X	Y	Type
1	100	60	6	21	160	180	4	41	240	40	10
2	100	90	3	22	160	220	9	42	240	60	9
3	100	130	2	23	180	60	9	43	240	80	4
4	100	180	10	24	180	100	5	44	240	100	8
5	100	230	4	25	180	140	8	45	240	120	1
6	120	50	4	26	180	180	4	46	240	140	10
7	120	90	10	27	180	220	8	47	240	180	7
8	120	130	6	28	200	60	8	48	240	200	6
9	120	150	9	29	200	100	9	49	240	210	7
10	120	190	5	30	200	130	9	50	240	220	2
11	120	230	9	31	200	140	3	F1	60	10	
12	140	40	9	32	200	170	7	F2	100	10	
13	140	80	2	33	200	180	10	F3	140	10	
14	140	100	9	34	200	220	4	F4	180	10	
15	140	140	4	35	220	40	9	F5	220	10	
16	140	180	10	36	220	60	9	F6	70	30	
17	140	220	7	37	220	100	10	F7	70	70	
18	160	60	5	38	220	160	9	F8	70	110	
18	160	100	2	39	220	200	7	F9	70	190	
20	160	140	5	40	220	220	5	F10	70	150	

The operation rates used are listed in Table 7. Fifty links are generated as the initial estimates. The result produced is illustrated in Figure 21. The total distance resulted from the genetic algorithm is 11,324.6 mm. The computation time is 9 hours 10 minutes for 10,000 iterations on the Compaq 386/20e PC. The optimal value was achieved at iteration 7,841. The total assembly distance at iteration 1,000 is 11,390 mm, which took about 55 minutes of computer time and the total assembly distance of iteration 100 is 11,549.6 mm, which took only 6 minutes of computer time. Figure 22 shows the travel distance as a function of iteration numbers.

Table 7. Operation rates

Operator	Operation Rate
Crossover Operator	1.72
Inversion Operator	1.52
Rotation Operator	1.52
Mutation Operator	1.55

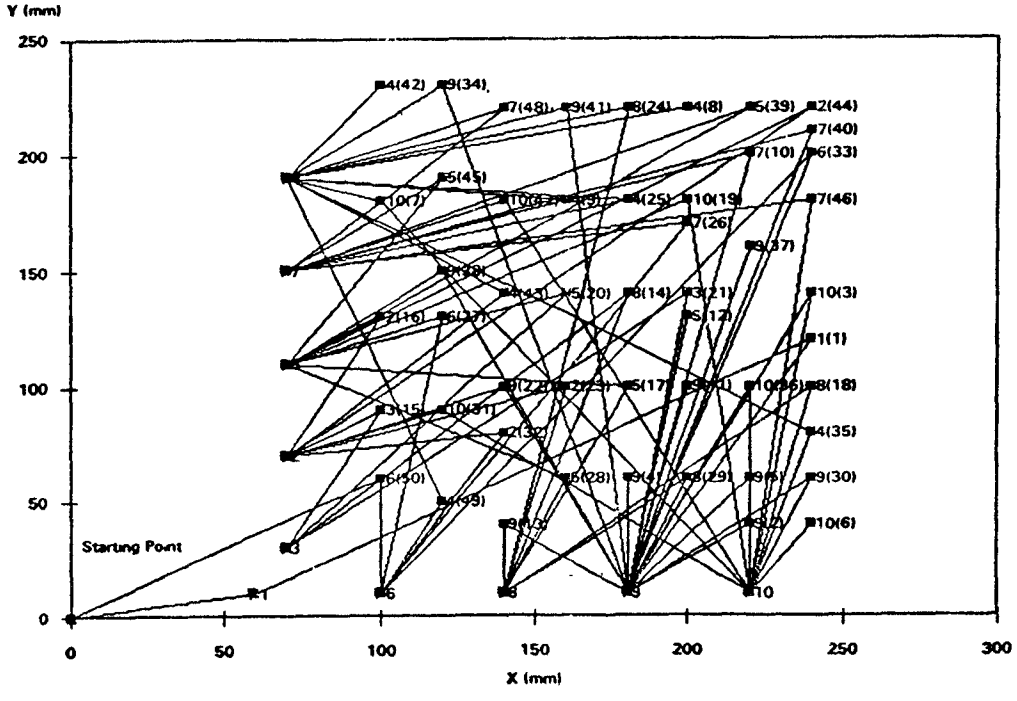


Figure 21. Optimal assembly sequence based on genetic algorithm

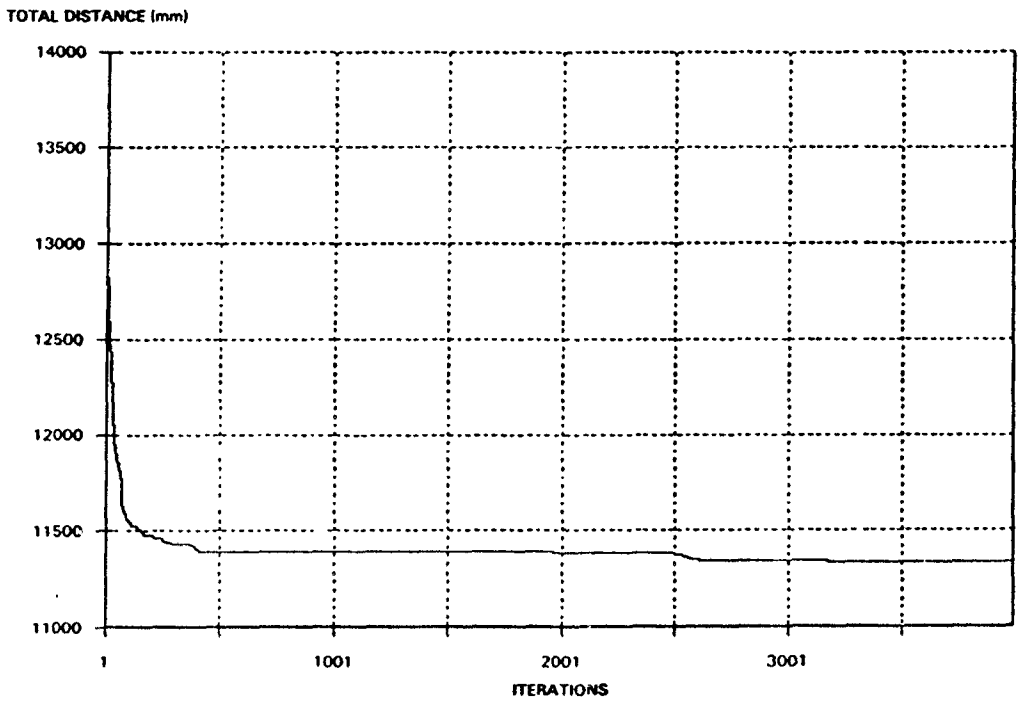


Figure 22. Solution from genetic algorithm

The optimal solution from [4] for the first board tested above and the same feeder locations has a total distance as 61,847 mm. The optimal solution from [4] of the second PCB and the same feeder locations is 13,010.7 mm, see Figure 23. The computation time was less than one second on the Compaq 386/20e PC for the second board. The method of [4] solved the assignment problem concerning the place movements only and neglecting the pick up movements. It solved only three fourths of the problems. The genetic approach solves both the sequencing problem and the assignment problem simultaneously but take a lot of random trials. The solutions of the two algorithms are both sub-optimal solutions. This explains why the

genetic approach took more calculation time but had a worse solution for the first board and a better solution for the second board.

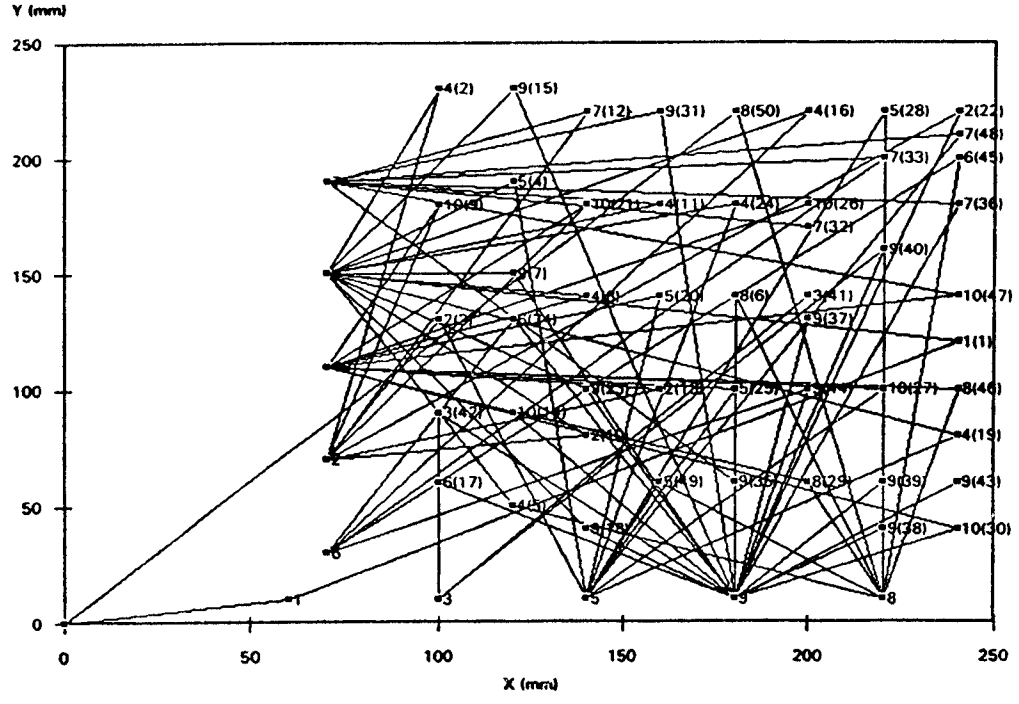


Figure 23. Assembly sequence from dual assignment method

Chapter 7

Genetic Algorithm for Moving Board with Time Delay Problem

7.1 Representation of PCB Assembly Planning Problems

The assembly machine of this type usually has a moving feeder carrier, a X-Y table for carrying the PCB, and a turret having multiple assembly heads. Unlike the above two types of problems, we must consider assembly time directly instead of travel distance in this problem. Considered in this type of problems are three different times: one is the traveling time of the PCB, another is the shifting time of the pick-up-head (i.e. the indexing time of the turret in multi-head assembly machines), and the other is the moving time of the feeder carrier. The longest time of the three is the time neck in the assembly of one component. The next movements of the other two will keep waiting until the neck movement has been completed. All the movements are considered as constant speed movements in order to simplify the problem.

We assume an n-component PCB having p types of components supplied by m feeders, where $m \geq p$. As in the pick-and-place

problem, we represent the problem by two links: one is the link for the assembly sequence, with n genes, and the other is the link for the feeder assignment, with m genes.

Another consideration is needed when the machine uses a turret carrying multi-assembly heads (for example, Panasert MQ1 NM-8257). The traveling time between two components is compared with the moving time of the feeder carrier for picking up a different component. For the first few components assembled in a batch of PCB's, there are only pick-up movements without any placement movement. For the last few components of the same batch, there are only placement movements without pick-up movements. If the quantity of PCB's is very large in a batch we can neglect these boundary effects and consider all components of this batch as a string of nearly infinite components.

Figure 24 shows an example of this problem. There is a "gap" of four components between the component being picked-up and the component being placed. For the first PCB of a batch, the machine will start at a rest location and pick up a component from a feeder having the type of component. Then the turret indexes to the next head, picks up the next component, and then the turret indexes again to pick up the third component. In the meantime, the PCB moves to the location and the first component is placed on the PCB. After the third component is picked up and the first component is placed, the turret indexes again to the next location, the feeder carrier moves to the next component, and the PCB moves to the next location.

