LAYOUT PLANNING WITH ISLES:

A GENETIC APPROACH

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Mechanical Engineering

By

Manas Ranjan Sahoo

&

Ashwini Kumar Maharana

Department of Mechanical Engineering

National Institute of Technology

Rourkela

2007

LAYOUT PLANNING WITH ISLES:

A GENETIC APPROACH

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

In

Mechanical Engineering

By

Manas Ranjan Sahoo

&

Ashwini Kumar Maharana

Under the guidance of

Prof. K. R. Patel

Department of Mechanical Engineering

National Institute of Technology

Rourkela

2007

National Institute of Technology

Rourkela

CERTIFICATE

This is to certify that the thesis entitled, **"Layout Planning with Isles: A Genetic Approach"** submitted by **Sri Manas Ranjan Sahoo** and **Sri Ashwini Kumar Maharana** in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Mechanical Engineering at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Prof. K.R.Patel

Date

Dept. of Mechanical Engineering

National Institute of Technology

Rourkela

ACKNOWLEDGEMENT

With great pleasure & deep sense of gratitude we would like to extent our sincere thanks to **Prof. K.R.Patel** for his valuable guidance & encouragement at each step of our project work.

We would like to express our sincere gratitude to **Prof. S.S.Mohapatra** for his guidance & encouragement at each step of the completion of our project work.

Date: -

Manas Ranjan Sahoo

Ashwini Kumar Maharana

Place:-

CONTENTS

ABSTRACT

Plant layout problems involve distributing different resources or departments in a given plant and achieving maximum efficiency for the services or goods being made or offered. To this end, plants are designed to optimize production flow from the first stage (i.e. as raw material) to finish product. However, optimization which is generally expressed either in terms of minimization (for example, of material handling costs) or of maximization (for example, the number of desired adjacencies in a qualitative chart) is not always feasible when real problems or real sizes are being handled. The level of complexity may turn out considerable as the number of parameters, restrictions and other variables considered in the study become larger. This kind of problem has been formulated, from a mathematical view point as a static quadratic assignment problem. However, the number of problems that are susceptible to being solved by optimization methods is very limited. Some alternatives have been called from the field of graph-theory, direct method algorithms, construction algorithms (such as CORELAP), and improvement algorithms (such as CRAFT).

Genetic algorithms (GAs) can be defined as meta heuristics based on the evolutionary process of natural systems. Since their inception, they have been applied to numerous optimization problems with highly acceptable results. The flexibility they provide means that restrictions, which more traditional methods of plant layout could hardly contemplate, can be incorporated or included in the problem parameters. The ability to include isles is just one example of this.

In this thesis work, an attempt is made to develop the algorithm for solving layout problem with real-life restriction like aisles, used in factories for the easy transfer of materials from one section to the other, using Genetic Algorithm.

LIST OF FIGURES

LIST OF TABLES

Page No.

Chapter- 1

INTRODUCTION

1.1 Introduction

The optimization and finalization of a plant layout i.e. deciding the location of the various departments involved, has been a challenge for decision makers in any industrial set up. Plant layout deals with the arrangement of work areas and equipments to produce the products economically and to provide good working environment for the workers. A good plant layout should provide ease of working, less health hazards, greater safety, reduced material handling, less congestion of materials, machines and men. Product layout, process layout, fixed position layout and group technology are different types of layout those are applied to different fields.

In our project we have consider facility layout problem. Facility layout problem in a manufacturing setting is defined as the determination of the relative locations for, and allocation of the available space among a given number of workstation. As for example in any job shop machines are arranged according to their function. The aim and objective of a facility layout problem has been to minimize the cost associated with material handling of the manufacturing system, as major expense of an industry is because of material handling. So now-a-days each and every industrialist is looking forward to reduce the material handling cost and for that it is required to have an optimal facility layout. An optimum assignment of departments would depend on the various inter-facility relationships. These relationships may be a quantitative and/or qualitative nature. The departments can also be of different sizes, areas, shapes and the type and amount of interaction between them may also vary. Thus the mass objective of the facility layout problem is to minimize the cost of interaction between the various departments subject to certain constraints. The constraints to the problem are the restricted area and there should be no overlap. The final layout generated through any procedure should satisfy all the constraints and at the same time should achieve the objective in the best possible way. The cost associated with material handling is expressed in terms of the work flow and the distance between the departments. In our project work we have dealt with combinatorial optimization problem like optimum assignment of n departments to a fixed area. We have solved the problem with two objectives.

(a) Considering total cost

(b) Considering total closeness rating

The objective function for total cost is

$$\text{Minimize } C = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (Fij \times Cij \times Dij)$$

$F_{ij} \rightarrow$ work flow from department i to department j

$D_{ij} \rightarrow$ distance between the departments i and j (centroid to centroid distance)

$C_{ij} \rightarrow$the unit cost to move one unit of workflow from department i to department j

n →the number of departments.

The objective function for closeness rating—

$$TCR = \sum_{i=1}^{i=n} (\text{closeness rating})_{\text{seq}\,[i][i+1]}$$

where closeness rating is being given by experts and it lies in between 0 and 1.

But achieving optimal layout is very tiresome and deterministic techniques are not computationally feasible. The quadratic assignment problems (QAP) formulation of the facility layout problem belongs to the class of NP-complete problems proposed by Garey and Johnson in 1979 and the size of the problems that can be solved by the existing optimal methods is limited (number of departments <= 15). So in order to handle larger problem size, suboptimal solutions are preferred over optimal solutions, those are computationally feasible. Solutions of facility layout problems can be obtained from traditional optimization such as direct method algorithms, construction algorithms (such as CORELAP, Moore and Lee 1967, ALDEP, Seehof and Evans, 1967) improvement algorithms (CRAFT, Armour and Buffa, 1963). Because of some limitations of the above mentioned algorithms, recently random search procedures such as Simulated Annealing (SA) and Genetic Algorithm (GA) have been applied to combinatorial optimization problem. And these two methods can also be used to tackle generalized facility layout problem i.e. problem free from constraints like unequal areas, irregular shapes etc.

Chapter- 2

OVERVIEW OF PLANT LAYOUTS

Requirements

Types

2.1 Overview of Plant Layouts

The basic objective of a plant layout is to ensure a smooth flow of work, material, and information through the system.

2.1.1 Requirements for effective layout:

- Minimize material handling costs;
- Utilize space efficiently;
- Utilize labor efficiently;
- Eliminate bottlenecks
- Facilitate communication and interaction between workers, between workers and their supervisors, or between workers and customers
- Reduce manufacturing cycle time and customer service time
- Eliminate wasted or redundant movement
- Facilitate the entry, exit, and placement of material, products, and people
- Incorporate safety and security measures; Promote product and service quality
- Encourage proper maintenance activities
- Provide a visual control of operations or activities
- Provide flexibility to adapt to changing conditions

2.1.2 Basic Layouts

There are 3 basic types of layouts:

- process
- product
- fixed - position

There are 3 hybrid types of layouts

- cellular
- flexible manufacturing systems
- mixed model assembly-lines

2.1.3 Process Layouts

Process layouts (also known as functional layouts) is a layout that groups similar activities together in departments of work centers according to the process or function that they perform. It offers characteristic of operations that serve different customers different needs. The equipment in a process layout is a general purpose one. The workers are skilled at operating the equipment in their department. The advantage of process layout is flexibility whereas the disadvantage is inefficiency. Process layouts are inefficient because jobs or customers do not flow through in an orderly fashion and so backtracking is common. Also, the workers may experience much "idle time" if they are waiting for more work to arrive from a different department. Material storage space in a process layout must be large to accommodate the large amount of in-process inventory. This inventory is high because material moves from work-center to work-center waiting to be processed. Finished goods inventory however is low because goods are being made for particular customers. Process layouts in manufacturing firms require flexible material handling equipment (such as forklifts) that can follow multiple paths, move in any direction, and carry large loads of in-process goods. All areas of the facility must have timely access to the material handling equipment. Process layouts in service firms require large aisles for customers to move back and forth and ample display space to accommodate different customer preferences.

2.1.4 Product Layout

Product layouts (also known as assembly lines) arrange activities in a line according to the sequence of operations that need to be performed to assemble a particular product. In product layout each product should have its own "line". Product layouts are suitable for mass production or repetitive operations in which demand is steady and volume is high. Its because of this product layouts are more autonomous than process layouts. The advantage of the product layout is its efficiency and ease of use whereas the disadvantage is its inflexibility. Each product must have a completely different assembly-line set up in product layout. The major concern in a product layout is balancing the assembly line so that no one workstation becomes a bottleneck and holds up the flow of work through the line. A product layout needs material moved in one direction along the assembly line and always in the same pattern. The most common material handling equipment used in product layouts is the conveyor which can be automatic (at a steady speed), or paced by the workers. In product layout the aisles are

narrow because material is moved only one way and it is not moved very far. Scheduling of the conveyors, once they are installed, is simple-the only variable is how fast they should operate. Storage space along an assembly line is quite small because in-process inventory is consumed in the assembly of the product as it moves down the assembly line. Finished good inventory may require a separate warehouse for storage before they are sold.

2.1.5 Fixed-Position Layouts

Fixed-Position layouts are layouts which are used in projects in which the product is too fragile, bulky, or heavy to move for example: ships, houses and aircraft. In this type of layout equipment, material and workers are brought to the production site. The equipment is often left on-site because it is too expensive to move frequently. The workers on such job sites are highly skilled at performing the special tasks that they are requested to do. In such a process the fixed costs would be low and variable costs would be high.

2.1.6 Designing Process Layouts

One main objective of the process layout is to minimize material handling costs. This implies that departments that incur the most interdepartmental movement should be located closest to one another. There are 2 techniques used to design layouts - block diagramming, and relationship diagramming.

2.1.6.1 Block Diagramming

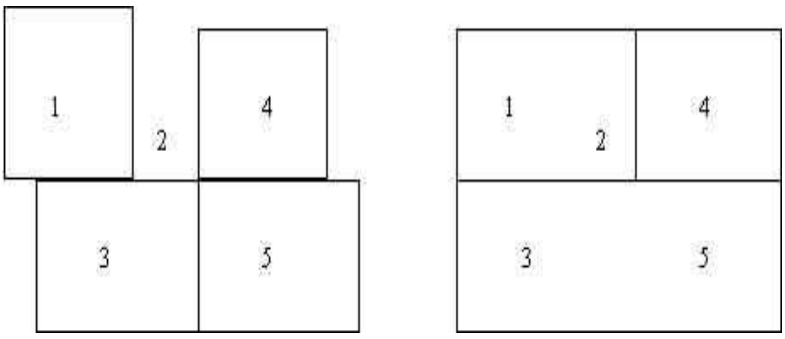


Fig. 2.1

Block Diagram- initial & final

In block diagramming historical or predicted movement of material in the existing or proposed facility must be analyzed. This info is usually provided with a from/to chart, or load summary chart. This gives the average number of unit loads moved between departments. A unit load can be a single unit, a pallet of material, a bin of material, or a crate of material-however material is normally moved from location to location. The next step in designing the layout is to calculate the *composite movements* between departments and rank them from most movement to least movement. Composite movement refers to the back-and-forth movement between each pair of departments. Finally, trial layouts are placed on a grid that graphically represents the relative distances between departments.