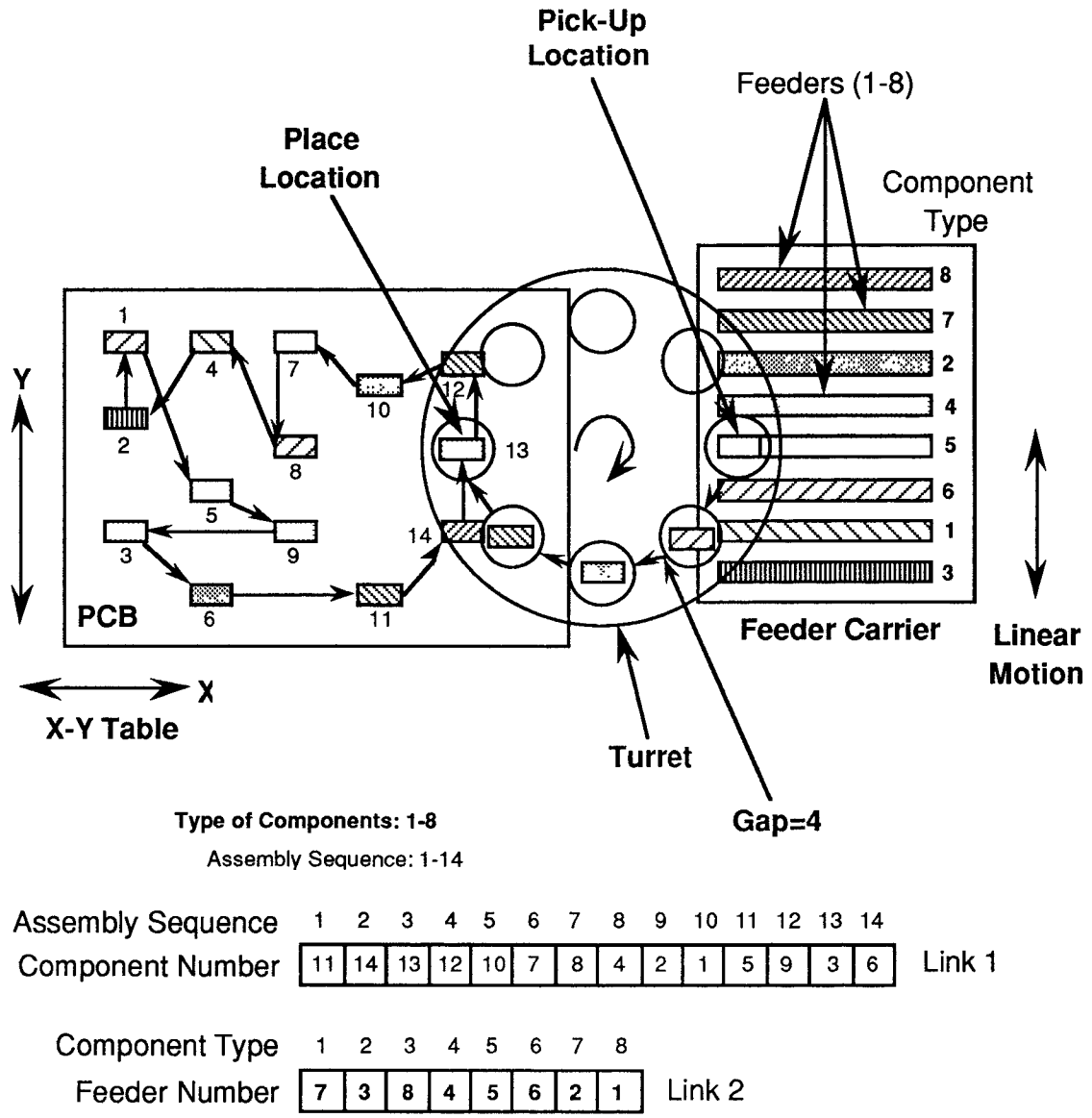


Figure 24. Link representation of type three problem

At the assembly of the last four components of a board, there is no need for picking up components for this board. Instead of wasting time to wait for the next board, the next movements can pick up the first four components for the next PCB.

Let g be the "gap" between the pick-up component and the place component, v_l denote the speed of the X-Y table linear motion, v_x denote the x-direction speed of X-Y the table motion, v_y denote the y-direction speed of the X-Y table motion, v_f denote the moving speed of the feeder carrier, $a[k]$ denote the feeder to which type k components are assigned and $b[j]$ denote the component type number for component j, where $1 \le k \le p$ and $1 \le b[j] \le p$ and $1 \le a[k] \le m$.

The assembly time from component i (x_i, y_i) to component $i+1$ (x_{i+1}, y_{i+1}) is

$$t_1 = \begin{cases} \max\left(\dfrac{|x_{i+1}-x_i|}{v_x}, \dfrac{|y_{i+1}-y_i|}{v_y}\right) & \text{for Chebyshev metric.} \\ \dfrac{\sqrt{|x_{i+1}-x_i|^2+|y_{i+1}-y_i|^2}}{v_l} & \text{for Euclidean metric.} \end{cases}$$

where $1 \le i \le n$.

The indexing time of the pick-up head is denoted as t_2.

The traveling time of the feeder carrier from the feeder of component $i+g$ $(x_{a[b[i+g]]}, y_{a[b[i+g]]})$ to feeder $i+g+1$ $(x_{a[b[i+g+1]]}, y_{a[b[i+g+1]]})$ is

$$t_3 = \frac{\sqrt{|x_{a[b[i+g+1]]}-x_{a[b[i+g]]}|^2+|y_{a[b[i+g+1]]}-y_{a[b[i+g]]}|^2}}{v_f},$$

where $1 \le i \le n$. If $i+g > n$, then component $i+g$ actually represents component $i+g-n$ of the next PCB in the batch.

The time neck for the assembly of component i is $t_i = max(t_1, t_2, t_3)$ and the evaluation function is $\sum_{j=1}^{j=n} t_i$ which is the total assembly time.

7.2 Results and Discussion

The same PCB of fifty components as that used in the type 2 problem study was tested for the type three problem. The feeder locations are in Table 8. The assembly time versus the iterations generated by the genetic algorithm is shown in Figure 25. The turret carries only one assembly head, so the "gap" here is 0. The X direction speed is 60 mm/second. The Y direction speed is 60 mm/second. The feeder speed is 60 mm/second. The turret rotates at one second/revolution. The optimal solution is shown in Figure 26. The solution is reached in iteration 7,204, and it has 51 seconds of total assembly time. Another solution obtained with a different set of initial estimates, shown in Figure 27, has an optimal total assembly time as 54 seconds reached in 204 iterations. The improvement during these 204 iterations is 26%. In comparison, the same improvement takes 490 iterations with the set of initial estimates used to obtain Figure 25. It is easy to see that if both the PCB and the feeder carrier move fast enough, the least assembly time will be 50 seconds, since there are 50 components and the turret indexing time is 1 second.

Table 8. Component locations and types and feeder locations. The feeders are designated by F1-F10.

	X	Y	Type		X	Y	Type		X	Y	Type
1	100	60	6	21	160	180	4	41	240	40	10
2	100	90	3	22	160	220	9	42	240	60	9
3	100	130	2	23	180	60	9	43	240	80	4
4	100	180	10	24	180	100	5	44	240	100	8
5	100	230	4	25	180	140	8	45	240	120	1
6	120	50	4	26	180	180	4	46	240	140	10
7	120	90	10	27	180	220	8	47	240	180	7
8	120	130	6	28	200	60	8	48	240	200	6
9	120	150	9	29	200	100	9	49	240	210	7
10	120	190	5	30	200	130	9	50	240	220	2
11	120	230	9	31	200	140	3	F1	100	10	
12	140	40	9	32	200	170	7	F2	115	10	
13	140	80	2	33	200	180	10	F3	130	10	
14	140	100	9	34	200	220	4	F4	145	10	
15	140	140	4	35	220	40	9	F5	160	10	
16	140	180	10	36	220	60	9	F6	175	10	
17	140	220	7	37	220	100	10	F7	190	10	
18	160	60	5	38	220	160	9	F8	205	10	
18	160	100	2	39	220	200	7	F9	220	10	
20	160	140	5	40	220	220	5	F10	235	10	

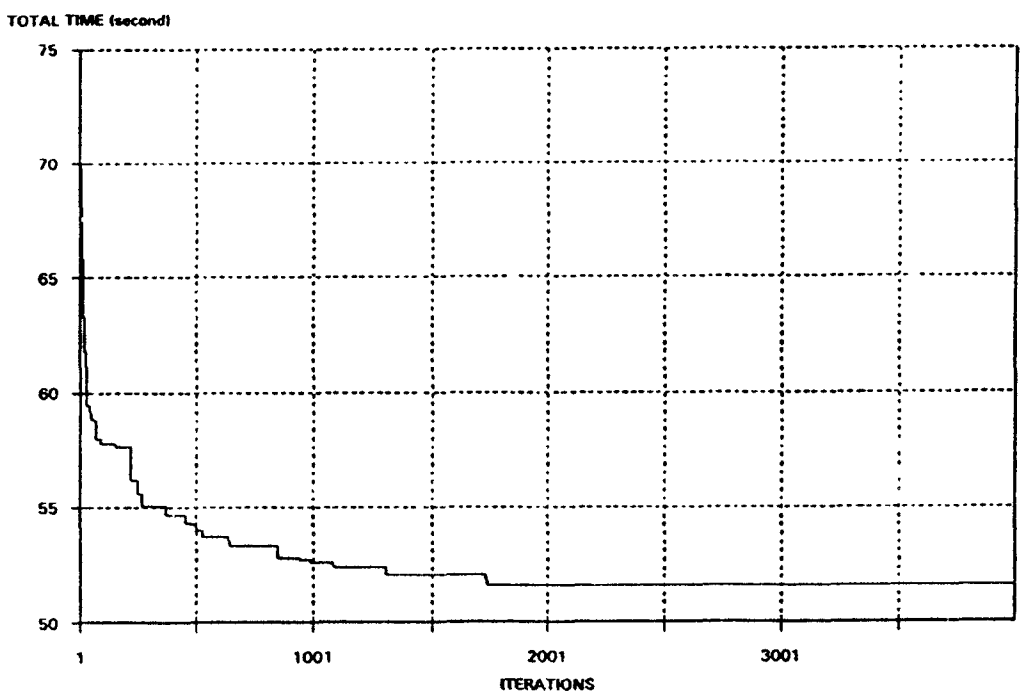


Figure 25. Solution from genetic algorithm

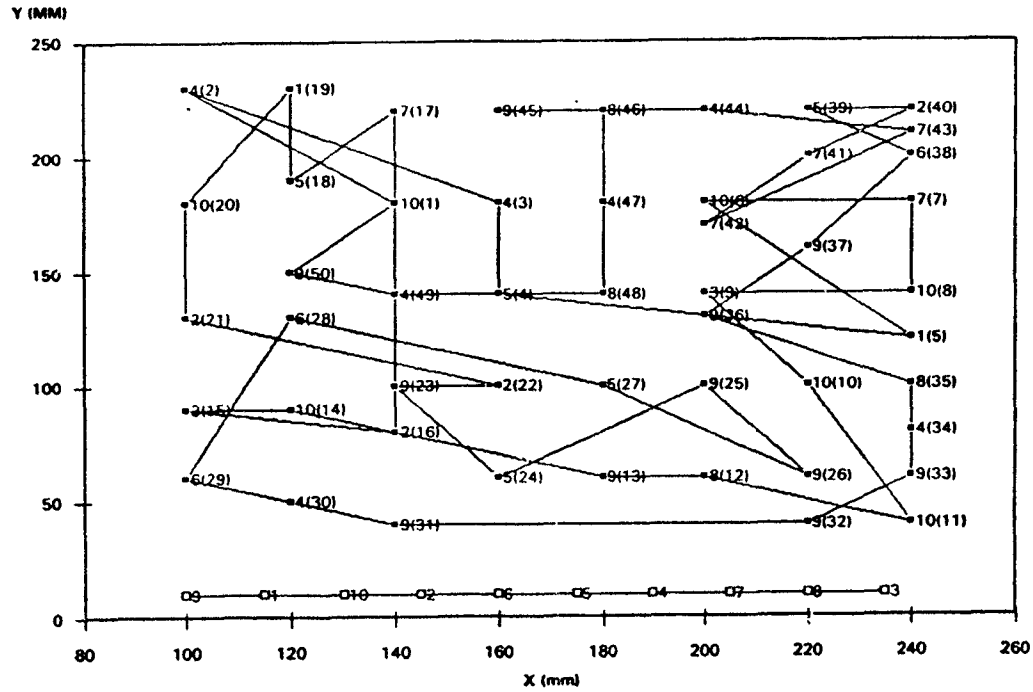


Figure 26. Optimal assembly sequence based on genetic algorithm

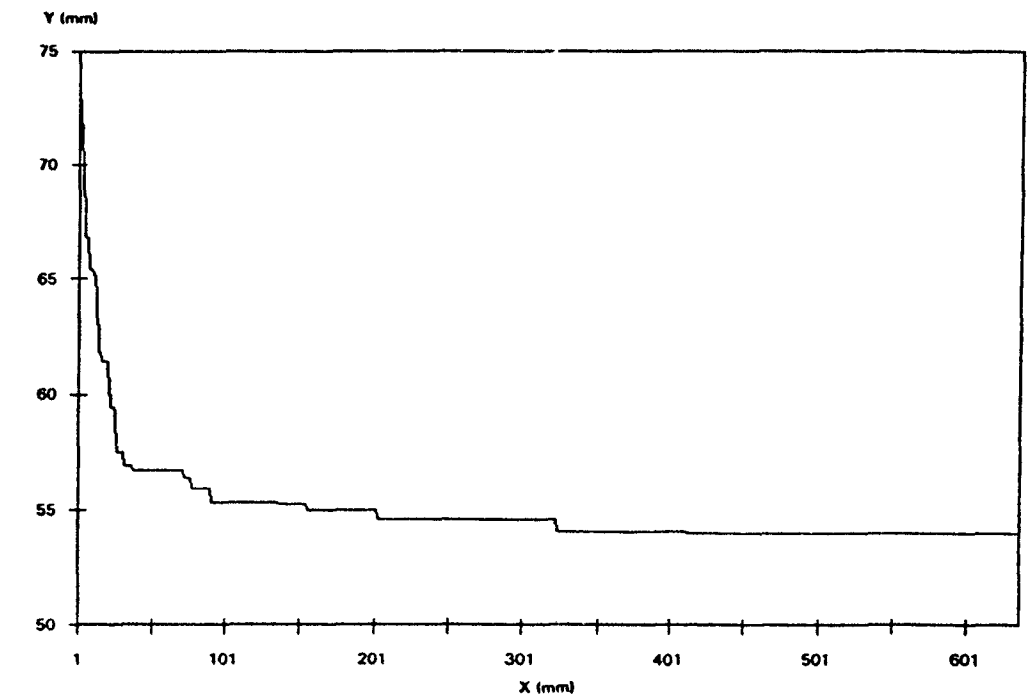


Figure 27. Solution from genetic algorithm

Chapter 8

Effect of Changing Operation Rates

8.1 Method

The meta-genetic method was introduced by Grefenstte, et al, to find optimal operation rates for a genetic algorithm [15]. In the meta-genetic method, a binary link is used to represent the rate of each operator; for example, a two-genes link is used to represent the rate of each operator; for example, a two-gene link for an operator can be setup as : "0-0" for 1.1, "0-1" for 1.2, "1-0" for 1.3, and "1-1" for 1.4. By combining the binary links of all operators as one link, the genetic algorithm method itself can be used to determine optimal operation rates together with other parameters. This may be computational intensive, however.

We device an easier way to find the operation rates for our genetic algorithm application. As discussed in Section 3.4, the changing operation rate method quantifies the effect of each operator in a certain number of iterations and adjusts its rate accordingly. The main idea of the changing operation rate approach is to increase or

decrease the rate of an operator depending on whether this operator has more or less effect than the other operators at some time during the process of iteration.

We define the "effect number" of operator i, Ni, for a specified number of iterations as the total number of times this operator is involved in generating the links serving as the parents in the iteration after the specified number of iterations. Also, let r_i be the operation rate of operator i. To adjust the operation rate in the next period of iterations, it is quite logical to have

$$r_i' \propto r_i \left(1 + \frac{N_i}{\sum_{i=1}^{4} N_i} \right)$$

where r_i' is the operation rate of operator i in the next period of iterations. The proportional factor in the above relationship should be such that the number of offspring generated in each iteration remains unchanged. For this condition to hold, it is required that

$$\prod_{i=1}^{4} r_i' = \prod_{i=1}^{4} r_i.$$

Therefore,

$$r_i' = k\, r_i \left(1 + \frac{N_i}{\sum_{i=1}^{4} N_i} \right)$$

$$\text{where } k = \left(\prod_{i=1}^{4} \left[1 + \frac{N_i}{\sum_{i=1}^{4} N_i} \right] \right)^{\frac{1}{4}}$$

Another consideration in the varying operation rate method as described above is that some of the operation rates may keep decreasing to a point that they may no longer effect the iterative process. Theoretically an operator can not effect in generating offspring if the operation rate is smaller or equal to 1. So the operation rate of each operator should be given a lower bound which is larger than one, i.e. $l_i > 1$, where l_i is the lower bound of operation rate of operator i.

The process of changing operation rates is thus as follows:

0. Set operation rates r_1, r_2, r_3 and r_4.
1. Let $m=1$ and m indexes each pre-specified number of iterations.
2. Let $N_i=0$ for $i=1$, 2, 3 and 4.
3. Apply the genetic algorithm to generate offspring from a set of parents and determine the most fit offspring as the set of parents for the next iteration. Increment N_i, $i=1,2,3$ and 4, by 1 if operator i is involved in generating one of parents used in the next operation, considering all of the parents.
4. Repeat step 3 for the number of iterations specified.

5. Compute the new operation rates r_i', i=1,2,3 and 4, using

$$r_i' = k\, r_i \left(1 + \frac{N_i}{\sum_{i=1}^{4} N_i} \right)$$

$$\text{where } k = \left(\prod_{i=1}^{4} \left[1 + \frac{N_i}{\sum_{i=1}^{4} N_i} \right] \right)^{\frac{1}{4}}$$

6. If $r_i' < l_i$, let $r_i' = l_i$ for i=1, 2, 3 and 4. Then compute the new operation rates as $r_i'' = k'\, r_i'$

$$\text{where } k' = \left(\frac{\prod_{i=1}^{4} r_i}{\prod_{i=1}^{4} r_{i'}} \right)^{\frac{1}{4}}$$

Repeat this step until the new operation rates satisfy the lower bound requirement.

7. Increment m by 1 and repeat steps 2 to 6.

8.2 Results

The type two problem in Section 6.2 was used for comparing fixed operation rate versus changing operation rate in implementing the genetic algorithm. Figure 28 shows the results of 1,000 iterations, of which the first 100 iterations are repeated in Figure 29 for clarity.

Case (1) is a fixed operation rate approach. Cases (2), (3), and (4) have the same initial operation rates but different initial estimates of the solution. Case (5) is another changing operation rate approach which has different initial operation from case (2), (3) and (4). The fixed operation rates are listed in Table 9, and the two sets of initial rates of changing operation rates are listed in Table 10. The operation rates in all of the variable cases are updated each 20 iterations.

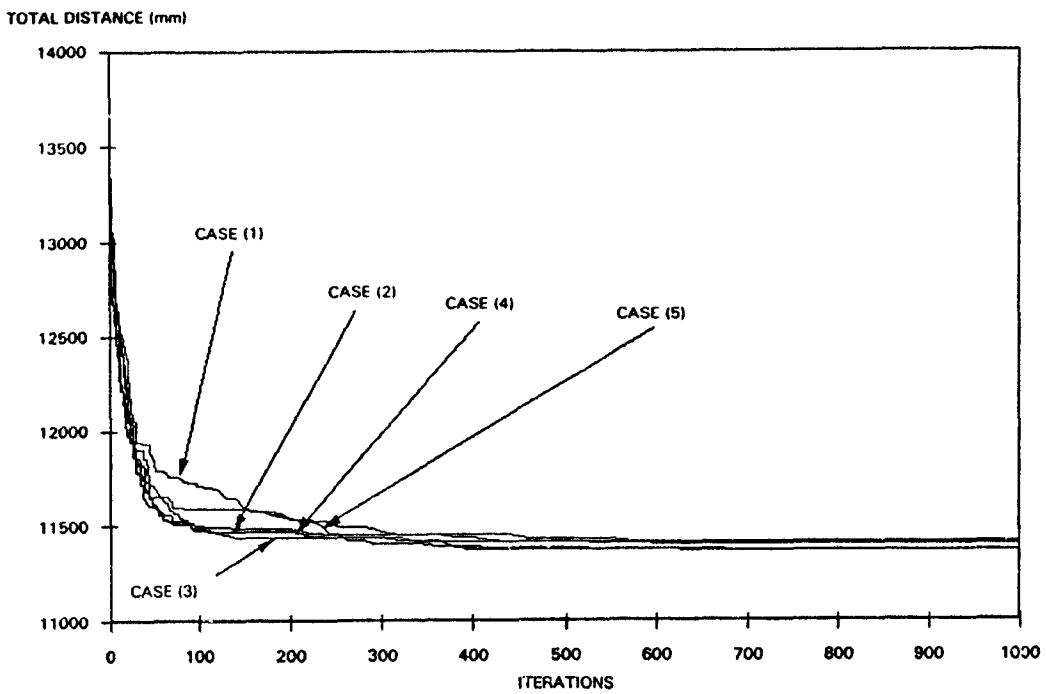


Figure 28. Solutions from genetic algorithm with different operation rates

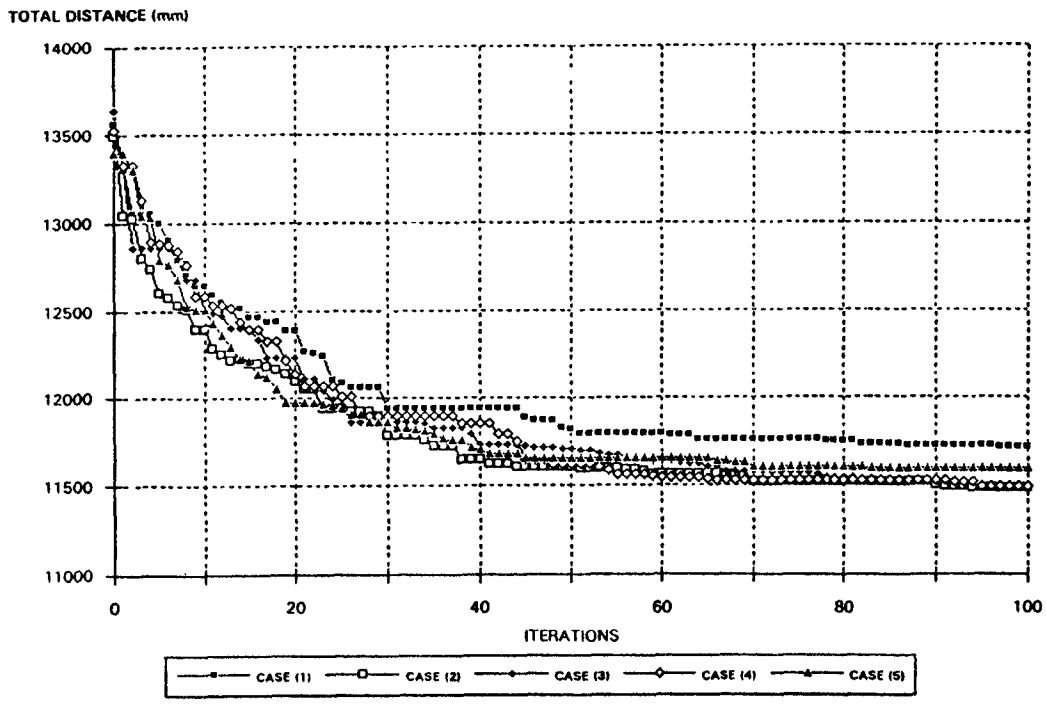


Figure 29. First 100 iterations of the calculations

Table 9. Fixed operation rates

Operator	Operation Rate
Crossover Operator	1.72
Inversion Operator	1.52
Rotation Operator	1.52
Mutation Operator	1.55

Table 10. Initial rates of variable operation rates. The operation rate 1 corresponds to cases (2), (3) and (4) and operation rate 2 corresponds to case (5) in Figures 28 and 29.

Operator	Operation Rate 1	Operation Rate 2
Crossover Operator	1.5	3.0
Inversion Operator	1.2	1.2
Rotation Operator	1.2	1.2
Mutation Operator	1.5	1.5

The results of all the changing rate operations are better than the result of the fixed rate operation after a large number of iterations. Figures 30, 31, 32 and 33 show the change of operation rates for cases (2), (3), (4) and (5) in Figures 28 and 29. In all four cases the crossover rate is the highest in the beginning of the iterative process and goes down after some iterations for these four calculations. Since the initial estimates are all randomly generated, this observation indicates that the crossover operator is more effective when the links are more distinct from one another. In other words, the crossover operator has a better chance than the other operators to generate good offspring if the parents are quite different in pattern. After many iterations the better offspring may be similar in pattern, and thus the crossover operator becomes less effective. The same observation, i.e. the crossover rate is the highest in the beginning of iteration but decreases afterwards, is made in the study of type 1 and type 3 problems.