2.1.6.2 Relationship Diagramming

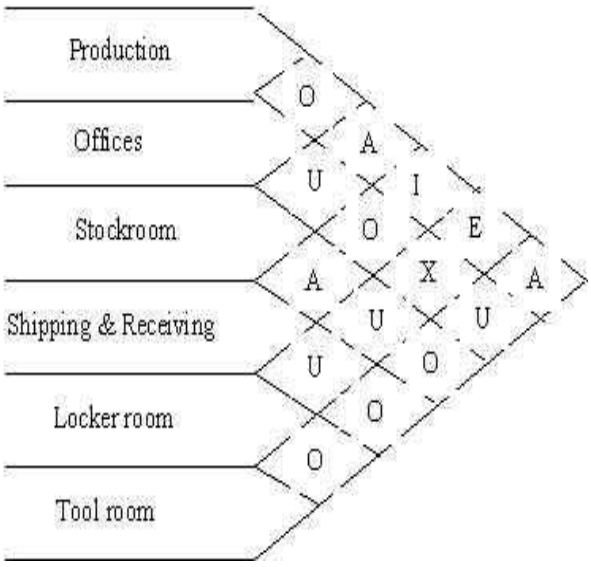


Fig. 2.2

Relationship Diagram

For situations in which quantitative data is hard to obtain it is more relevant to use a relationship diagramming technique. Richard Muther developed a format for displaying manager preferences for departmental locations, known as **Muther's grid.** Muther's diagram uses codes and letters to represent how close departments are to one another. The info from Muther's diagram can be used to make a relationship diagram to evaluate a current layout or proposed layouts.

2.1.7 Computerized layout solutions

Several computer programs exist that assist in designing process layouts. The best know are CRAFT (Computerized Relative Allocation of Facilities Technique) and CORELAP (Computerized Relationship Layout Planning)

Basically the computer program is given layout data and it makes a recommendation.

2.1.8 Service Layout

Most service organizations use process layouts because of the variability in customer requests for service. Service organizations look to maximize profits per unit of display space, rather than minimize customer flow. The layout must be aesthetically pleasing as well as functional.

2.1.9 Designing Product Layouts

The main objective of a product layout is to arrange workers or machines in a line according to the operations that need to be performed. Thus it would seem that the layout could be determined by following the order of assembly. To maximize efficiency on the assembly line line-balancing must be considered.

Line balancing is an attempt to equalize the amount of work at each work station. It cuts down on idle time for the workers. There are two constraints to line balancing - precedence requirements and cycle time restrictions. Precedence requirements are physical restrictions on the order in which operations are performed on the assembly line. Cycle time refers to the maximum amount of time the product is allowed to spend at each work station if the schedule production rate is to be reached. When designing a product layout line balancing must be completed with these two factors being considered.

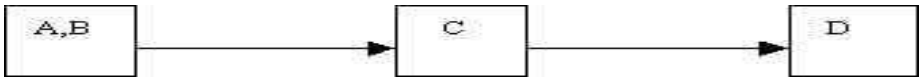


Fig. 2.3

Assembly Line

2.1.10 Computerized Line Balancing

Because complicated calculations are involved in line balancing software packages have been developed to assist this process. IBM's COMSOAL (Computer Method for Sequencing Operations for Assembly Lines) and GE's ASYBL (Assembly Line Configuration Program) are 2 popular packages widely used in line balancing.

2.1.11 Hybrid Layouts

Hybrid layouts modify and/or combine some aspects of product and process layouts.There are 3 hybrid layouts discussed. They are:

- Cellular layouts
- Flexible Manufacturing systems
- Mixed-model assembly lines

2.1.12 Cellular Layouts

Cellular layouts attempt to combine the flexibility of a process layout with the efficiency of a product layout. Based on the concept of group technology (GT), dissimilar machines are grouped into work centers, called *cells,* to process parts with similar shapes or processing requirements. The layout of machines *within* each cell resembles a small assembly line. Production flow analysis (PFA) is a group technology technique that reorders part routing matrices to identify families of parts with similar processing requirements.

2.1.12.1 Advantages of cellular layout

- Reduced material and Transit time
- Reduced Setup time
- Reduced work-in-process inventory
- Better use of human resources
- Easier to control
- Disadvantages of a cellular layout
- Inadequate part families
- Poorly balanced cells
- Expanded training and scheduling of workers

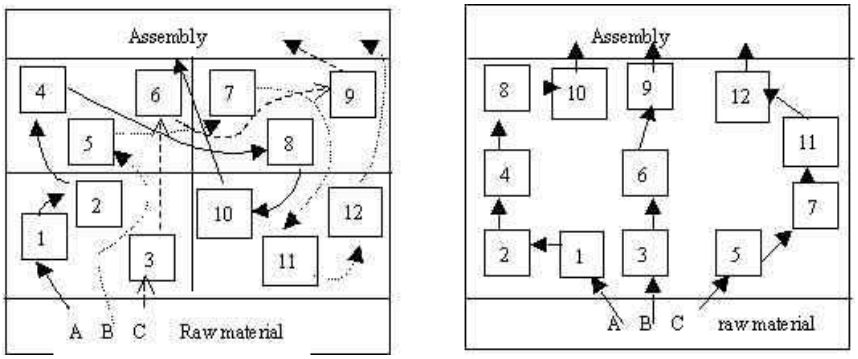


Fig.- 2.4

Cellular Layout

2.1.13 Flexible Manufacturing Systems

A flexible manufacturing system can produce an enormous variety of items. It is large, complex, and expensive. The emphasis for FMS is on automation. Computers run all the machines that complete the process. Not many industries can afford traditional FMS hence the trend is towards smaller versions call flexible manufacturing cells.

2.1.14 Mixed-Model Assembly Lines

The problem with the traditional layout is that it does take into account consumers' change in demand. Others adopted a mixed-model assembly line technique instead. There are several steps involved in a mixed-model assembly. The first step is to reduce the amount of time needed to change over the line to produce different models. Then they trained their workers to perform a variety of tasks and allowed them to work at more than one workstation on the line. Then the organization must change the way the line is arranged and scheduled. There are also several factors to consider in designing a mixed-model assembly line:

- Line Balancing
- U-shaped lines
- Flexible workforce
- Model sequencing
- Line Balancing
- The time to complete a task can vary from model to model
- When planning the completion time an array of values is used
- Otherwise, mixed-model lines are balanced in much the same way as single-model lines

CHAPTER- 3

INTRODUCTION TO GENETIC ALGORITHM

3.1 Introduction to genetic algorithm

Genetic Algorithm (GA), proposed by Holland (1975), is heuristic search and optimization techniques that imitate the natural selection and biological evolutionary process. The characteristic features of individual species of a population are governed by the presence or absence of genes and their position. The different traits are passed on from one generation to the next through different biological processes, which operate on the genetic structure. By this process of the genetic change and survival of the fittest, a population will adapt to the environment results.

In a genetic algorithm, an individual (any feasible solution to the problem) is an element of population. Here the population is a subset of the solution space i.e. held in hand at any instant the solution process. The magnitude of the population depends on the size and the nature of the problem, and the size of computer memory being used. Each element of the population is called a chromosome. In each generation chromosomes are selected by the roulette wheel principle to act as parents. This principle is basically a process by which a good parent (solution) gets a higher probability of being selected compared to the bad ones. In GA the genetic operators commonly employed for obtaining a new population from the current population are crossover, mutation, and a selection policy based on the survival of the fittest principle. The crossover operator is used to produce new offspring from selected pairs of solution in the current population. Mutation is performed by arbitrarily changing one or more elements in the solution string. The selection policy maintains a constant population size by allowing only better-fit solutions to survive. A constant population size is necessary for the algorithm to converge to a stable population. As a result, eventually the population will consist of solutions, which are well suited to the problem specifications (i.e. near optimal).

CHAPTER-4

GENETIC ALGORITM USED IN LAYOUT PROBLEM

Population Representation

Selection Policy

Operators

4.1 GENETIC ALGORITHM USED IN LAYOUT PROBLEM

4.1.1 POPULATION REPRESENTATION:

The integers denote the facilities and their positions in the string denote the positions of the facilities in the layout. For example, in case of the 8/8 problem with two rows and four columns, the rows are numbered 0 and 1 and the columns 0, 1, 2 and 3.The string positions are numbered 0 to 7.The position in the layout of the i^{th} integer in the string is given by

Row: integer part of (I divided by number of columns)

Column: remainder of (I divided by number of column)

Thus for the 8/8 problem the locations are coded as follows:

0 1 2 3

4 5 6 7

The following assignment of the 8/8 problems

7 6 5 3

8 4 1 2

will be represented by the solution string 7 6 5 3 8 4 1 2

At the start of each run of the program, the user is asked to specify the required number of rows and column, as well as the number of departments to be located in the layout. Immediately the locations are coded as discussed above.

4.1.2 FEASIBILITY/INFEASIBILITY OF THE SOLUTIONS IN THE PROBLEM:

For QAP we see that the total numbers of assignments, which are feasible, are n. However, if the restriction that one facility should be assigned to exactly at one location is relaxed i.e. infeasible assignments are allowed, and then the total number of assignments is n^n . The major issue involved is that weather to allowed infeasible solutions in the population or not

have been mentioned above. Here we show that, for a QAP infeasible solutions cannot be allowed in the population. It may be easily seen the ratio of the number of feasible solutions to the total number of solutions, allowing for infeasibility, approaches zero as n increases.

$$\lim_{n \to \infty} \frac{n!}{n^n} = \lim_{n \to \infty} (2\prod n) \times \frac{\left(\frac{n}{e}\right)^n}{n^n}$$

$$= \lim_{n \to \infty} \frac{\sqrt{(2 \Pi n)}}{e^n} = 0$$

Consequently, the probability of feasible solutions existing in a randomly generated initial population without imposing feasible conditions becomes zero for large n. It may be noted for n=8 only 0.2% of the total solutions are feasible. Hence in a population of 100 random candidates not even one is expected to be feasible.