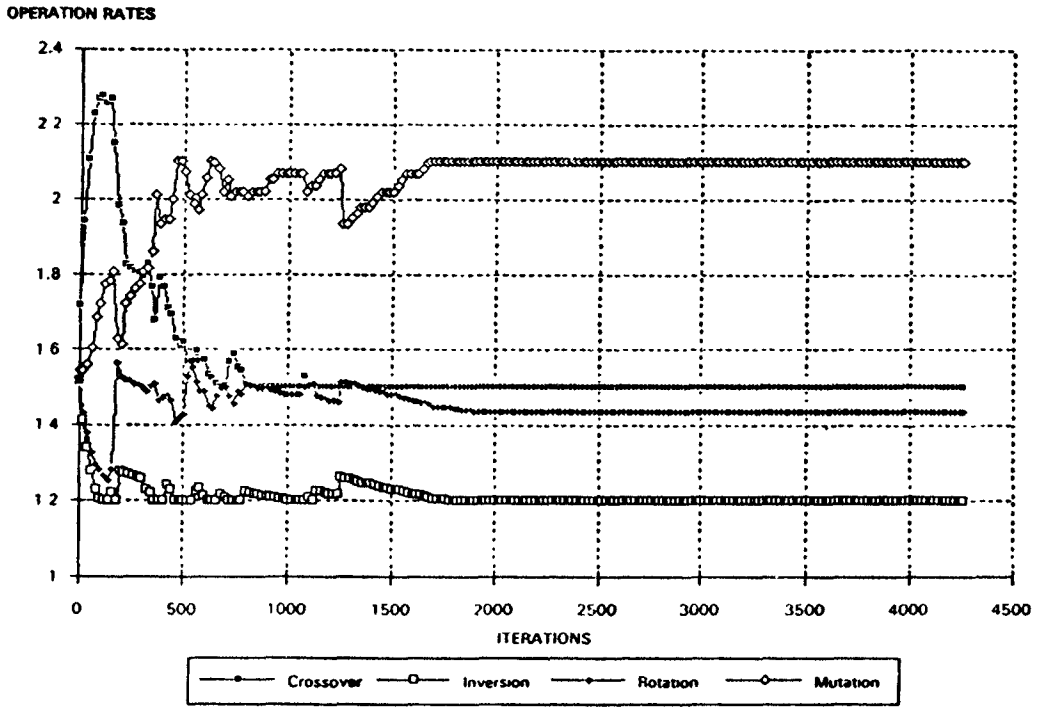


Figure 30. The operation rates of changing rate calculation (I)

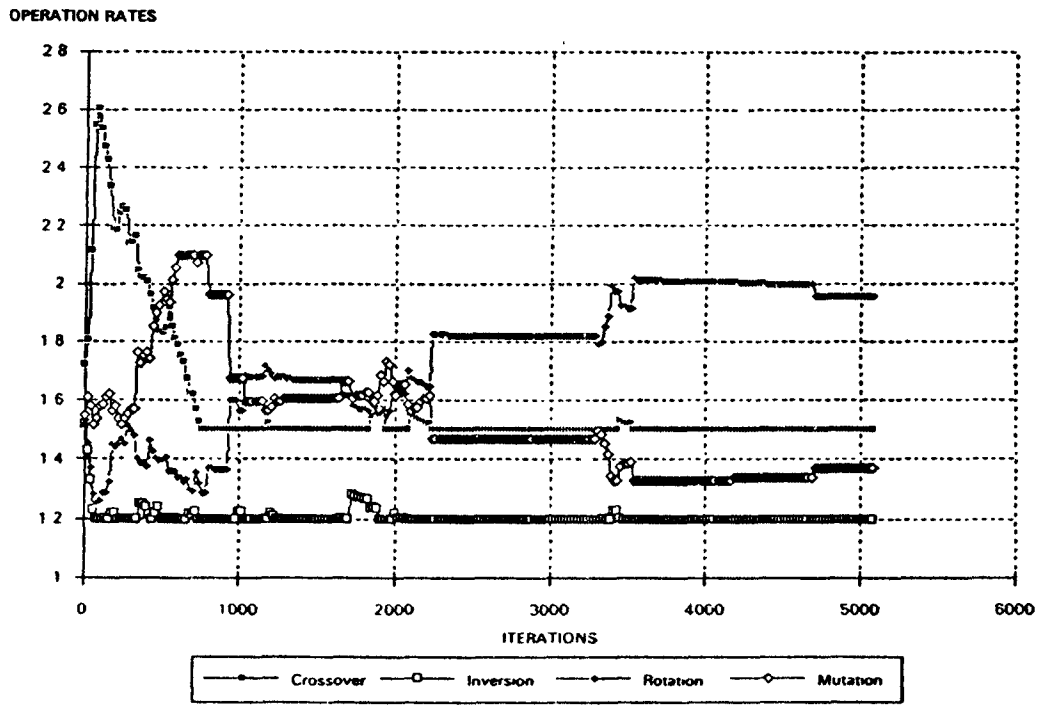


Figure 31. The operation rates of changing rate calculation (II)

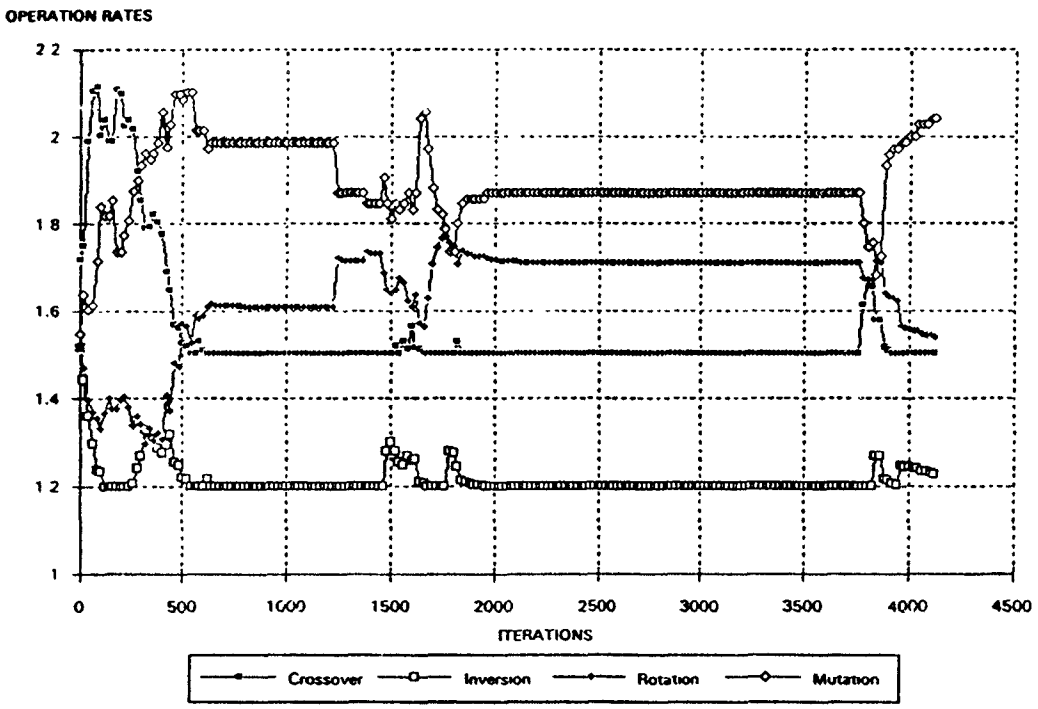


Figure 32. The operation rates of changing rate calculation (III)

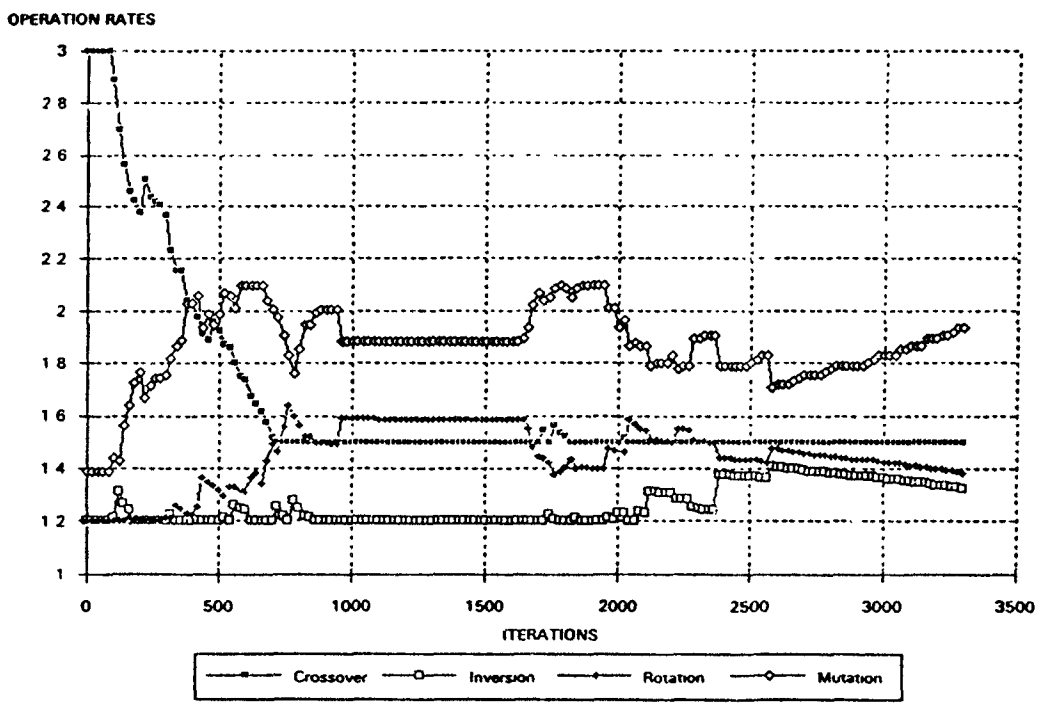


Figure 33. The operation rates of changing rate calculation (IV)

Another interesting observation is found from comparing the effectiveness of each operator. We use r_i to represent the operation rate of operator i as above. Let N_i represent the effect number of operator i for the whole iterative process. We define the effectiveness index of operator i as

$$\eta_i = \frac{N_i}{N_1} \frac{r_1 - 1}{r_i - 1}.$$

By this definition $\eta_1=1$ serves the reference for comparison. We let i=1 represent the crossover operator. Table 11 shows the effectiveness indices for the four operators calculated from their average operation rates and effect numbers for case (2) in Figure 28. Table 12 lists the effectiveness indices for the four operators for all of the five cases in Figure 28. It clearly shows that the crossover rate is the most effective among the four operators. The inversion operator is the second most effective in all the cases where the operation rates are allowed to vary during the iteration.

Table 11. Calculation of effectiveness indices of operators for case (2) in Figure 28.

Operator	Average Operation Rate	Effect numbers	Effectiveness index
Crossover Operator	1.66	36416	1
Inversion Operator	1.26	5236	0.363
Rotation Operator	1.45	5072	0.205
Mutation Operator	1.87	11211	0.233

Table 12. Effectiveness indices of operators for the five cases in Figure 28.

	Crossover	Inversion	Rotation	Mutation
Case 1 (fixed)	1	0.255	0.381	0.405
Case 2 (var.)	1	0.363	0.205	0.233
Case 3 (var.)	1	0.409	0.209	0.126
Case 4 (var.)	1	0.464	0.155	0.380
Case 5 (var.)	1	0.424	0.158	0.161

Chapter 9

Conclusion

Described in this thesis is a genetic algorithm and its application to solve scheduling and assignment problems in PCB assembly. Three types of PCB assembly planning problems have been studied: the traveling salesperson problem, the pick-and-place problem, and the moving board with time delay problem. All the problems are represented in the form such that the genetic algorithm can be applied to minimize assembly time. Four genetic operators are used: order crossover operator, inversion operator, rotation operator, and mutation operator.

The major contributions of this thesis study are as follows:

1) A new application of the genetic algorithm approach is introduced to solve PCB assembly planning problems.
2) The planning problems in various types of PCB assembly machines are properly modeled such that they can be solved using the genetic algorithm approach.
3) The approach is shown to be beneficial over other methods for solving PCB assembly planning problems, i.e. it is capable of solving the planning problem for many classes of assembly machines instead of just one class of assembly machines.

4) A multi-link genetic algorithm is introduced for dealing with the complexity of the assembly planning problem encounted in some types of assembly machines.

5) A varying operation rate technique is devised and is shown to generate better results than the fixed rate method on algorithm efficiency.

6) A rotation operator is introduced for the genetic algorithm and is shown to contribute to solving the PCB planning problems.

7) A quantitative study of relative effectiveness among the various genetic operators is performed. The crossover operator is shown to be the most effective.

References

[1] Ball, Michael O. and Michael J. Magazine (March-April 1988). "Sequencing of insertion in printed circuit board assembly," *Operations Research*, Vol 36 (2), 192-201.

[2] Leipälä, T. and O. Nevalainen (1989). "Optimization of the movements of a component placement machine," *European Journal of Operational Research*, Vol. 38, 167-177.

[3] Bard, J. F., R. W. Clayton and T. A. Feo (1990). "Machine Setup and Component Placement in Printed Circuit Board Assembly," Operational Research Group, College of Engineering, University of Texas, Austin, TX 78712.

[4] Ji, Zhiming, Ming C. Leu, and Hermean Wong (1991). "Application of Linear Assignment Model for Planning of Robotic PC Board Assembly,"

[5] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor, MI.

[6] Jong, K. D. (1980). "Adaptive System Design : A Genetic Approach," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-10 (9), 566-574.

[7] Shahookar, K. and P. Mazumder (May 1990). "A genetic approach to standard cell placement using meta-genetic parameter optimization," *IEEE Transactions on Computer-Aided Design*, Vol. 9 (5), 500-511.

[8] Englander, A. C. (1985). "Machine Learning of Visual Recognition using Genetic Algorithms," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 197-201.

[9] Davis, L. (1985). "Job Shop Scheduling with Genetic Algorithms," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 136-140.

[10] Stadnyk, I. (1987). "Schema Recombination in Pattern Recognition Problems," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 27-35.

[11] Grefenstette, J. J., R. Gopal, B. Rosmaita, and D. Van Gucht (1985). "Genetic algorithms for the traveling salesman problem," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 160-168.

[12] D. E. Goldberg and R. Lingle "Alleles, loci, and the traveling salesman problem," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 1985.

[13] Leicht, T., R. D. Schraft and E. Wolf (1989). *PCB Assembly Systems 1989 - The international Directory of SMT/Insertion Equipment*, IFS Publications/Springer - Verlag.

[14] I. M. Oliver, D. J. Smith, and J, R, C, Holland. (1985) "A study of permutation crossover operators on the traveling salesman problem," *Proceeding of International Conference on Genetic Algorithms and Their Applications*, 224-230.

[15] Grefenstette, J. J. (1986). "Optimization of Control Parameters for Genetic Algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16 (1), 122-128.

[16] Emmons, Hamilton, A. Dale Flowers, et al. (1987) *STORM personal version 2.0.*

APPENDIX A

```
/*********************************************************************/
/*                                                                   */
/*          Genetic Algorithm for planning of PCB Assembly           */
/*                                                                   */
/*********************************************************************/

#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <dos.h>
#include <float.h>
#include <math.h>
#include <fcntl.h>
#include <conio.h>
#include <io.h>
#include <string.h>
#include <alloc.h>

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

#define nMax1stGene 200
#define nMax2ndGene 200
#define nMaxLink 2
#define nMaxGroup 400
#define nMaxInitGroup 100
#define nMaxAGroup 500
#define nMaxProblems 3
#define nMaxMovement 2
#define nMaxIterations 9999
#define nRateBase 2
#define rUCRate 3.0
#define rLCRate 1.5
#define rUIRate 2.1
#define rLIRate 1.2
#define rURRate 2.1
#define rLRRate 1.2
#define rUMRate 2.1
#define rLMRate 1.3
#define nReDefault 10
#define lRan 65539
#define l232 2147483648
#define r232 0.4655613e-9
#undef  abs

#define MessageWindow   window( 2,  2, 40, 14)
#define TimeWindow      window( 2, 16, 40, 19)
#define ErrorWindow     window( 2, 21, 40, 23)
#define IterationWindow window(42,  4, 79, 15)
#define AlgorithmWindow window(42, 17, 79, 23)
#define WholeWindow     window( 1,  1, 80, 24)

/*********************************************************************/
/*                                                                   */
/*                        Type Definitions                           */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/*  rPoint : two dimensional point with real coordinates             */
/*  cGroup : group chain                                             */
/*  cLink  : link chain                                              */
/*                                                                   */
/*********************************************************************/

typedef struct rpoint {
   float x, y;
} rPoint;
```

```
typedef struct cgruop {
   int No1stLink;
   int No2ndLink;

   long nC;
   long nI;
   long nR;
   long nM;

   int nCr;
   int nIr;
   int nRr;
   int nMr;

   int *link1st;
   int *link2nd;

   float rW;
} cGroup;

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void DBand(int, int, int, int);
void DSHLine(int, int, int);
void DSVLine(int, int, int);

void InitialSettings();
void SetupMenu();
void ReadSettings();
void InitialGeneration();
void ApplyCrossOver();
void ApplyInversion();
void ApplyRotation();
void ApplyMutation();
void SortandSelect();
void SwitchAlgorithmMessage();
void ErrorHandler(int);
void CloseandExit();
void RefreshOperationRates();
void ToNormalText();
void ToSpecialText();
void ShowOperationRate();
void GenerateRandomSque(int, float raRand[], int naRand[]);
void SortwithOrder(int, float raR[], int naR[]);
void AdjustRates(float, float, float, float);
void EvaluateLinks();
void EvaluateLinks1();
void EvaluateLinks2();
void EvaluateLinks3();
void BCrossOne(int fpc1Link[], int fpc2Link[],
       int fpc3Link[], int nDim);
void BInverseOne(int fpc1Link[], int fpc2Link[], int nDim);
void BRotateOne(int fpc1Link[], int fpc2Link[], int nDim);
void BMutateOne(int fpc1Link[], int fpc2Link[], int nDim);
void MyRandomize();

int RandomChooseEP(int);
int RandomChooseLP(int);

float Distance(struct rpoint p1, struct rpoint p2);
float DistMax(struct rpoint p1, struct rpoint p2);
float MyRand();

/*********************************************************************/
/*                                                                   */
/*                        Global Variables                           */
/*                                                                   */
```

```
/*********************************************************************/
/*                                                                   */
/*  fInFile    : a inout file                                        */
/*  fMinFile   : a outout file for storing minimum values            */
/*  fAveFile   : a outout file for storing average values            */
/*  fPreFile   : predefinied data file                               */
/*                                                                   */
/*  nSaveDatatoFile : indicator for saving intermediate data         */
/*    to file or not, 1=Yes, 0=No                                    */
/*                                                                   */
/*  nTypeofProblem : indicator for problem type                      */
/*    1 : traveling salesperson problem                              */
/*    2 : pick and place problem with feeder assignment              */
/*    3 : sequencial place with feeder assignment                    */
/*                                                                   */
/*  nTypeofMovement : indicator for type of movement                 */
/*    1 : linear movement                                            */
/*    2 : x-y table movement (Manhattan movement)                    */
/*                                                                   */
/*  nNIG : No. of initial group                                      */
/*  nNIP : amount of intermediate population                         */
/*  nNP : amount of final population                                 */
/*  nLineMe : Display line of message window                         */
/*  nLineIte : Display line if iteration window                      */
/*  loop : index of iteration                                        */
/*  ch : input character                                             */
/*  sdBDate : date structure of begin date                           */
/*  stBTime : time structure of begin time                           */
/*  sdEDate : date structure of end date                             */
/*  stETime : time structure of end time                             */
/*  naType : type array for pick and place problem                   */
/*  raLink : coordinates of components                               */
/*  raFeeder : coordinates of feeders                                */
/*  sInFile : input file name                                        */
/*  sMinFile : output file name for minimum values                   */
/*  sAveFile : output file name for average values                   */
/*  sAlgoM : message for the genetic algorithm                       */
/*  nLink1 : number of genes in link 1                               */
/*  nLink2 : number of genes in link 2                               */
/*    0  : traveling salesperson problem                             */
/*    >0 : pick and place problem with feeder location assignment    */
/*  rMin : minimum of the links                                      */
/*  rAve : average of the links                                      */
/*  nbase : base line for operation rate output                      */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

int nSaveDatatoFile=1, nTypeofProblem, nTypeofMovement;
int loop=1, nNIG, nNP, nNIP, nLineAl=1, nLineIte=1;
int nLink1, nLink2, nbase=7, nPreDefGroup, nRefresh;
int nOC=0, nOI=0, nOR=0, nOM=0, nCompGap;
int nL[nMaxGroup], nA[nMaxGroup], nB[nMaxGroup];

char gch, *sInFile, *sMinFile, *sErrorM[20];
char *sAlgoM[7], *sUnit, *sPreFile, *sAveFile;
char *sLinkFile, *sOFile, *sRateFile;

float rCrossOverRate;
float rInversionRate;
float rRotationRate;
float rMutationRate;

float rMin, rAve, rSwapTime;
float rL[nMaxGroup];
float rXS, rYS, rFS;

FILE *fInFile, *fMinFile, *fPreFile, *fLinkFile, *fRateFile, *fAveFile;

rPoint startp, raLink[nMax1stGene], raFeeder[nMax2ndGene];
cGroup paGroup[nMaxGroup], gaT[nMaxGroup];
```

```
int *fpk1[nMaxGroup],  *fpk2[nMaxGroup];
int naType[nMax1stGene];
long nSeed, nRV=100;

struct date sdBDate, sdEDate;
struct time stBTime, stETime;

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void AdjustRates(float rC1, float rI1, float rR1, float rM1) {
   int nT1, i;
   float rK, rT1, rT2;

   for (i=0, nT1=nNP+1; ((i<10) && (nT1>nNP)); i++) {
      rT1 = nNP;
      rT2 = nNIG;
      rK  = (rT1 / rT2) / (rC1*rI1*rR1*rM1);

      rK  = (rK<=0) ? 1 : sqrt(sqrt(rK));

      rC1 *= rK;
      rI1 *= rK;
      rR1 *= rK;

      if (rC1>rUCRate) rC1 = rUCRate;
      if (rC1<rLCRate) rC1 = rLCRate;
      if (rI1>rUIRate) rI1 = rUIRate;
      if (rI1<rLIRate) rI1 = rLIRate;
      if (rR1>rURRate) rR1 = rURRate;
      if (rR1<rLRRate) rR1 = rLRRate;

      rM1 = rT1 / floor(floor(floor(rT2*rC1)*rI1)*rR1);

      if (rM1>rUMRate) rM1 = rUMRate;
      if (rM1<rLMRate) rM1 = rLMRate;

      nT1 = floor(floor(floor(floor(rT2*rC1)*rI1)*rR1)*rM1+0.5);
   }


   if (i<10) {
      rCrossOverRate = rC1;
      rInversionRate = rI1;
      rRotationRate  = rR1;
      rMutationRate  = rM1;

      nNP = nT1;
   }
} /* of AdjustRates */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ApplyCrossOver() {
   int nN1, nN2, nNN, i;


   SwitchAlgorithmMessage();

   nNN  = nNIG;
   nNIP = floor(nNN * rCrossOverRate);

   for (i=nNN; i<nNIP; i++) {
```

```
      nN1 = RandomChooseLP(nNN);
      do {
         nN2 = RandomChooseLP(nNN);
      } while (nN1==nN2);

      paGroup[i].nC = ((paGroup[nN1].nC>paGroup[nN2].nC) ?
         paGroup[nN1].nC : paGroup[nN2].nC) + 1;

      paGroup[i].nCr = ((paGroup[nN1].nC>paGroup[nN2].nCr) ?
         paGroup[nN1].nCr : paGroup[nN2].nCr) + 1;

      BCrossOne(paGroup[nN1].link1st, paGroup[nN2].link1st,
         paGroup[i].link1st, paGroup[i].No1stLink);

      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Link Error at CrossOver Operator! 1\n");
         exit(-100);
      }

      switch (nTypeofProblem) {
         case 1 : break;

         case 2 :
         case 3 :
         BCrossOne(paGroup[nN1].link2nd, paGroup[nN2].link2nd,
            paGroup[i].link2nd, paGroup[i].No2ndLink);

         if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
            ErrorWindow;
            printf("Link Error at CrossOver Operator! 2\n");
            exit(-100);
         }
      }
   }
} /* of ApplyCrossOver */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ApplyInversion() {
   int nN1, nNN, i;


   SwitchAlgorithmMessage();

   nNN  = nNIP;
   nNIP = floor(nNN * rInversionRate);

   for (i=nNN; i<nNIP; i++) {
      nN1 = RandomChooseLP(nNN);

      paGroup[i].nI  = paGroup[nN1].nI  + 1;
      paGroup[i].nIr = paGroup[nN1].nIr + 1;

      BInverseOne(paGroup[nN1].link1st, paGroup[i].link1st,
         paGroup[i].No1stLink);


      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Link Error at Inversion Operator 1!\n");
         exit(-100);
      }

      switch (nTypeofProblem) {
         case 1 : break;
```

```
         case 2 :
         case 3 :
         BInverseOne(paGroup[nN1].link2nd, paGroup[i].link2nd,
            paGroup[i].No2ndLink);

         if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
            ErrorWindow;
            printf("Link Error at Inversion Operator 2!\n");
            exit(-100);
         }
      }
   }