4.1.3 CALCULATION OF DISTANCE:

The method used in the software is variable size column. Suppose an arrangement, as 4,2,5,1,6,3. The dimensions of the columns for this example will be calculated as follows:

D4		D1
D2		D6
D5		D3

Column 1:

Surface to be laid out = 12 + 15 + 10 =37

Height of the surface = 8

Width of the column = 37/8 = 4.625

Column 2:

Surface to be laid out = 10 + 15 + 10 =35

Height of the surface = 8

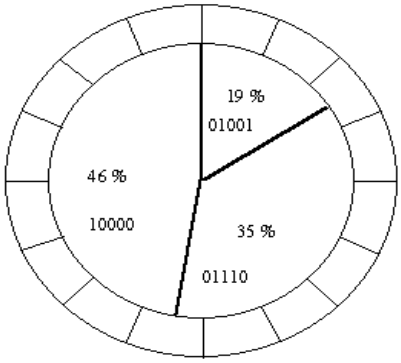
Width of the column = 35/8 = 4.375

So the distance between any two department = mod (x1-x2) + mod (y1-y2)

Where x and y is centroid of a department.

4.1.4 SELECTION POLICY:

The Roulette Wheel Method simply chooses the strings in a statistical fashion based solely upon their relative (i.e. percentage) fitness values. To look abstractly at this method, consider the roulette wheel below, which is based on the previous example above.



Roulette Wheel Selection

Fig. 4.1

When selecting the three strings that will be placed in the mating pool, the roulette wheel is spun three times, with the results indicating the string to be placed in the pool. It is obvious from the above wheel that there's a good chance that string 10000 will be selected more than once. This is fine. Multiple copies of the same string can exist in the mating pool. This is even desirable, since the stronger strings will begin to dominate, eradicating the weaker ones from the population. There are difficulties with this, as it can lead to premature convergence on a local optimum.

4.1.5 CROSSOVER:

From the above discussion we see that the solutions maintained in the population have to be all-feasible. Hence crossover operation has to be such that it generates only feasible offspring from a pair of feasible parents. Other desirable features are that the crossover should maintain diversity but at the same time should maintain structural arrangements, which may be contributing to better quality of the solutions. In the crossover operator new strings are created by exchanging information among strings. In most crossover operators, two strings are picked from the mating pool at random and some portions of the strings are exchanged between the strings. A single point crossover operator is performed by randomly choosing a crossing site along the string and by exchanging all bits on the right side of the crossing site as shown:

$$0\,0\,|\,0\,0\,0 \quad \Rightarrow \quad 0\,0\,|\,1\,1\,1$$

$$1\,1\,|\,1\,1\,1 \qquad\qquad 1\,1\,|\,0\,0\,0$$

Position based crossover, partially mapped crossover (PMX), order crossover (OX) are different types of crossover operator. In this case, we have tried out the PMX for the test problems.

4.1.6 Partially Mapped Crossover (PMX):

This type of crossover was proposed by Goldberg and Linble. It builds an offspring choosing a subsequence arrangement from one parent and preserving the order and position of as many arrangement as possible from the other parent.

For example the two parents

P1	1	2	3	4	5	6	7	8	9

P2	4	5	2	1	8	7	6	9	3

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

O1	X	X	X	1	8	7	6	X	X

O2	X	X	X	4	5	6	7	X	X

This swapping also defined a series of mapping

0 – 4, 8 – 5, 7 – 6, 6 – 7.

Then we can fill further gene (from the original parents), for which there is no conflict.

O1	X	2	3	1	8	7	6	X	9

O2	X	X	2	4	5	6	7	9	3

Finally the first X in the offspring 1 is replaced by 4, because of the mapping 1 – 4. Similarly the second X in the offspring O1 is replaced by 5 and the X in the offspring O2 are replaced by 1 and 8.The off springs are

O1	4	2	3	1	8	7	6	5	9

O2	1	8	2	4	5	6	7	9	3

4.1.7 Ordered Crossover:

Davis proposed this type of crossover. It built offspring by choosing a subsequence of a gene sequence from one parent and preserving the relative order of gene from the other parents. For example (for two cut point marked by color)

P1	1	2	3	4	5	6	7	8	9

P2	4	5	2	1	8	7	6	9	3

O1	X	2	3	4	5	6	7	X	9

O2	X	X	2	1	8	7	6	9	3

Reverse of P2: 9 3 4 5 2 1 8 7 6

Remove 4 5 6 7 already in the first P1

We get then 9 3 2 1 8

Finally offspring is

O1	3	4	5	1	8	7	6	9	2

O2	2	1	8	4	5	6	7	9	3

CHAPTER- 5

ALGORITHM FOR OPTIMIZATION

5.1 ALGORITHM FOR OPTIMIZATION

STEP 1:

Reproduction or Generation operator is used to generate random sequences of different departments for a facility layout problem. The number of such sequences (species) in the population pool and the number of departments are given as input (say n and m).

STEP 2:

Fitness value or total cost and closeness rating associated with each sequences thus generated in the first step, are calculated using the above mentioned formula

For total cost—

$$C = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (Fij \times Cij \times Dij)$$

For closeness rating—

$$TCR = \sum_{i=1}^{i=n} (\text{closeness rating})_{\text{seq}\,[i][i+1]}$$

STEP 3:

Selection policy- On the basis of total cost or total closeness rating the sequences are sorted in descending order. Then with the help of roulette wheel algorithm two sequences are selected.

STEP 4:

Position based crossover is used as a Crossover operator in our program. The two sequences thus obtained from selection policy are considered to be parent strings for crossover operator.

The algorithm for Position based crossover –

For example say we consider 9 departments

Parent1 : 1 2 3 4 5 6 7 8 9

* * * * *

and star marks indicate the fixed position. These are randomly chosen.

Parent2 : 2 6 8 1 4 9 3 7 5

= = = = =

double underlines indicate, the departments in the positions fixed in parent will remain as it is

offspring: 1 2 3 4 9 6 7 8 5

STEP 5:

Mutation Operator

The offspring obtained in step 4 are also mutated to get two different offspring or mutated strings. We used position-based mutation to remove a department from one potion and are put in another in an offspring obtained from step 4.These positions are selected randomly.

STEP 6:

Again fitness value is calculated for the mutated string, thus generated in step 5

STEP 7:

Again sorting is done and the population pool matrix is updated by discarding the two strings with strings with worst fitness value (as the pop size is always maintained constant). Thus the new sequence is generated.

Now from step 3 to step 7, the process is repeated for a given number of generations so that finally all sequence is the population or generation pool (new sequence matrix) will give some total cost.

CHAPTER- 6

RESULTS AND DISCUSSION

6.1 RESULTS AND DISCUSSION

The general facility layout problem takes into account the following constraints:

(a) unequal areas

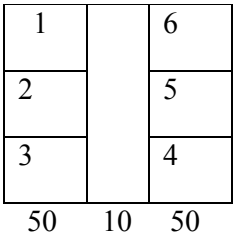
(b) irregular shapes

(c) a certain groups of unit squares should be placed together

(d) considering aisle in between certain departments

In this work, a simplified model, considering aisle in between certain departments i.e. constraint (d) has modeled as a simplified version of generalized facility layout problem. The area of each department is taken equal. Two objectives as mentioned in model description have been taken into account. These two models have been considered separately and solved using Genetic Algorithm. The two test problems that we have dealt with are as follows:

1. To find out optimal sequence of six departments with a 10cm aisle in between the departments.



The distance matrix, the flow matrix, and the closeness rating matrix given as input to the program are given below

Distance Matrix

0	10	20	80	70	60
10	0	10	70	60	70
20	10	0	60	70	80
80	70	60	0	10	20
70	60	70	10	0	10
60	70	80	20	10	0

Table-6.1

Flow matrix

0	10	5	8	3	7
10	0	2	5	9	1
5	2	0	4	6	8
8	5	4	0	11	9
3	9	6	11	0	6
7	1	8	9	6	0

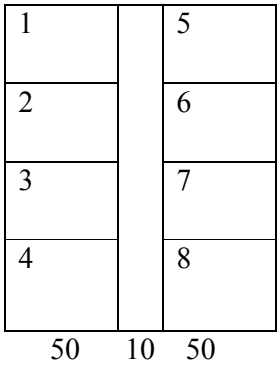
Table-6.2

Closeness rating matrix

0	3	6	1	9	6
3	0	3	9	6	9
6	3	0	6	9	1
1	9	6	0	3	6
9	6	9	3	0	3
6	9	1	6	3	0

Table-6.3

1. To find out optimal sequence of eight departments with a 10cm aisle in between the departments.



The distance matrix and the flow matrix given as input to the program are given below

DISTANCE MATRIX

0	10	20	30	60	70	80	90
10	0	10	20	70	60	70	80
20	10	0	10	80	70	60	70
30	20	10	0	90	80	70	60
60	70	80	90	0	10	20	30
70	60	70	80	10	0	10	20
80	70	60	70	20	10	0	10
90	80	70	60	30	20	10	0

Table-6.4

FLOW MATRIX

0	10	5	8	3	7	4	5
10	0	2	5	9	1	3	2
5	2	0	4	6	8	1	2
8	5	4	0	11	9	5	9
3	9	6	11	0	6	6	7
7	1	8	9	6	0	7	6
4	3	1	5	6	7	0	8
5	2	2	9	7	6	8	0

Table-6.5

CLOSENESS RATING MATRIX

0	3	6	1	9	6	2	1
3	0	3	9	6	9	3	3
6	3	0	6	9	1	5	4
1	9	6	0	3	6	7	5
9	6	9	3	0	3	8	9
6	9	1	6	3	0	1	2
2	3	4	5	6	8	0	3
4	5	7	8	2	9	4	0

Table-6.6

The program has main function which calls the generation function, fitness function, crossover function, and mutation function. Generation function generates random sequences. We have checked the effectiveness of the program by generating five random sequences at first and then by generating ten random sequences. Fitness function calculates the total cost/total closeness rating for all the random sequences and two sequences are selected to serve as parent according to the roulette wheel selection policy. Crossover function is then call in the main function to generate two offspring. These two off springs are given as input to the mutation function and thus two mutated strings are returned to the main function. Then these functions are put in a loop of 50 generations and the results were obtained, also we increased the number of generation to 150. The result for six departments and ten population sizes with maximizing total closeness rating as the objective is given below:

THE INITIAL POPULATION

1 3 2 4 5 6
3 6 4 2 1 5
2 1 6 3 4 5
2 5 3 4 6 1
4 2 1 5 3 6
3 4 6 1 2 5
3 5 4 2 6 1
2 6 4 1 3 5
4 2 1 3 6 5
2 5 1 6 3 4

THE GENERATION NUMBER 1

THE NEW SEQUENCE

4 3 5 2 6 1 36
3 5 4 2 6 1 36
2 5 3 4 6 1 33
4 2 1 5 3 6 31
2 6 4 1 3 5 31
3 5 4 6 2 1 30
3 6 4 2 1 5 28
2 5 1 6 3 4 28
3 4 6 1 2 5 27
1 3 2 4 5 6 24

THE GENERATION NUMBER 3

THE NEW SEQUENCE

3 5 4 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36
4 3 2 6 1 5 33
6 1 2 4 3 5 33
2 5 3 4 6 1 33
2 6 4 1 3 5 31
4 2 1 5 3 6 31
2 6 4 1 3 5 31
3 5 4 6 2 1 30

THE GENERATION NUMBER 10

THE NEW SEQUENCE

3 4 2 6 1 5 39
4 3 5 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36
4 3 5 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36
3 5 4 2 6 1 36

THE GENERATION NUMBER 12

THE NEW SEQUENCE

3 4 2 6 1 5 39
4 3 5 2 6 1 36
3 5 4 2 6 1 36
3 5 4 2 6 1 36
4 3 5 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36
4 3 5 2 6 1 36
4 3 5 2 6 1 36
3 5 4 2 6 1 36

THE GENERATION NUMBER 41

THE NEW SEQUENCE

4 2 6 1 5 3 42
3 4 2 6 1 5 39
5 3 4 2 6 1 39
3 4 2 6 1 5 39
5 3 4 2 6 1 39
3 4 2 6 1 5 39
5 3 4 2 6 1 39
3 4 2 6 1 5 39
3 4 2 6 1 5 39
3 4 2 6 1 5 39

THE GENERATION NUMBER 85

THE NEW SEQUENCE

4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
2 4 6 1 5 3 39

THE GENERATION NUMBER 90

THE NEW SEQUENCE

4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42
4 2 6 1 5 3 42

Thus optimal sequence, i.e. 4 2 6 1 5 3, is obtained after 90 generations and the maximum closeness rating is 42. A graph is plotted taking generation number in X axis and Fitness value (Total closeness rating) in Y axis and is compared with the graph obtained for five population size. The number of iteration comes out to be 50 for the latter case.

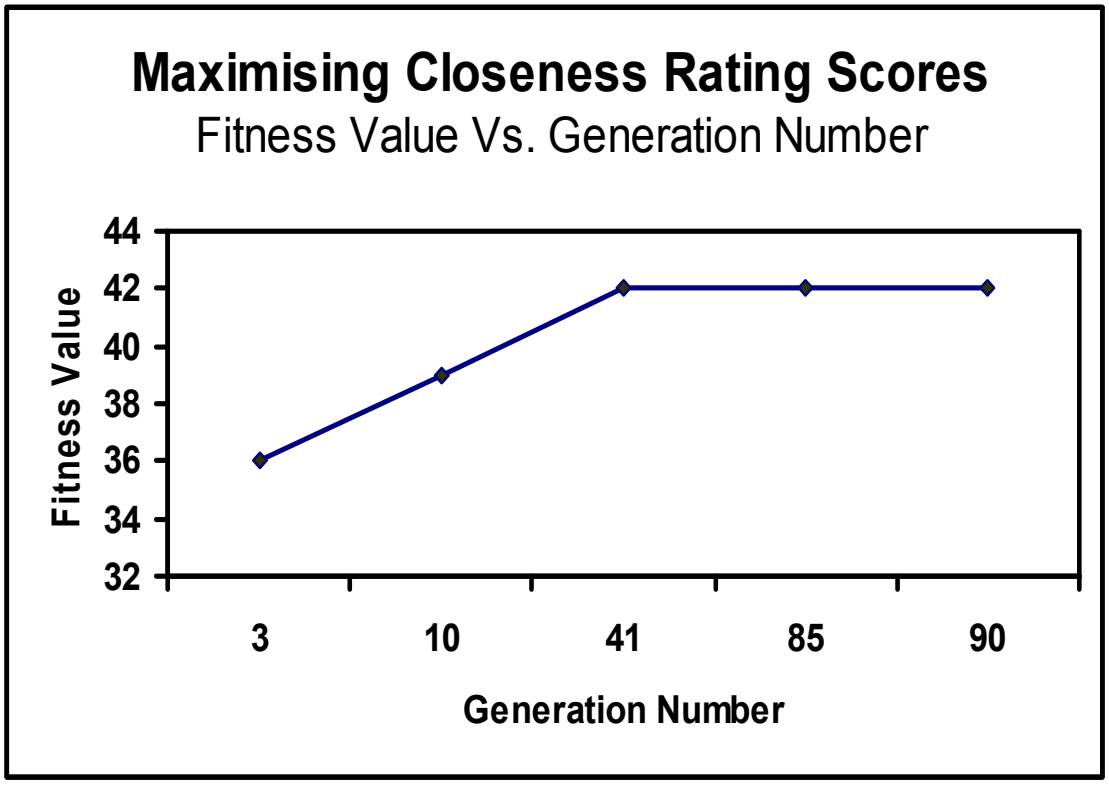


Fig.-6.1

Number of departments = 6 Number of generations = 100

Population size = 10 Maximum closeness rating = 42

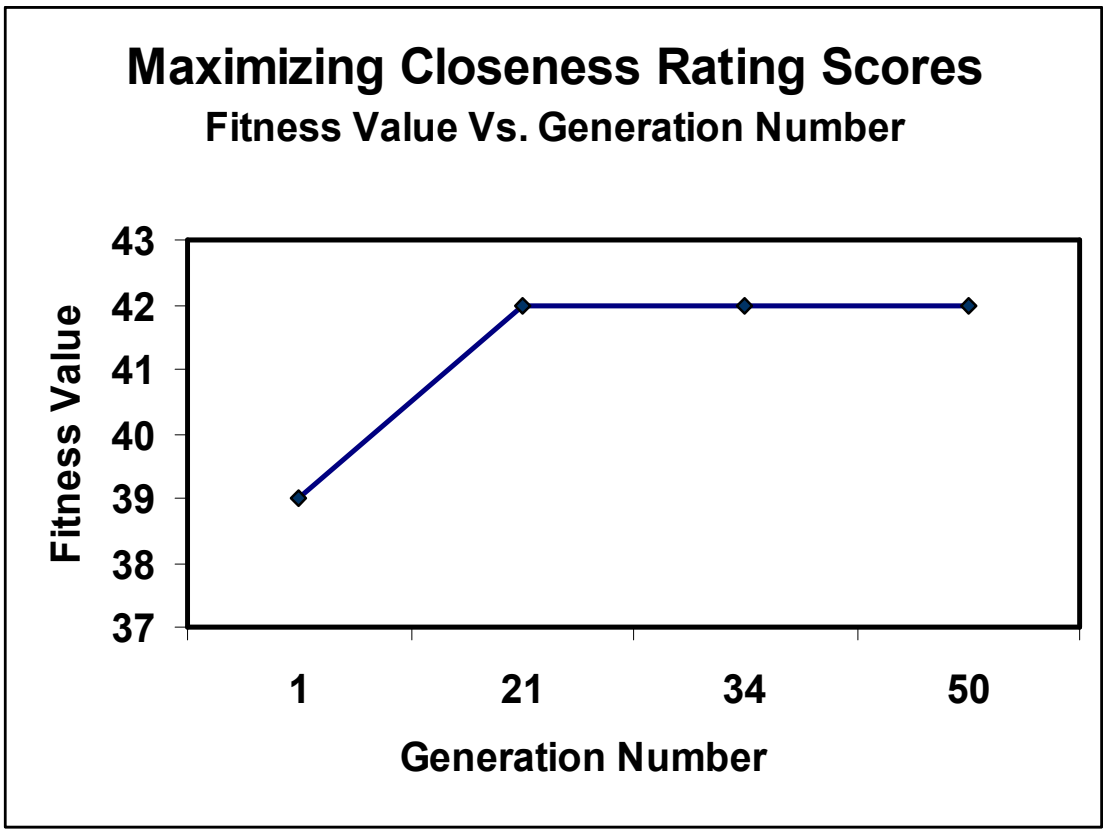


Fig.-6.2

Number of departments = 6 Number of generations = 50

Population size = 5 Maximum closeness rating = 42

The result for six departments and ten population size with minimizing total cost as the objective is given below:

THE INITIAL POPULATION

1 3 2 4 5 6

3 6 4 2 1 5

2 1 6 3 4 5

2 5 3 4 6 1

4 2 1 5 3 6

3 4 6 1 2 5

3 5 4 2 6 1

2 6 4 1 3 5

4 2 1 3 6 5

2 5 1 6 3 4

THE GENERATION NUMBER 1

THE NEW SEQUENCE

2 6 4 1 3 5 9760

3 5 4 2 6 1 8640

4 2 1 5 3 6 8360

2 1 6 3 4 5 8280

2 5 3 4 6 1 8240

2 5 1 6 3 4 8200

1 3 2 4 5 6 8200

3 6 4 2 1 5 8160

3 4 6 1 2 5 8080

4 2 1 3 6 5 8080

THE GENERATION NUMBER 2

THE NEW SEQUENCE

3 5 4 2 6 1 8640

2 6 1 3 5 4 8640

4 2 1 5 3 6 8360

2 1 6 3 4 5 8280

2 5 3 4 6 1 8240

2 5 1 6 3 4 8200

1 3 2 4 5 6 8200

3 6 4 2 1 5 8160

3 4 6 1 2 5 8080

4 2 1 3 6 5 8080

THE GENERATION NUMBER 3

THE NEW SEQUENCE

4 2 1 5 3 6 8360

2 1 6 3 4 5 8280

2 5 3 4 6 1 8240

1 3 2 4 5 6 8200

2 5 1 6 3 4 8200

1 3 2 4 5 6 8200

3 6 4 2 1 5 8160

3 4 6 1 2 5 8080

4 2 1 3 6 5 8080

3 2 1 4 5 6 8000

THE GENERATION NUMBER 13

THE NEW SEQUENCE

1 6 3 4 2 5 8200

3 6 4 2 1 5 8160

3 6 4 1 5 2 8160

5 2 1 6 3 4 8120

2 3 1 4 6 5 8080

4 2 1 3 6 5 8080

3 4 6 1 2 5 8080

2 5 1 6 4 3 8080

3 2 1 4 5 6 8000

2 4 1 3 6 5 7960

THE GENERATION NUMBER 14

THE NEW SEQUENCE

1 5 2 4 6 3 8160

3 6 4 1 5 2 8160

3 6 4 2 1 5 8160

5 2 1 6 3 4 8120

2 3 1 4 6 5 8080

4 2 1 3 6 5 8080

3 4 6 1 2 5 8080

2 5 1 6 4 3 8080

3 2 1 4 5 6 8000

2 4 1 3 6 5 7960

THE GENERATION NUMBER 53

THE NEW SEQUENCE

3 2 1 4 5 6 8000

3 2 1 4 5 6 8000

2 4 1 6 3 5 8000

3 2 1 4 5 6 8000

3 1 2 4 5 6 8000

3 1 2 4 5 6 8000

3 2 1 4 5 6 8000

2 4 1 3 6 5 7960

2 4 5 3 1 6 7960

6 3 1 2 4 5 7760

THE GENERATION NUMBER 67

THE NEW SEQUENCE

3 2 1 4 5 6 8000

3 2 1 4 5 6 8000

3 1 2 4 5 6 8000

2 4 5 3 1 6 7960

2 4 1 3 6 5 7960

3 5 2 1 4 6 7960

6 1 3 2 4 5 7920

4 2 5 3 1 6 7840

6 3 1 2 4 5 7760

6 3 1 2 5 4 7680

THE GENERATION NUMBER 79

THE NEW SEQUENCE

3 2 1 4 5 6 8000

2 4 5 3 1 6 7960

2 4 1 3 6 5 7960

3 5 2 1 4 6 7960

6 1 3 2 4 5 7920

6 1 3 2 4 5 7920

4 2 5 3 1 6 7840

6 3 1 2 4 5 7760

6 3 1 2 5 4 7680

3 6 1 2 4 5 7640

THE GENERATION NUMBER 124

THE NEW SEQUENCE

6 3 1 2 5 4 7680

6 3 1 2 5 4 7680

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

THE GENERATION NUMBER 125

THE NEW SEQUENCE

6 3 1 2 5 4 7680

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

THE GENERATION NUMBER 126

THE NEW SEQUENCE

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

3 6 1 2 4 5 7640

The optimal sequence, i.e.3 6 1 2 4 5, is obtained after 126 generations or iterations and the total cost comes out to be 7640. Then a graph is plotted taking generation number in X axis and Fitness value (Total cost) in Y axis and is compared with the graph obtained for five population size. The number of iterations comes out to be 50.

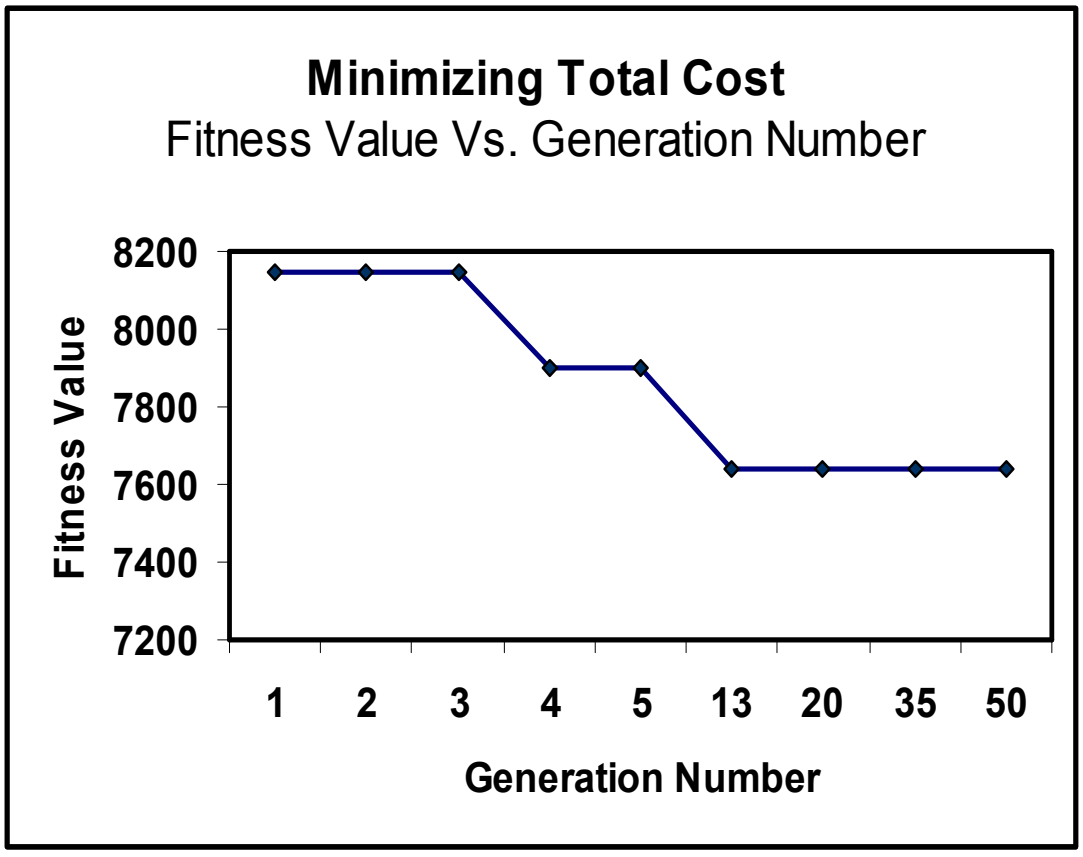


Fig.-6.3

Number of departments=6 Population size = 5

Number of generation = 50 Minimum cost = Rs7640

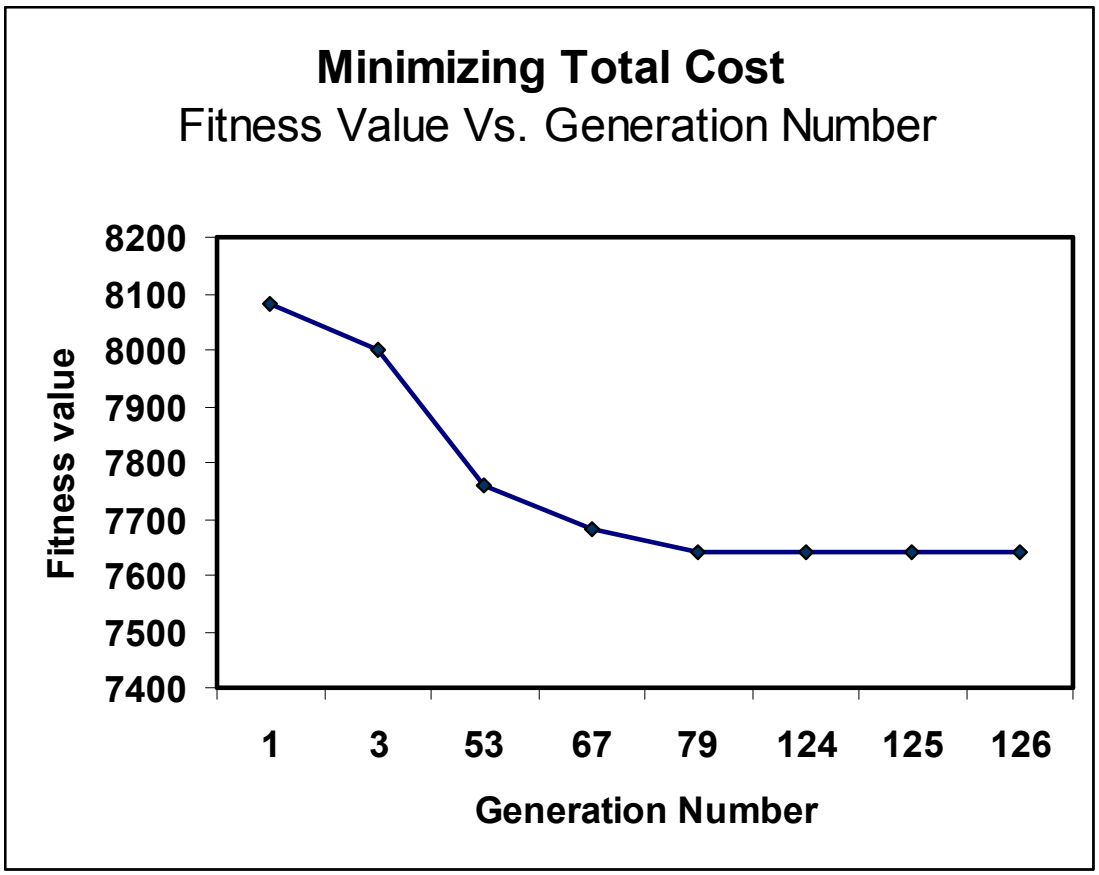


Fig.-6.4

Number of departments = 6 Population size =10

Number of generation =130 Minimum cost = Rs 7640

The same problem for eight departments was also solved for both the objectives and the graphs plotted are given below.

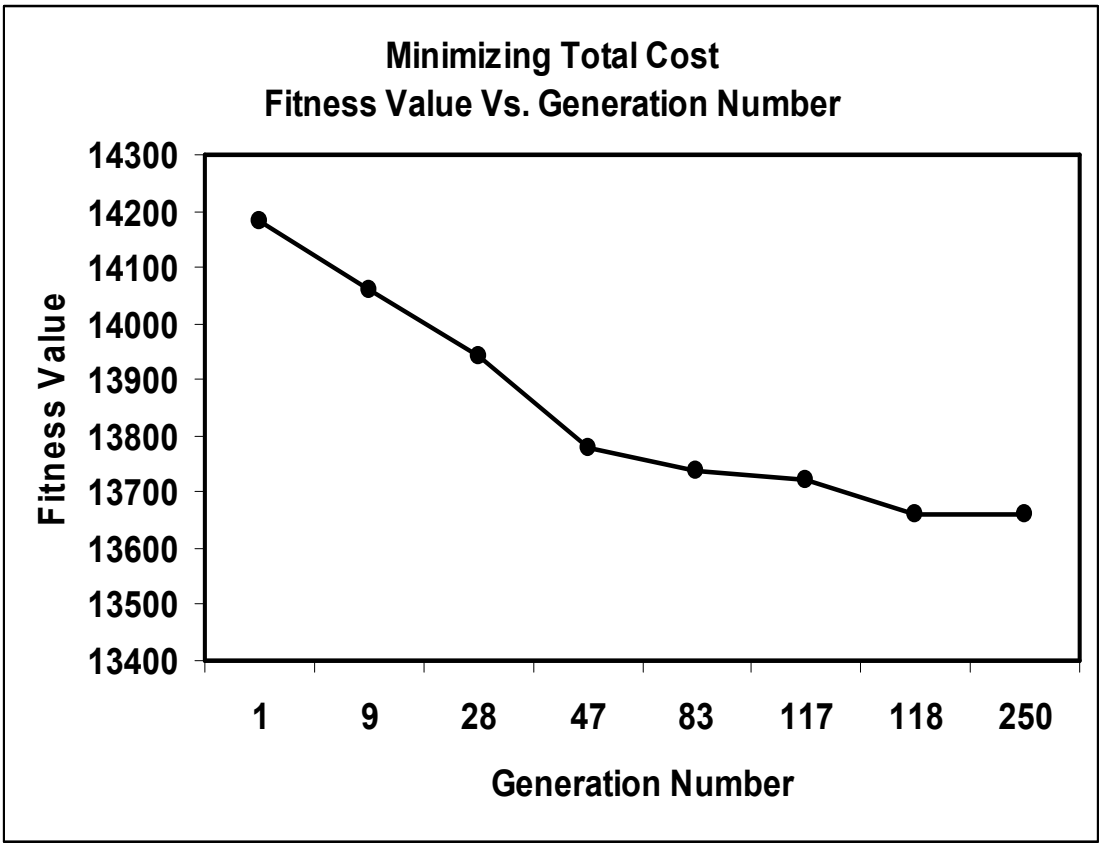


Fig.-6.5

Number of departments = 8 Population size =10

Number of Generations = 250 Minimum cost = Rs 13660

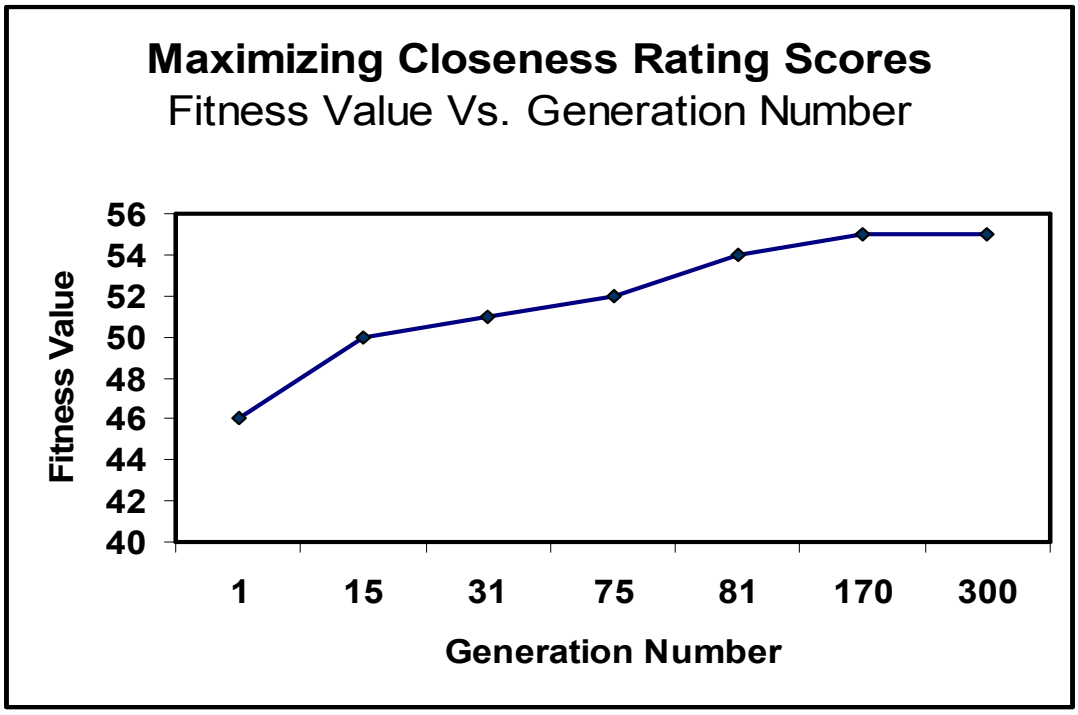


Fig.-6.6

Number of departments = 8 Population size = 10

Number of generations = 300 Maximum Closeness rating = 55

CHAPTER-7

PROGRAMS

Program for maximizing closeness rating

Program for minimizing material handling cost

PROGRAM FOR MAXIMIZING CLOSENESS RATING:

```
/*Program for layout planning using genetic algorithm for maximizing closeness rating*/
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<stdlib.h>
#define pop_size 6
#define data_set 6
#define no_of_generation 100
void ini_generation();
void cross_over();
void mutation();
int parent1[data_set+1],parent2[data_set+1],offspring1[data_set+1],offspring2[data_set+1];
int mutated1[data_set+1],mutated2[data_set+1];
FILE *fin1;
FILE *fout1,*fout2;
  main()
   {
    int i,j,k,seq[data_set+3][data_set+3],temp[data_set+1],temp1[data_set+1];
    int totclose,ran1,ran2,generation;
    float fitness(int temp[]);
    clrscr();

    /*INITIALIZATION*/
    ini_generation();
    fin1=fopen("result2.out","r");
    for(i=1;i<=pop_size;i++){
     for(j=1;j<=data_set;j++){
     fscanf(fin1,"%d",&seq[i][j]);}
     fscanf(fin1,"\n");}

    fout2=fopen("inipop1.out","w");
    fprintf(fout2,"THE INITIAL POPULATION\n");
```

```
 for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set;j++){
 fprintf(fout2,"%3d",seq[i][j]);}
 fprintf(fout2,"\n");}


i=1;
do{
for(j=1;j<=data_set;j++)
{
temp[j]=seq[i][j];
//printf("*%3d",temp[j]);
}
//printf("\n");

totclose=fitness(temp);
seq[i][data_set+1]=totclose;
//printf("totclose=%5d\n",totclose);
i++;
} while(i<=pop_size);

/*fprintf(fout2,"\nTHE INITIAL SEQUENCE WITH FITNESS\n");
for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set+1;j++){
 fprintf(fout2,"%5d",seq[i][j]);}
 fprintf(fout2,"\n");}*/

  /*sorting of initial population in descending order*/
      for(i=1;i<=pop_size;i++)
      {
      for(j=1;j<=pop_size-1;j++)
      {
      if(seq[j][data_set+1]<=seq[j+1][data_set+1])
```

```
        {
        for(k=1;k<=data_set+1;k++){
        temp1[k]=seq[j][k];}
        for(k=1;k<=data_set+1;k++){
        seq[j][k]=seq[j+1][k];}
        for(k=1;k<=data_set+1;k++){
        seq[j+1][k]=temp1[k];}
        }
        else1
        continue;
        }
        }


/* fprintf(fout2,"\nTHE SORTED MATRIX\n");
for(i=1;i<=pop_size;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}*/
  generation=1;

  do{
  fprintf(fout2,"\nTHE GENERATION NUMBER %3d\n",generation);
  ran1=(int)(1.0+pop_size*(float)rand()/RAND_MAX);
  ran2=(int)(1.0+pop_size*(float)rand()/RAND_MAX);
  //fprintf(fout2,"\n%3d%3d\n",ran1,ran2);

  for(j=1;j<=data_set;j++)
  {
  parent1[j]=seq[ran1][j];
  parent2[j]=seq[ran2][j];
  }
  cross_over();
```

```
/* fprintf(fout2,"\nTHE CROSSOVER\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",offspring1[j]);
 fprintf(fout2,"\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",offspring2[j]);
 fprintf(fout2,"\n");*/

 mutation();
 /*fprintf(fout2,"\nTHE MUTATION\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",mutated1[j]);
 fprintf(fout2,"\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",mutated2[j]);
 fprintf(fout2,"\n");*/

 for(j=1;j<=data_set;j++)
 seq[pop_size+1][j]=mutated1[j];
 for(j=1;j<=data_set;j++)
 seq[pop_size+2][j]=mutated2[j];

for(i=pop_size+1;i<=pop_size+2;i++)
{
for(j=1;j<=data_set;j++)
{
temp[j]=seq[i][j];
}
totclose=fitness(temp);
seq[i][data_set+1]=totclose;
}

/*fprintf(fout2,"\nTHE REVISED SEQUENCE WITH FITNESS\n");
```

```
for(i=1;i<=pop_size+2;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}*/

       /*sorting of revised matrix*/
       for(i=1;i<=pop_size+2;i++)
       {
       for(j=1;j<=pop_size+2-1;j++)
       {
       if(seq[j][data_set+1]<=seq[j+1][data_set+1])
       {
       for(k=1;k<=data_set+1;k++){
       temp1[k]=seq[j][k];}
       for(k=1;k<=data_set+1;k++){
       seq[j][k]=seq[j+1][k];}
       for(k=1;k<=data_set+1;k++){
       seq[j+1][k]=temp1[k];}
       }
       else
       continue;
       }
       }

 /*fprintf(fout2,"\nTHE SORTED MATRIX OF REVISED SEQUENCE\n");
for(i=1;i<=pop_size+2;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}*/

 for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set+1;j++){
 seq[i][j]=seq[i][j];}}
```

```
  fprintf(fout2,"\nTHE NEW SEQUENCE\n");
for(i=1;i<=pop_size;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}

generation++;

} while(generation<=no_of_generation);

fclose(fin1);
fclose(fout2);
 return 0;
 }

 void ini_generation()
 {
 int i,j,rand_num,count,k;
 int sequence[data_set+1][data_set+1];
 fout1=fopen("result2.out","w");
 for(i=0;i<data_set;i++){
  for(j=0;j<data_set;j++)
  sequence[i][j]=0;}

  k=1;
  do{