} /* of ApplyInversion */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ApplyMutation() {
   int nN1, nNN, i;


   SwitchAlgorithmMessage();

   nNN  = nNIP;
   nNIP = floor(nNN * rMutationRate + 0.5);

   for (i=nNN; i<nNIP; i++) {
      nN1 = RandomChooseLP(nNN);

      paGroup[i].nM  = paGroup[nN1].nM  + 1;
      paGroup[i].nMr = paGroup[nN1].nMr + 1;

      BMutateOne(paGroup[nN1].link1st, paGroup[i].link1st,
         paGroup[i].No1stLink);


      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Link Error at Mutation  Operator 1!\n");
         exit(-100);
      }
      switch (nTypeofProblem) {
         case 1 : break;

         case 2 :
         case 3 :
         BMutateOne(paGroup[nN1].link2nd, paGroup[i].link2nd,
            paGroup[i].No2ndLink);

         if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
            ErrorWindow;
            printf("Link Error at Mutation  Operator 2!\n");
            exit(-100);
         }
      }
   }
} /* of ApplyMutation */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ApplyRotation() {
   int nN1, nNN, i;

   SwitchAlgorithmMessage();
```

```
   nNN  = nNIP;
   nNIP = floor(nNN * rRotationRate);

   for (i=nNN; i<nNIP; i++) {
      nN1 = RandomChooseLP(nNN);

      paGroup[i].nR  = paGroup[nN1].nR  + 1;
      paGroup[i].nRr = paGroup[nN1].nRr + 1;

      BRotateOne(paGroup[nN1].link1st, paGroup[i].link1st,
         paGroup[i].No1stLink);


      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Link Error at Rotation  Operator 1!\n");
         exit(-100);
      }
      switch (nTypeofProblem) {
         case 1 : break;

         case 2 :
         case 3 :
         BRotateOne(paGroup[nN1].link2nd, paGroup[i].link2nd,
            paGroup[i].No2ndLink);

         if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
            ErrorWindow;
            printf("Link Error at Rotation  Operator 2!\n");
            exit(-100);
         }
      }
   }
} /* of ApplyRotation */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void BCrossOne(int fpc1Link[], int fpc2Link[], int fpc3Link[], int nDim) {
   int i, j, k, nN1, nT1;

   nN1 = RandomChooseEP(nDim);

   if ((nN1<0) || (nN1>=nDim)) {
      ErrorWindow;
      printf("Error at RandomChooseEP when Crossover\n");
   }

   for (j=0; j<nN1; j++) {
      fpc3Link[j] = fpc1Link[j];
   }

   for (i=nN1, j=0; ((j<nDim) && (i<nDim)); j++) {
      nT1 = 0;

      for (k=0; ((k<nN1) && (nT1==0)); k++) {
         if (fpc2Link[j]==fpc3Link[k]) {
            nT1 = 1;
         }
      }

      if (nT1==0) {
         fpc3Link[i] = fpc2Link[j];
         i++;
      }
   }
```

```
   if (i<nDim-1) {
      ErrorWindow;
      clrscr();
      printf(" Error Link when Appling CrossOver at loop %4d\n", loop);
      exit(-100);
   }
} /* of BCrossOne */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void BInverseOne(int fpc1Link[], int fpc2Link[], int nDim) {
   int j, k, nN1, nN2, nT1;

   nN1 = RandomChooseEP(nDim);
   do {
      nN2 = RandomChooseEP(nDim);
   } while (nN1==nN2);

   if (nN1>nN2) {
      nT1 = nN1;
      nN1 = nN2;
      nN2 = nT1;
   }


   for (k=0; k<nN1; k++) {
      fpc2Link[k] = fpc1Link[k];
   }

   for (k=nN1, j=nN2; k<nN2+1; k++, j--) {
      fpc2Link[k] = fpc1Link[j];
   }

   for(k=nN2+1; k<nDim; k++) {
      fpc2Link[k] = fpc1Link[k];
   }

} /* of BInverseOne */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void BMutateOne(int fpc1Link[], int fpc2Link[], int nDim) {
   int j, k, nN1, nN2, nT1;

   nN1 = RandomChooseEP(nDim);
   do {
      nN2 = RandomChooseEP(nDim);
   } while (nN1==nN2);


   for (k=0; k<nDim; k++) {
      fpc2Link[k] = fpc1Link[k];
   }

   nT1           = fpc2Link[nN1];
   fpc2Link[nN1] = fpc2Link[nN2];
   fpc2Link[nN2] = nT1;
} /* of BMutateOne */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
```

```
/*********************************************************************/

void BRotateOne(int fpc1Link[], int fpc2Link[], int nDim) {
   int j, k, nN1, nN2, nT1;

   nN1 = RandomChooseEP(nDim);
   do {
      nN2 = RandomChooseEP(nDim);
   } while (abs(nN1-nN2)<2);

   if (nN1>nN2) {
      nT1 = nN1;
      nN1 = nN2;
      nN2 = nT1;
   }


   nT1 = RandomChooseEP(2);

   for (k=0; k<nN1; k++) {
      fpc2Link[k] = fpc1Link[k];
   }


   if (nT1==1) {
      fpc2Link[nN2] = fpc1Link[nN1];

      for (k=nN1; k<nN2; k++) {
         fpc2Link[k] = fpc1Link[k+1];
      }
   }
   else {
      fpc2Link[nN1] = fpc1Link[nN2];

      for (k=nN1; k<nN2; k++) {
         fpc2Link[k+1] = fpc1Link[k];
      }
   }


   for(k=nN2+1; k<nDim; k++) {
      fpc2Link[k] = fpc1Link[k];
   }
} /* of BRotateOne */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void CloseandExit() {
   float tt1, tt2, tt3;
   int i, *fpc1Link, nCt=0, nIt=0, nRt=0, nMt=0;
   FILE *fOFile;


   getdate(&sdEDate);
   gettime(&stETime);

   TimeWindow;

   gotoxy(2, 1);
   printf("Computer time used :");

   gotoxy(4, 2);
   printf("From : %2d/%2d/%4d  %2d:%02d:%02d.%02d",
      sdBDate.da_mon, sdBDate.da_day, sdBDate.da_year,
      stBTime.ti_hour, stBTime.ti_min, stBTime.ti_sec, stBTime.ti_hund);

   gotoxy(4, 3);
```

```
printf("To   : %2d/%2d/%4d  %2d:%02d:%02d.%02d",
   sdEDate.da_mon, sdEDate.da_day, sdEDate.da_year,
   stETime.ti_hour, stETime.ti_min, stETime.ti_sec, stETime.ti_hund);

fclose(fMinFile);
fclose(fAveFile);
fclose(fRateFile);

if ((fOFile=fopen(sOFile, "wt"))==NULL) {
   ErrorHandler(13);
}
else {
   fpc1Link = paGroup[0].link1st;

   for (i=0; i<paGroup[0].No1stLink; i++) {
      fprintf(fOFile, "%3d\n", fpc1Link[i]);
   }

   switch (nTypeofProblem) {
      case 1 : break;

      case 2 :
      case 3 :
      fpc1Link = paGroup[0].link2nd;

      for (i=0; i<paGroup[0].No2ndLink; i++) {
         fprintf(fOFile, "%3d\n", fpc1Link[i]);
      }
   }

   fprintf(fOFile, "Total Distance : %10.3f\n", paGroup[0].rW);

   fprintf(fOFile, "Computer time used :\n");

   fprintf(fOFile, "From : %2d/%2d/%4d  %2d:%02d:%02d.%02d\n",
      sdBDate.da_mon, sdBDate.da_day, sdBDate.da_year,
      stBTime.ti_hour, stBTime.ti_min, stBTime.ti_sec, stBTime.ti_hund);

   fprintf(fOFile, "To   : %2d/%2d/%4d  %2d:%02d:%02d.%02d\n",
      sdEDate.da_mon, sdEDate.da_day, sdEDate.da_year,
      stETime.ti_hour, stETime.ti_min, stETime.ti_sec, stETime.ti_hund);

   for (i=0; i<nNIP; i++) {
      nCt += paGroup[i].nC;
      nIt += paGroup[i].nI;
      nRt += paGroup[i].nR;
      nMt += paGroup[i].nM;
   }

   fprintf(fOFile, "Crossover Operator effects %5d times.\n", nCt);
   fprintf(fOFile, "Inversion Operator effects %5d times.\n", nIt);
   fprintf(fOFile, "Rotation  Operator effects %5d times.\n", nRt);
   fprintf(fOFile, "Mutation  Operator effects %5d times.\n", nMt);

   fclose(fOFile);
}


switch (nTypeofProblem) {
   case 1 :
   for (i=nNP-1; i>=0; i--) {
      free(fpk1[i]);
   }
   break;

   case 2 :
   case 3 :
   for (i=nNP-1; i>=0; i--) {
      free(fpk2[i]);
      free(fpk1[i]);
   }
```

```
      break;
   }


   WholeWindow;
   gotoxy(1, 24);
} /* of CloseandExit */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void DBand(int x1, int y1, int x2, int y2) {

   int i;

   if (((x2-x1)>1) && ((y2-y1)>1) && (x1>0) && (y1>0) && (x2<81) && (y2<26)) {
      WholeWindow;
   }
} /* of DBand */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void DSHLine(int x1, int x2, int y) {

   int i;

   if (((x2-x1)>1) && (x2<81) && (x1>0) && (y>0) && (y<26)) {
      WholeWindow;
   }
} /* of DSHLine */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void DSVLine(int x, int y1, int y2) {

   int i;

   if (((y2-y1)>1) && (y1>0) && (y2<26) && (x<81) && (x>0)) {
      WholeWindow;
   }
} /* of DSVLine */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ErrorHandler(int nErr) {
  ErrorWindow;
  printf(" Error %2d : %s %c", nErr, sErrorM[nErr], 7);
} /* of ErrorHandler */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void EvaluateLinks() {
```

```
   SwitchAlgorithmMessage();


   switch (nTypeofProblem) {
      case 1 :
      EvaluateLinks1();
      break;

      case 2 :
      EvaluateLinks2();
      break;

      case 3 :
      EvaluateLinks3();
      break;
   }
} /* of EvaluateLinks */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void EvaluateLinks1() {
   int i, j, nLP;
   float rTD;

   if (loop==1) {
      nLP = 0;
   }
   else {
      nLP = nNIG;
   }

   for (i=nLP; i<nNIP; i++) {
      for (j=0, rTD=0; j<(paGroup[i].No1stLink - 1); j++) {
         switch (nTypeofMovement) {
            case 1 :
            rTD += Distance(raLink[(paGroup[i].link1st)[j]],
               raLink[(paGroup[i].link1st)[j+1]]);
            break;


            case 2 :
            rTD += DistMax(raLink[(paGroup[i].link1st)[j]],
               raLink[(paGroup[i].link1st)[j+1]]);
            break;
         }
      }

      switch (nTypeofMovement) {
         case 1 :
         rTD += Distance(raLink[(paGroup[i].link1st)[j]],
            raLink[(paGroup[i].link1st)[0]]);
         break;


         case 2 :
         rTD += DistMax(raLink[(paGroup[i].link1st)[j]],
               raLink[(paGroup[i].link1st)[0]]);
         break;
      }

      paGroup[i].rW = rTD;
   }
} /* of EvaluateLinks1 */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
```

```
/*                                                                    */
/**********************************************************************/

void EvaluateLinks2() {
   int i, j, nLP;
   int *pN1, *pN2;
   float rTD;

   if (loop==1) {
      nLP = 0;
   }
   else {
      nLP = nNIG;
   }

   for (i=nLP; i<nNIP; i++) {
      pN1 = paGroup[i].link1st;
      pN2 = paGroup[i].link2nd;

      switch (nTypeofMovement) {
         case 1 :
         rTD = Distance(startp, raFeeder[pN2[naType[pN1[0]]]]);
         break;


         case 2 :
         rTD = DistMax(startp, raFeeder[pN2[naType[pN1[0]]]]);
         break;
      }


      for (j=0; j<(paGroup[i].No1stLink - 1); j++) {
         switch (nTypeofMovement) {
            case 1 :
            rTD += Distance(raFeeder[pN2[naType[pN1[j]]]], raLink[pN1[j]]);
            rTD += Distance(raLink[pN1[j]], raFeeder[pN2[naType[pN1[j+1]]]]);
            break;


            case 2 :
            rTD += DistMax(raFeeder[pN2[naType[pN1[j]]]], raLink[pN1[j]]);
            rTD += DistMax(raLink[pN1[j]], raFeeder[pN2[naType[pN1[j+1]]]]);
            break;
         }
      }


      switch (nTypeofMovement) {
         case 1 :
         rTD += Distance(raFeeder[pN2[naType[pN1[j]]]], raLink[pN1[j]]);
         rTD += Distance(raLink[pN1[j]], startp);
         break;


         case 2 :
         rTD += DistMax(raFeeder[pN2[naType[pN1[j]]]], raLink[pN1[j]]);
         rTD += DistMax(raLink[pN1[j]], startp);
         break;
      }

      paGroup[i].rW = rTD;
   }
} /* of EvaluateLinks2 */

/**********************************************************************/
/*                                                                    */
/*                                                                    */
/*                                                                    */
/**********************************************************************/

void EvaluateLinks3() {
```

```
    int i, j, k, j1, k1, nLP;
    int *pN1, *pN2;
    float rTD, rT1, rT2, rT3;


    if (loop==1) {
       nLP = 0;
    }
    else {
       nLP = nNIG;
    }

    for (i=nLP; i<nNIP; i++) {
       pN1 = paGroup[i].link1st;
       pN2 = paGroup[i].link2nd;

       for (j=0, k=nCompGap, rTD=0; j<paGroup[i].No1stLink; j++, k++) {
          j1 = j + 1;
          k1 = k + 1;