  sequence[k][1]=(int)(1.0+data_set*(float)rand()/RAND_MAX);
  j=1;
  do{
  rand_num=(int)(1.0+data_set*(float)rand()/RAND_MAX);
  count=0;
  for(i=1;i<=j;i++){
  if(sequence[k][i]-rand_num==0)
```

```
count=count+1;}
if(count<1)
{
j++;
sequence[k][j]=rand_num;
}
} while(j<=data_set);
k++;
} while(k<=pop_size);

for(k=1;k<=pop_size;k++){
for(j=1;j<=data_set;j++){
fprintf(fout1,"%3d",sequence[k][j]);}
fprintf(fout1,"\n");}
fclose(fout1);
}

      float fitness(int temp[])
      {
      int i,j,k,sequence[data_set+1];;
      int closerat[data_set+1][data_set+1];
      int sum;
      FILE *fp1;

      fp1=fopen("loutfit.txt","r");

       for(i=1;i<=data_set;i++)
        sequence[i]=temp[i];

/*for(j=1;j<=data_set;j++)
{
printf("**%3d",sequence[j]);
}
```

```
    printf("\n");*/

          for(i=1;i<=data_set;i++){
           for(j=1;j<=data_set;j++){
             fscanf(fp1,"%d",&closerat[i][j]);}
              fscanf(fp1,"\n");}
             sum=0.0;
             k=1;
             do{
             sum=sum+closerat[sequence[k]][sequence[k+1]];
             k++;
             } while(k<data_set);
  fclose(fp1);
  return sum;
  }


/*float fitness(int temp[])
{
static int i,j,x[data_set+1],d[data_set+1][data_set+1],c[data_set+1][data_set+1],TC;
FILE *fin1,*fin2;
fin1=fopen("input1.in","r");

for(i=1;i<=data_set;i++)
 x[i]=temp[i];

for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 fscanf(fin1,"%d",&d[x[i]][x[j]]);
 }
 fscanf(fin1,"\n");
```

```
}

fin2=fopen("input2.in","r");
for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 fscanf(fin2,"%d",&c[i][j]);
 }
 fscanf(fin2,"\n");
}

TC=0;
for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 TC=TC+(d[i][j]*c[i][j]);
 }
}
fclose(fin1);
fclose(fin2);
return TC;
}*/

    void cross_over()
    {
    int fixposno,temp1[data_set+1],temp2[data_set+1],i,fixpos[20],jj,count1,rand_nub;
    int k,par2_inherit1[data_set+1],par1_inherit2[data_set+1];

    /*for(i=1;i<=data_set;i++)
        printf("%5d",parent1[i]);
    printf("\n");
```

```
for(i=1;i<=data_set;i++)
     printf("%5d",parent2[i]);*/

/*for offspring1*/
fixposno=(int)(1.0+data_set*(float)rand()/RAND_MAX);

for(i=1;i<=data_set;i++){
offspring1[i]=0;
temp1[i]=0;}

for(i=1;i<=data_set;i++){
temp1[i]=parent2[i];}


jj=1;
fixpos[jj]=(int)(1.0+data_set*(float)rand()/RAND_MAX);

do{
rand_nub=(int)(1.0+data_set*(float)rand()/RAND_MAX);
count1=0;
for(i=1;i<=jj;i++){
if(fixpos[i]-rand_nub==0)
count1=count1+1;}
if(count1<1)
{
jj++;
fixpos[jj]=rand_nub;
}
} while(jj<=fixposno);

for(i=1;i<=fixposno;i++){
offspring1[fixpos[i]]=parent1[fixpos[i]];}
```

```
k=1;
do{
if(offspring1[k]>0)
{
for(i=1;i<=data_set;i++){
if(temp1[i]-offspring1[k]==0)
temp1[i]=0;}
}
k++;
} while(k<=data_set);

k=1;
for(i=1;i<=data_set;i++)
{
if(temp1[i]>0){
par2_inherit1[k]=temp1[i];
k++; }
}

k=1;
for(i=1;i<=data_set;i++)
{
if(offspring1[i]<1){
offspring1[i]=par2_inherit1[k];
k++;}
}

/*for offspring2*/
fixposno=(int)(1.0+data_set*(float)rand()/RAND_MAX);

for(i=1;i<=data_set;i++){
offspring2[i]=0;
temp2[i]=0;}
```

```
for(i=1;i<=data_set;i++){
temp2[i]=parent1[i];}


jj=1;
fixpos[jj]=(int)(1.0+data_set*(float)rand()/RAND_MAX);

do{
rand_nub=(int)(1.0+data_set*(float)rand()/RAND_MAX);
count1=0;
for(i=1;i<=jj;i++){
if(fixpos[i]-rand_nub==0)
count1=count1+1;}
if(count1<1)
{
jj++;
fixpos[jj]=rand_nub;
}
} while(jj<=fixposno);

for(i=1;i<=fixposno;i++){
offspring2[fixpos[i]]=parent2[fixpos[i]];}


k=1;
do{
if(offspring2[k]>0)
{
for(i=1;i<=data_set;i++){
if(temp2[i]-offspring2[k]==0)
temp2[i]=0;}
}
k++;
```

```
} while(k<=data_set);

k=1;
for(i=1;i<=data_set;i++)
{
if(temp2[i]>0){
par1_inherit2[k]=temp2[i];
k++; }
}

k=1;
for(i=1;i<=data_set;i++)
{
if(offspring2[i]<1){
offspring2[i]=par1_inherit2[k];
k++;}
}
}


void mutation()
{
int temp[data_set+1],temp1[data_set+1];
int tamp[data_set+1],temp2[data_set+1];
int mutpos1,mutpos2;
int i,k;
//srand(5);
/*for mutation1*/
mutpos1=(int)(1.0+data_set*(float)rand()/RAND_MAX);
mutpos2=(int)(1.0+data_set*(float)rand()/RAND_MAX);
for(i=1;i<=data_set;i++){
temp[i]=0;}
temp[mutpos2]=offspring1[mutpos1];
```

```
k=1;
for(i=1;i<=data_set;i++)
{
if(i!=mutpos1){
temp1[k]=offspring1[i];
k++; }
}
k=1;
for(i=1;i<=data_set;i++)
{
if(temp[i]<1){
temp[i]=temp1[k];
k++;}
}
for(i=1;i<=data_set;i++){
mutated1[i]=temp[i];}

/*for mutation2*/
mutpos1=(int)(1.0+data_set*(float)rand()/RAND_MAX);
mutpos2=(int)(1.0+data_set*(float)rand()/RAND_MAX);
for(i=1;i<=data_set;i++){
tamp[i]=0;}
tamp[mutpos2]=offspring2[mutpos1];

k=1;
for(i=1;i<=data_set;i++)
{
if(i!=mutpos1){
temp2[k]=offspring2[i];
k++; }
}

k=1;
```

```
for(i=1;i<=data_set;i++)
{
if(tamp[i]<1){
tamp[i]=temp2[k];
k++;}
}

for(i=1;i<=data_set;i++){
mutated2[i]=tamp[i];}

/*for(i=1;i<=data_set;i++)
printf("%3d",mutated1[i]);
printf("\n");
for(i=1;i<=data_set;i++)
printf("%3d",mutated2[i]);*/

}
```

PROGRAM FOR MINIMIZING MATERIAL HANDLING COST:

```
/*Program for layout planning using genetic algorithm for minimizing total cost*/
#include<stdio.h>
#include<math.h>
#include<conio.h>
#include<stdlib.h>
#define pop_size 6
#define data_set 6
#define no_of_generation 100
void ini_generation();
void cross_over();
void mutation();
int parent1[data_set+1],parent2[data_set+1],offspring1[data_set+1],offspring2[data_set+1];
int mutated1[data_set+1],mutated2[data_set+1];
FILE *fin1;
FILE *fout1,*fout2;
  main()
   {
    int i,j,k,seq[data_set+3][data_set+3],temp[data_set+1],temp1[data_set+1];
    int totclose;
    float fitness(int temp[]),ran1,ran2,generation;
    clrscr();

    /*INITIALIZATION*/
    ini_generation();
    fin1=fopen("result2.out","r");
    for(i=1;i<=pop_size;i++){
     for(j=1;j<=data_set;j++){
     fscanf(fin1,"%d",&seq[i][j]);}
     fscanf(fin1,"\n");}

    fout2=fopen("inipop.out","w");
    fprintf(fout2,"THE INITIAL POPULATION\n");
```

```
 for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set;j++){
 fprintf(fout2,"%3d",seq[i][j]);}
 fprintf(fout2,"\n");}


i=1;
do{
for(j=1;j<=data_set;j++)
{
temp[j]=seq[i][j];
//printf("*%3d",temp[j]);
}
//printf("\n");

totclose=(int)fitness(temp);
seq[i][data_set+1]=totclose;
//printf("totclose=%5d\n",totclose);
i++;
} while(i<=pop_size);

/*fprintf(fout2,"\nTHE INITIAL SEQUENCE WITH FITNESS\n");
for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set+1;j++){
 fprintf(fout2,"%5d",seq[i][j]);}
 fprintf(fout2,"\n");}*/

  /*sorting of initial population in descending order*/
      for(i=1;i<=pop_size;i++)
      {
      for(j=1;j<=pop_size-1;j++)
      {
      if(seq[j][data_set+1]<=seq[j+1][data_set+1])
```

```
        {
        for(k=1;k<=data_set+1;k++){
        temp1[k]=seq[j][k];}
        for(k=1;k<=data_set+1;k++){
        seq[j][k]=seq[j+1][k];}
        for(k=1;k<=data_set+1;k++){
        seq[j+1][k]=temp1[k];}
        }
        else
        continue;
        }
        }


/* fprintf(fout2,"\nTHE SORTED MATRIX\n");
for(i=1;i<=pop_size;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}*/
   generation=1;

   do{
  fprintf(fout2,"\nTHE GENERATION NUMBER %3d\n", generation);
  ran1=(int)(1.0+pop_size*(float)rand()/RAND_MAX);
  ran2=(int)(1.0+pop_size*(float)rand()/RAND_MAX);
  for(j=1;j<=data_set;j++)
  {
  parent1[j]=seq[pop_size-1][j];
  parent2[j]=seq[pop_size][j];
  }
  cross_over();
  fprintf(fout2,"\nTHE CROSSOVER\n");
  for(j=1;j<=data_set;j++)
```

```
 fprintf(fout2,"%5d",offspring1[j]);
 fprintf(fout2,"\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",offspring2[j]);
 fprintf(fout2,"\n");

 mutation();
 fprintf(fout2,"\nTHE MUTATION\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",mutated1[j]);
 fprintf(fout2,"\n");
 for(j=1;j<=data_set;j++)
 fprintf(fout2,"%5d",mutated2[j]);
 fprintf(fout2,"\n");

 for(j=1;j<=data_set;j++)
 seq[pop_size+1][j]=mutated1[j];
 for(j=1;j<=data_set;j++)
 seq[pop_size+2][j]=mutated2[j];

for(i=pop_size+1;i<=pop_size+2;i++)
{
for(j=1;j<=data_set;j++)
{
temp[j]=seq[i][j];
}
totclose=(int)fitness(temp);
seq[i][data_set+1]=totclose;
}

fprintf(fout2,"\nTHE REVISED SEQUENCE WITH FITNESS\n");
for(i=1;i<=pop_size+2;i++){
 for(j=1;j<=data_set+1;j++){
```

```
 fprintf(fout2,"%5d",seq[i][j]);}
 fprintf(fout2,"\n");}

      /*sorting of revised matrix*/
      for(i=1;i<=pop_size+2;i++)
      {
      for(j=1;j<=pop_size+2-1;j++)
      {
      if(seq[j][data_set+1]<=seq[j+1][data_set+1])
      {
      for(k=1;k<=data_set+1;k++){
      temp1[k]=seq[j][k];}
      for(k=1;k<=data_set+1;k++){
      seq[j][k]=seq[j+1][k];}
      for(k=1;k<=data_set+1;k++){
      seq[j+1][k]=temp1[k];}
      }
      else
      continue;
      }
      }


 fprintf(fout2,"\nTHE SORTED MATRIX OF REVISED SEQUENCE\n");
for(i=1;i<=pop_size+2;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");}

 for(i=1;i<=pop_size;i++){
 for(j=1;j<=data_set+1;j++){
 seq[i][j]=seq[i+2][j];}}
```

```
  fprintf(fout2,"\nTHE NEW SEQUENCE\n");
for(i=1;i<=pop_size;i++){
  for(j=1;j<=data_set+1;j++){
  fprintf(fout2,"%5d",seq[i][j]);}
  fprintf(fout2,"\n");
  }
  generation++ ;
  }while(generation<=no_of_generation);

fclose(fin1);
fclose(fout2);
 return 0;
 }

 void ini_generation()
 {
 int i,j,rand_num,count,k;
 int sequence[data_set+1][data_set+1];
 fout1=fopen("result2.out","w");
 for(i=0;i<data_set;i++){
  for(j=0;j<data_set;j++)
  sequence[i][j]=0;}

  k=1;
  do{
  sequence[k][1]=(int)(1.0+data_set*(float)rand()/RAND_MAX);
  j=1;
  do{
  rand_num=(int)(1.0+data_set*(float)rand()/RAND_MAX);
   count=0;
  for(i=1;i<=j;i++){
  if(sequence[k][i]-rand_num==0)
  count=count+1;}
```

```
    if(count<1)
    {
    j++;
    sequence[k][j]=rand_num;
    }
    } while(j<=data_set);
    k++;
    } while(k<=pop_size);

    for(k=1;k<=pop_size;k++){
    for(j=1;j<=data_set;j++){
    fprintf(fout1,"%3d",sequence[k][j]);}
    fprintf(fout1,"\n");}
    fclose(fout1);
    }

float fitness(int temp[])
{
static int i,j,x[data_set+1],d[data_set+1][data_set+1],c[data_set+1][data_set+1],TC;
FILE *fin1,*fin2;
fin1=fopen("input1.txt","r");

for(i=1;i<=data_set;i++)
 x[i]=temp[i];

for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 fscanf(fin1,"%d",&d[x[i]][x[j]]);
 }
 fscanf(fin1,"\n");
}
```

```
fin2=fopen("input2.txt","r");
for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 fscanf(fin2,"%d",&c[i][j]);
 }
 fscanf(fin2,"\n");
}

TC=0;
for(i=1;i<=data_set;i++)
{
 for(j=1;j<=data_set;j++)
 {
 TC=TC+(d[i][j]*c[i][j]);
 }
}
fclose(fin1);
fclose(fin2);
return TC;
}

    void cross_over()
    {
    int fixposno,temp1[data_set+1],temp2[data_set+1],i,fixpos[20],jj,count1,rand_nub;
    int k,par2_inherit1[data_set+1],par1_inherit2[data_set+1];

    /*for(i=1;i<=data_set;i++)
        printf("%5d",parent1[i]);
    printf("\n");
    for(i=1;i<=data_set;i++)
        printf("%5d",parent2[i]);*/
```

```
/*for offspring1*/
fixposno=(int)(1.0+data_set*(float)rand()/RAND_MAX);

for(i=1;i<=data_set;i++){
offspring1[i]=0;
temp1[i]=0;}

for(i=1;i<=data_set;i++){
temp1[i]=parent2[i];}


jj=1;
fixpos[jj]=(int)(1.0+data_set*(float)rand()/RAND_MAX);

do{
rand_nub=(int)(1.0+data_set*(float)rand()/RAND_MAX);
count1=0;
for(i=1;i<=jj;i++){
if(fixpos[i]-rand_nub==0)
count1=count1+1;}
if(count1<1)
{
jj++;
fixpos[jj]=rand_nub;
}
} while(jj<=fixposno);

for(i=1;i<=fixposno;i++){
offspring1[fixpos[i]]=parent1[fixpos[i]];}

k=1;
do{
if(offspring1[k]>0)
```

```
{
for(i=1;i<=data_set;i++){
if(temp1[i]-offspring1[k]==0)
temp1[i]=0;}
}
k++;
} while(k<=data_set);

k=1;
for(i=1;i<=data_set;i++)
{
if(temp1[i]>0){
par2_inherit1[k]=temp1[i];
k++; }
}

k=1;
for(i=1;i<=data_set;i++)
{
if(offspring1[i]<1){
offspring1[i]=par2_inherit1[k];
k++;}
}

/*for offspring2*/
fixposno=(int)(1.0+data_set*(float)rand()/RAND_MAX);

for(i=1;i<=data_set;i++){
offspring2[i]=0;
temp2[i]=0;}

for(i=1;i<=data_set;i++){
temp2[i]=parent1[i];}
```

```
jj=1;
fixpos[jj]=(int)(1.0+data_set*(float)rand()/RAND_MAX);

do{
rand_nub=(int)(1.0+data_set*(float)rand()/RAND_MAX);
count1=0;
for(i=1;i<=jj;i++){
if(fixpos[i]-rand_nub==0)
count1=count1+1;}
if(count1<1)
{
jj++;
fixpos[jj]=rand_nub;
}
} while(jj<=fixposno);

for(i=1;i<=fixposno;i++){
offspring2[fixpos[i]]=parent2[fixpos[i]];}


k=1;
do{
if(offspring2[k]>0)
{
for(i=1;i<=data_set;i++){
if(temp2[i]-offspring2[k]==0)
temp2[i]=0;}
}
k++;
} while(k<=data_set);

k=1;
```

```
for(i=1;i<=data_set;i++)
{
if(temp2[i]>0){
par1_inherit2[k]=temp2[i];
k++; }
}

k=1;
for(i=1;i<=data_set;i++)
{
if(offspring2[i]<1){
offspring2[i]=par1_inherit2[k];
k++;}
}
}

void mutation()
{
int temp[data_set+1],temp1[data_set+1];
int tamp[data_set+1],temp2[data_set+1];
int mutpos1,mutpos2;
int i,k;
//srand(5);
/*for mutation1*/
mutpos1=(int)(1.0+data_set*(float)rand()/RAND_MAX);
mutpos2=(int)(1.0+data_set*(float)rand()/RAND_MAX);
for(i=1;i<=data_set;i++){
temp[i]=0;}
temp[mutpos2]=offspring1[mutpos1];
k=1;
for(i=1;i<=data_set;i++)
{
if(i!=mutpos1){
```

```
temp1[k]=offspring1[i];
k++; }
}
k=1;
for(i=1;i<=data_set;i++)
{
if(temp[i]<1){
temp[i]=temp1[k];
k++;}
}
for(i=1;i<=data_set;i++){
mutated1[i]=temp[i];}

/*for mutation2*/
mutpos1=(int)(1.0+data_set*(float)rand()/RAND_MAX);
mutpos2=(int)(1.0+data_set*(float)rand()/RAND_MAX);
for(i=1;i<=data_set;i++){
tamp[i]=0;}
tamp[mutpos2]=offspring2[mutpos1];
k=1;
for(i=1;i<=data_set;i++)
{
if(i!=mutpos1){
temp2[k]=offspring2[i];
k++; }
}
k=1;
for(i=1;i<=data_set;i++)
{
if(tamp[i]<1){
tamp[i]=temp2[k];
k++;}
}
```

```
for(i=1;i<=data_set;i++){
mutated2[i]=tamp[i];}
for(i=1;i<=data_set;i++)
printf("%3d",mutated1[i]);
printf("\n");
for(i=1;i<=data_set;i++)
printf("%3d",mutated2[i]);

}
```

Chapter -8

FUTURE SCOPE OF THE PROJECT

8.1 Future Scope of the Project

The project can be further improved by adding on features to optimize plant layouts with unequal areas of departments, irregular space between them and cases where some departments are clustered together whereas others are spaced apart. It can also be improved by giving it a graphical user interface (GUI) or by merging it with a database like iSQL server.

REFERENCES:

1. TATE, D.M and SMITH, A.E., 1993.A genetic approach to the quadratic assignment problem, Computational and Operational Research; in Press.

2. KUSIAK, A. and Heragu, S.S; 1987. The facility layout problem, Eur.J.Operational.Research; 29: 229-251.

3. MELLER, R.D and BOZER, Y.A., 1991. Solving the facility layout with simulated annealing, Technical report 91-20, Department of Industrial and Operations Engineering, the University of Michigan.

4. HARHALAKIS, G.Porth, J.M., and XIE, X.L, 1990.Manufacturing cell design using simulated annealing: An industrial application, J. Intelligent Manf .1:185-191.

5. JAJODIA, S., MINIS, I., HARHALAKIS, G. and PROYH, J-M., 1992.CLASS: Computerized layout solution using simulated annealing. Int. J. Prod. Res., 30:95-108.

6. ROSENBLATT, M.T., 1979. The facility layout problem: A multi goal approach. Int . J. Prod. Res., 17:323-331

7. Dutta, K.N, and SAHU, S., 1982. A multi goal heuristic for facilities design problem: MUGHAL, Int. J. Prod. Res., 20:147-154

8. FORTENBERRY, J.C. and COX, J.F., 1985. Multiple criteria approach to the facilities layout problem. Int. J. Prod. Res. 23:773-782

9. WAGHODEKAR, P.H. and SAHU, S., 1986.Facilities layout with multiple objectives. Eng. Costs Prod. Ecnom.,10:104-112

10. URBAN, T.L., 1987. A multiple criteria model for facilities layout problem. Int. J. Prod. Res., 25:1805-1812

11. MALAKOOTI, B.,1989. Multiple objective facility layout: A heuristic to gerate efficient alternatives. Int. J. Prod. Res: 27:1225-1238

12. HOUSHIYAR, A., 1991. Computer aided facility layout problem : An multigoal approach. Comp. and Ind. Engg. 20:177-186