          if (j1>=paGroup[i].No1stLink) j1 -= paGroup[i].No1stLink;
          if (k >=paGroup[i].No1stLink) k  -= paGroup[i].No1stLink;
          if (k1>=paGroup[i].No1stLink) k1 -= paGroup[i].No1stLink;


          switch (nTypeofMovement) {
             case 1 :
             rT1 = Distance(raLink[pN1[j]], raLink[pN1[j1]]) / rXS;
             break;


             case 2:
             rT1 = fabs(raLink[pN1[j]].x - raLink[pN1[j1]].x) / rXS;
             rT3 = fabs(raLink[pN1[j]].y - raLink[pN1[j1]].y) / rYS;
             rT1 = (rT1>=rT3) ? rT1 : rT3;
             break;
          }


          rT2 = Distance(raFeeder[pN2[naType[pN1[k]]]],
             raFeeder[pN2[naType[pN1[k1]]]]) / rFS;

          rTD += (float) max(rSwapTime, (float) max(rT1, rT2));
       }

       paGroup[i].rW = rTD;
    }
}

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void GenerateRandomSque(int nDim, float raRand[], int naRand[]) {
    int i;
    float ra, rb, rc;

    if (nDim>nMax1stGene) {
       ErrorHandler(10);
       exit(-10);
    }

    rb = RAND_MAX;
    rc = nDim;

    for (i=0; i<nDim; i++) {
       ra = rand();
       raRand[i] = (ra * rc) / rb;
       naRand[i] = i;
    }
```

```
   SortwithOrder(nDim, raRand, naRand);
} /* GenerateRandomSque */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void InitialGeneration() {
   int   i, j, k;
   int   naRand[nMax1stGene];
   float raRand[nMax1stGene];


   AlgorithmWindow;
   ToSpecialText();
   gotoxy(2, nLineAl);
   cprintf("%s", sAlgoM[nLineAl - 1]);

   for (i=0; i<nNP; i++) {
      if ((paGroup[i].link1st=calloc(nLink1, sizeof(int)))==NULL) {
         ErrorHandler(8);

         WholeWindow;
         exit(-8);
      }


      fpk1[i] = paGroup[i].link1st;
      paGroup[i].No1stLink = nLink1;

      paGroup[i].nC = 0;
      paGroup[i].nI = 0;
      paGroup[i].nR = 0;
      paGroup[i].nM = 0;

      paGroup[i].nCr = 0;
      paGroup[i].nIr = 0;
      paGroup[i].nRr = 0;
      paGroup[i].nMr = 0;

      paGroup[i].rW = 0;


      switch (nTypeofProblem) {
         case 1 :
         paGroup[i].No2ndLink = 0;
         break;

         case 2 :
         case 3 :
         if ((paGroup[i].link2nd = calloc(nLink2, sizeof(int)))==NULL) {
            ErrorHandler(8);

            WholeWindow;
            exit(-8);
         }
         fpk2[i] = paGroup[i].link2nd;
         paGroup[i].No2ndLink = nLink2;
      }
   }


   if ((fPreFile = fopen(sPreFile, "rt"))==NULL) {
      nPreDefGroup = 0;
   }
   else {
      if (fscanf(fPreFile, "%d", &nPreDefGroup)<1) {
         nPreDefGroup = 0;
```

```
      }
      else {
         for (i=0; i<nPreDefGroup; i++) {
            for (k=0; k<paGroup[i].No1stLink; k++) {
               if (fscanf(fPreFile, "%d", &((paGroup[i].link1st)[k]))<1) {
                  ErrorHandler(7);

                  nPreDefGroup = i;
                  break;
               }
            }

            switch (nTypeofProblem) {
               case 1 : break;

               case 2 :
               case 3 :
               for (k=0; k<paGroup[i].No2ndLink; k++) {
                  if (fscanf(fPreFile, "%d", &((paGroup[i].link2nd)[k]))<1) {
                     ErrorHandler(7);

                     nPreDefGroup = i;
                     break;
                  }
               }
            }
         }
      }

      fclose(fPreFile);
   }

   for (i=nPreDefGroup; i<nNIG; i++) {
      GenerateRandomSque(paGroup[i].No1stLink, raRand, naRand);

      for (j=0; j<paGroup[i].No1stLink; j++) {
         (paGroup[i].link1st)[j] = naRand[j];
      }

      switch (nTypeofProblem) {
         case 1 : break;

         case 2 :
         case 3 :
         GenerateRandomSque(paGroup[i].No2ndLink, raRand, naRand);

         for (j=0; j<paGroup[i].No2ndLink; j++) {
            (paGroup[i].link2nd)[j] = naRand[j];
         }
      }
   }


   ToNormalText();
   gotoxy(2, nLineAl);
   cprintf("%s", sAlgoM[nLineAl++ - 1]);
} /* of InitialGeneration */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void InitialSettings() {
   struct time tR;
   long rR;


   textmode(C80);
   ToNormalText();
```

```
gettime(&tR);
rR  = (tR.ti_hund + tR.ti_sec + (tR.ti_min + tR.ti_hour*60)*60)*100;
nRV = rR % 65536;



sErrorM[0] = "Program terminated normally.";
sErrorM[1] = "Can't open input file.";
sErrorM[2] = "Can't open output file.";
sErrorM[3] = "Too many initial guesses!";
sErrorM[4] = "Too many components.";
sErrorM[5] = "Too many feeders.";
sErrorM[6] = "Feeder type error.";
sErrorM[7] = ".PRF file error.";
sErrorM[8] = "Out of memory.";
sErrorM[9] = "Unrecognizable problem.";
sErrorM[10] = "Random generation Error.";
sErrorM[11] = "Operation rate Error.";
sErrorM[12] = "Can't open link file.";
sErrorM[13] = "Can't open .SQE file";
sErrorM[14] = "Can't open .RAT file";
sErrorM[15] = "Unrecognizable movement.";


sAlgoM[0] = "     Generating Initial Guesses     ";
sAlgoM[1] = "     Applying CrossOver Operator    ";
sAlgoM[2] = "     Applying Inversion Operator    ";
sAlgoM[3] = "     Applying Rotation  Operator    ";
sAlgoM[4] = "     Applying Mutation  Operator    ";
sAlgoM[5] = "         Evaluating  Links          ";
sAlgoM[6] = "    Sort  and  Selecting  Links     ";


MyRandomize();
SetupMenu();

if ((fMinFile = fopen(sMinFile, "wt")) == NULL) {
   ErrorHandler(2);

   WholeWindow;
   gotoxy(1, 24);
   exit(-2);
}


if ((fAveFile = fopen(sAveFile, "wt")) == NULL) {
   ErrorHandler(2);

   WholeWindow;
   gotoxy(1, 24);
   exit(-2);
}


if ((fRateFile = fopen(sRateFile, "wt")) == NULL) {
   ErrorHandler(14);

   WholeWindow;
   gotoxy(1, 24);
   exit(-14);
}
fprintf(fRateFile, "step, Crossover, Inversion, Rotation, Mutation\n");


ReadSettings();
InitialGeneration();

getdate(&sdBDate);
gettime(&stBTime);
```

```
} /* of InitialSettings */

/*************************************************************************/
/*                                                                       */
/*                                                                       */
/*                                                                       */
/*************************************************************************/

void MyRandomize() {
   struct time tR;
   long rR;

   gettime(&tR);
   rR = (tR.ti_hund + tR.ti_sec + (tR.ti_min + tR.ti_hour*60)*60)*100;
   nSeed = rR % 65536;
} /* of MyRandomize */

/*************************************************************************/
/*                                                                       */
/*                                                                       */
/*                                                                       */
/*************************************************************************/

void ReadSettings() {
   FILE *fInFile;
   int i;


   if ((fInFile = fopen(sInFile, "rt")) == NULL) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }


   if (fscanf(fInFile, "%d", &nTypeofProblem)<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }

   if ((nTypeofProblem>nMaxProblems) || (nTypeofProblem<=0)) {
      ErrorHandler(9);

      WholeWindow;
      gotoxy(1, 24);
      exit(-9);
   }


   if (fscanf(fInFile, "%d", &nTypeofMovement)<1) {
      ErrorHandler(15);

      WholeWindow;
      gotoxy(1, 24);
      exit(-15);
   }

   if ((nTypeofMovement>nMaxMovement) || (nTypeofMovement<=0)) {
      ErrorHandler(15);

      WholeWindow;
      gotoxy(1, 24);
      exit(-15);
   }
```

```
if (fscanf(fInFile, "%d", &nNIG)<1) {
   ErrorHandler(1);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((nNIG>nMaxInitGroup) || (nNIG<=0)) {
   ErrorHandler(3);

   gotoxy(2, 2); printf("Automatically adjusted!");
   if (getch()==0)   getch();
   ErrorWindow;
   clrscr();

   nNIG = nMaxInitGroup;
}

if (fscanf(fInFile, "%d", &nNP)<1) {
   ErrorHandler(1);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((nNP>nMaxGroup) || (nNP<=nNIG)) {
   ErrorHandler(3);

   gotoxy(2, 2); printf("Automatically adjusted!");
   if (getch()==0)   getch();
   ErrorWindow;
   clrscr();

   nNP = nMaxGroup;
}

if (fscanf(fInFile, "%d", &nRefresh)<1) {
   ErrorHandler(1);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if (nRefresh<0) {
   ErrorHandler(3);

   gotoxy(2, 2); printf("Automatically adjusted!");
   if (getch()==0)   getch();
   ErrorWindow;
   clrscr();

   nRefresh = nReDefault;
}

if (fscanf(fInFile, "%f", &rCrossOverRate)<1) {
   ErrorHandler(11);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((rCrossOverRate>rUCRate) || (rCrossOverRate<rLCRate)) {
   rCrossOverRate = (rUCRate + rLCRate) / 2;
}

if (fscanf(fInFile, "%f", &rInversionRate)<1) {
   ErrorHandler(11);
```

```
   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((rInversionRate>rUIRate) || (rInversionRate<rLIRate)) {
   rInversionRate = (rUIRate + rLIRate) / 2;
}

if (fscanf(fInFile, "%f", &rRotationRate)<1) {
   ErrorHandler(11);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((rRotationRate>rURRate) || (rRotationRate<rLRRate)) {
   rRotationRate = (rURRate + rLRRate) / 2;
}

if (fscanf(fInFile, "%f", &rMutationRate)<1) {
   ErrorHandler(11);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}

if ((rMutationRate>rUMRate) || (rMutationRate<rLMRate)) {
   rMutationRate = (rUMRate + rLMRate) / 2;
}

AdjustRates(rCrossOverRate, rInversionRate, rRotationRate, rMutationRate);

if (fscanf(fInFile, "%d", &nLink1)<1) {
   ErrorHandler(1);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}


if (nLink1>nMax1stGene) {
   ErrorHandler(4);

   gotoxy(2, 2); printf("Automatically adjusted!");
   if (getch()==0)   getch();
   ErrorWindow;
   clrscr();

   nLink1 = nMax1stGene;
}

switch (nTypeofProblem) {
   case 1 : break;

   case 2 :
   case 3 :
   if (fscanf(fInFile, "%d", &nLink2)<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }

   if (nLink2>nMax2ndGene) {
      ErrorHandler(5);
```

```
      gotoxy(2, 2); printf("Automatically adjusted!");
      if (getch()==0)   getch();
      ErrorWindow;
      clrscr();

      nLink2 = nMax2ndGene;
   }
}


if (fscanf(fInFile, "%s", sUnit)<1) {
   ErrorHandler(1);

   WholeWindow;
   gotoxy(1, 24);
   exit(-1);
}


switch (nTypeofProblem) {
   case 1 :                          /* Traveling Salesperson Problem */
   for (i=0; i<nLink1; i++) {
      if (fscanf(fInFile, "%f %f", &(raLink[i].x), &(raLink[i].y))<1) {
         ErrorHandler(1);

         WholeWindow;
         gotoxy(1, 24);
         exit(-1);
      }
   }

   switch (nTypeofMovement) {
      case 1 : break;

      case 2 :
      if (fscanf(fInFile, "%f %f", &rXS, &rYS)<1) {
         ErrorHandler(1);

         WholeWindow;
         gotoxy(1, 24);
         exit(-1);
      }
   }
   break;


   case 2 :                              /* Pick and Place Problem */
   if (fscanf(fInFile, "%f %f", &(startp.x), &(startp.y))<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }

   for (i=0; i<nLink1; i++) {
      if (fscanf(fInFile, "%f %f %d", &(raLink[i].x), &(raLink[i].y), &naType[i])<1) {
         ErrorHandler(1);

         WholeWindow;
         gotoxy(1, 24);
         exit(-1);
      }

      if ((naType[i]>nLink2) || (naType[i]<1)) {
         ErrorHandler(6);

         gotoxy(2, 2); printf("Automatically adjusted!");
         if (getch()==0)   getch();
```

```
      if (naType[i]<1) {
         naType[i] = 1;
      }
      else {
         naType[i] = nLink2;
      }
   }

   naType[i] -= 1;
}

for (i=0; i<nLink2; i++) {
   if (fscanf(fInFile, "%f %f", &(raFeeder[i].x), &(raFeeder[i].y))<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }
}

switch (nTypeofMovement) {
   case 1 : break;

   case 2 :
   if (fscanf(fInFile, "%f %f", &rXS, &rYS)<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }
}
break;


case 3 :              /* sequencial place plus feeder assignment */
for (i=0; i<nLink1; i++) {
   if (fscanf(fInFile, "%f %f %d", &(raLink[i].x), &(raLink[i].y), &naType[i])<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }

   if ((naType[i]>nLink2) || (naType[i]<1)) {
      ErrorHandler(6);

      gotoxy(2, 2); printf("Automatically adjusted!");
      if (getch()==0)   getch();

      if (naType[i]<1) {
         naType[i] = 1;
      }
      else {
         naType[i] = nLink2;
      }
   }

   naType[i] -= 1;
}

for (i=0; i<nLink2; i++) {
   if (fscanf(fInFile, "%f %f", &(raFeeder[i].x), &(raFeeder[i].y))<1) {
      ErrorHandler(1);

      WholeWindow;
      gotoxy(1, 24);
      exit(-1);
   }
```

```
      }

      switch (nTypeofMovement) {
         case 1 :
         if (fscanf(fInFile, "%f %f", &rXS, &rFS)<1) {
            ErrorHandler(1);

            WholeWindow;
            gotoxy(1, 24);
            exit(-1);
         }
         break;

         case 2 :
         if (fscanf(fInFile, "%f %f %f", &rXS, &rYS, &rFS)<1) {
            ErrorHandler(1);

            WholeWindow;
            gotoxy(1, 24);
            exit(-1);
         }
      }

      if (fscanf(fInFile, "%f", &rSwapTime)<1) {
         ErrorHandler(1);

         WholeWindow;
         gotoxy(1, 24);
         exit(-1);
      }

      if (fscanf(fInFile, "%d", &nCompGap)<1) {
         ErrorHandler(1);

         WholeWindow;
         gotoxy(1, 24);
         exit(-1);
      }
   }

   fclose(fInFile);

   MessageWindow;
   gotoxy(20, nbase-2);   printf("%3d", nNIG);
   gotoxy(20, nbase-1);   printf("%3d", nNP);
   gotoxy( 2, nbase-3);

   switch (nTypeofProblem) {
      case 1 :
      printf("Traveling Salesperson Problem.");
      break;

      case 2 :
      printf("Pick and Place, Fixed.");
      break;

      case 3 :
      printf("Pick and Place, Moving.");
      break;
   }

   ShowOperationRate();
} /* of ReadSettings() */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void RefreshOperationRates() {
```

```
   int nT1, nT2, i, j;
   float rTC=0.0, rTI=0.0, rTR=0.0, rTM=0.0, rTT;

   for (i=0; i<nNIG; i++) {
      rTC += paGroup[i].nCr;
      rTI += paGroup[i].nIr;
      rTR += paGroup[i].nRr;
      rTM += paGroup[i].nMr;

      paGroup[i].nCr = 0;
      paGroup[i].nIr = 0;
      paGroup[i].nRr = 0;
      paGroup[i].nMr = 0;
   }


   rTT = rTC + rTI + rTR + rTM;
   if (rTT==0) rTT = 1;


   rTC = (nRateBase + rTC / rTT) * rCrossOverRate;
   rTI = (nRateBase + rTI / rTT) * rInversionRate;
   rTR = (nRateBase + rTR / rTT) * rRotationRate;
   rTM = (nRateBase + rTM / rTT) * rMutationRate;

   AdjustRates(rTC, rTI, rTR, rTM);

   ShowOperationRate();

   fprintf(fRateFile, "%4d, %6.3f, %6.3f, %6.3f, %6.3f\n",
      loop, rCrossOverRate, rInversionRate, rRotationRate, rMutationRate);
} /* of RefreshOperationRates */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void SetupMenu() {
   int i, nb;

   WholeWindow;
   clrscr();

   DBand(1, 1, 80, 24);
   DSHLine( 1, 41,  4);
   DSHLine( 1, 41, 15);
   DSHLine( 1, 41, 20);
   DSHLine(41, 80, 16);
   DSHLine(41, 80,  3);
   DSVLine(41,  1, 24);
   DSVLine(49,  1, 16);
   DSVLine(65,  1, 16);

   gotoxy(41,  3);   printf("+");
   gotoxy(41,  4);   printf("¦");
   gotoxy(41, 15);   printf("¦");
   gotoxy(41, 16);   printf("+");
   gotoxy(41, 20);   printf("¦");
   gotoxy(49,  3);   printf("+");
   gotoxy(49, 16);   printf("-");
   gotoxy(65,  3);   printf("+");
   gotoxy(65, 16);   printf("-");

   gotoxy(42, 2);
   printf(" STEPS ¦    MINIMUM    ¦    AVERAGE  ");

   for (i=0; i<7; i++) {
      gotoxy(43, 17+i);
      printf("%s", sAlgoM[i]);
```

```
   }

   MessageWindow;
   gotoxy(6,   1);      printf("Genetic Algorithm for Planning");
   gotoxy(13,  2);      printf("of PCB Assembly");
   gotoxy(2, nbase-2);  printf("Initial Guesses :");
   gotoxy(2, nbase-1);  printf("Population      :");
   gotoxy(2, nbase);    printf("Crossover Rate  :");
   gotoxy(2, nbase+1);  printf("Inversion Rate  :");
   gotoxy(2, nbase+2);  printf("Rotation  Rate  :");
   gotoxy(2, nbase+3);  printf("Mutation  Rate  :");
   gotoxy(2, nbase+5);  printf("press ESC to stop");

   ShowOperationRate();

} /* of SetupMenu */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ShowOperationRate() {
   MessageWindow;

   gotoxy(20, nbase);   printf("%5.2f", rCrossOverRate);
   gotoxy(20, nbase+1); printf("%5.2f", rInversionRate);
   gotoxy(20, nbase+2); printf("%5.2f", rRotationRate);
   gotoxy(20, nbase+3); printf("%5.2f", rMutationRate);
} /* of ShowOperationRate */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void SortandSelect() {
   float rT1, rT2;
   int i, j, k;

   SwitchAlgorithmMessage();

   for (i=0; i<nNIP; i++) {
      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Error link before SortandCollect\n");
         exit(-100);
      }

      if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
         ErrorWindow;
         printf("Error link before SortandCollect\n");
         exit(-100);
      }
   }

   for (i=0; i<nNIP; i++) {
      nA[i] = i;
      rL[i] = paGroup[i].rW;
   }

   SortwithOrder(nNIP, rL, nA);


   nL[0] = nA[0];
   for (i=1, j=0, k=1; i<nNIP; i++) {
      if (rL[i]==rL[i-1]) {
         nB[j] = nA[i];
         j++;
```

```
   }
   else {
      nL[k] = nA[i];
      k++;
   }
}

for (i=k, j=0; i<nNIP; i++, j++) {
   nL[i] = nB[j];
}


rMin = paGroup[nL[0]].rW;
rT1  = 0;

for (i=0; i<nNIP; i++) {
   gaT[i].link1st = paGroup[nL[i]].link1st;

   switch (nTypeofProblem) {

   case 1 : break;

   case 2 :
      gaT[i].link2nd = paGroup[nL[i]].link2nd;
   }

   gaT[i].nC = paGroup[nL[i]].nC;
   gaT[i].nI = paGroup[nL[i]].nI;
   gaT[i].nR = paGroup[nL[i]].nR;
   gaT[i].nM = paGroup[nL[i]].nM;
   gaT[i].rW = paGroup[nL[i]].rW;

   gaT[i].nCr = paGroup[nL[i]].nCr;
   gaT[i].nIr = paGroup[nL[i]].nIr;
   gaT[i].nRr = paGroup[nL[i]].nRr;
   gaT[i].nMr = paGroup[nL[i]].nMr;

   if (i<nNIG) {
      rT1 += gaT[i].rW;
   }
}

rT1 /= nNIG;
rAve = rT1;


for (i=0; i<nNIP; i++) {
   paGroup[i].link1st = gaT[i].link1st;

   switch (nTypeofProblem) {

   case 1 : break;

   case 2 :
      paGroup[i].link2nd = gaT[i].link2nd;
   }

   paGroup[i].nC = gaT[i].nC;
   paGroup[i].nI = gaT[i].nI;
   paGroup[i].nR = gaT[i].nR;
   paGroup[i].nM = gaT[i].nM;
   paGroup[i].rW = gaT[i].rW;

   paGroup[i].nCr = gaT[i].nCr;
   paGroup[i].nIr = gaT[i].nIr;
   paGroup[i].nRr = gaT[i].nRr;
   paGroup[i].nMr = gaT[i].nMr;
}


for (i=0; i<nNIG; i++) {
```

```
      if (DetectLink(paGroup[i].link1st, paGroup[i].No1stLink)!=0) {
         ErrorWindow;
         printf("Error link after SortandCollect\n");
         exit(-100);
      }

      if (DetectLink(paGroup[i].link2nd, paGroup[i].No2ndLink)!=0) {
         ErrorWindow;
         printf("Error link after SortandCollect\n");
         exit(-100);
      }
   }

   fprintf(fMinFile, "%10.3f\n", rMin);
   fprintf(fAveFile, "%10.3f\n", rAve);
} /* of SortandSelect */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void SortwithOrder(int nD, float raR[], int naR[]) {
   int i, j, k, nt;
   float rt;

   for (i=1; i<nD; i++) {
      j = 0;
      do {
         if (raR[i]<raR[j]) {
            rt = raR[i];
            nt = naR[i];

            for (k=i-1; k>=j; k--) {
               raR[k+1] = raR[k];
               naR[k+1] = naR[k];
            }

            raR[j] = rt;
            naR[j] = nt;
            j = i;
         }
         else j++;
      } while (j!=i);
   }
} /* of SortwithOrder */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void SwitchAlgorithmMessage() {
   int y, i;

   AlgorithmWindow;

   if (nLineAl==2) {
      y = 7;
   }
   else {
      y = nLineAl - 1;
   }

   gotoxy(2, y);
   cprintf("%s", sAlgoM[y - 1]);

   ToSpecialText();
   gotoxy(2, nLineAl);
```

```
   cprintf("%s", sAlgoM[nLineAl++ - 1]);

   if (nLineAl>7) nLineAl = 2;
   ToNormalText();
} /* of SwitchAlgorithmMessage */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ToSpecialText() {
   textcolor(LIGHTGREEN);
   textbackground(LIGHTRED);
}  /* of ToNormalText */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

void ToNormalText() {
   textcolor(YELLOW);
   textbackground(BLUE);
}  /* of ToNormalText */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

int RandomChooseEP(int nN) {
   float ra;

   if (nN<=0) {
      ErrorWindow;
      printf("RandomChooseEP Error, nN<=0!");
      exit(-110);
   }
   else {
      ra = floor(MyRand() * nN);
      return(ra);
   }
} /* of RandomChooseEP */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

int RandomChooseLP(int nN) {
   float ra, rb;

   if (nN<=0) {
      ErrorWindow;
      printf("RandomChooseLP Error, nN<=0!");
      exit(-110);
   }
   else {
      ra = sqrt(MyRand());
      rb = floor((1 - ra) * nN);

      if (rb<0)    rb = 0;
      if (rb>nN-1) rb = nN-1;

      return(rb);
   }
```

```
} /* of RandomChooseLP */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

float Distance(struct rpoint p1, struct rpoint p2) {
   float rT1, rT2, rT3;

   rT1 = p2.x - p1.x;
   rT2 = p2.y - p1.y;
   rT3 = rT1*rT1 + rT2*rT2;
   rT3 = (rT3 <= 0) ? 0 : sqrt(rT3);

   return(rT3);
} /* of Distance */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

float DistMax(struct rpoint p1, struct rpoint p2) {
   float rT1, rT2, rT3;

   rT1 = fabs(p2.x - p1.x);
   rT2 = fabs(p2.y - p1.y);
   rT3 = (rT1 >= rT2) ? rT1 : rT2;

   return(rT3);
} /* of DistMax */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

int DetectLink(int Link1[], int nDim) {
   int i;
   int nA[nMax1stGene];

   for (i=0; i<nDim; i++) {
      nA[i] = nDim;
   }

   for (i=0; i<nDim; i++) {
      nA[Link1[i]] = Link1[i];
   }

   for (i=0; i<nDim; i++) {
      if (nA[i]!=i) {
         return(-1);
      }
   }

   return(0);
}

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

float MyRand(){
   nRV = nRV * lRan + nSeed;
   if (nRV<0) {
```

```
      nRV += l232;
   }
   return(nRV * r232);
} /* of MyRand() */

/*********************************************************************/
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/

main() {
   int nDel=0;


   InitialSettings();

   do {
      IterationWindow;
      if (nLineIte<12) {
         gotoxy(2, nLineIte);
         printf("%5d", loop);
      }
      else {
         switch (nDel) {

            case 2 :
            delline();
            break;

            case 1 :
            nDel += 1;

            default :
            break;
         }
         gotoxy(2, nLineIte);
         printf("%5d ¦               ¦             ", loop);
      }


      if (nRefresh!=0) {
         if ((loop % nRefresh)==0) {
            RefreshOperationRates();
         }
      }


      if (loop==1) {
         fprintf(fRateFile, "%4d, %6.3f, %6.3f, %6.3f, %6.3f\n",
            loop, rCrossOverRate, rInversionRate,
            rRotationRate, rMutationRate);
      }


      ApplyCrossOver();
      ApplyInversion();
      ApplyRotation();
      ApplyMutation();
      EvaluateLinks();
      SortandSelect();


      IterationWindow;
      gotoxy(11, nLineIte);
      printf("%10.3f   ¦  %10.3f", rMin, rAve);
      if (nLineIte<12) {
         nLineIte++;
         if (nLineIte==12) nDel=1;
      }
```

```
      if (kbhit()) {
         gch=getch();
         if (gch==0) {
            gch=getch();
            gch=0;
         }
      }
   } while ((loop++<=nMaxIterations) && (gch!=27));


   CloseandExit();
   return(0);
} /* of main */
```