A Project report on

TRANSPORTATION SERVICE SYSTEM FOR GROCERY KART USING GENETIC ALGORITHM

Submitted in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

by

DEEPANK KARTIKEY (14BIT0051)

SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

April 2018

DECLARATION

I hereby declare that the project entitled "**Transportation Service System for Grocery Kart using Genetic Algorithm"** submitted by me, for award of the degree of Bachelor of Technology in Information Technology to Vellore Institute of Technology, Vellore is a record of bonafide work carried out by me under the supervision of Prof. Vanmathi C.

I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Vellore

Date: Signature of the Candidate

CERTIFICATE

This is to certify that the Project Report entitled "**Transportation Service System for Grocery Kart using Genetic Algorithm**" submitted by DEEPANK KARTIKEY (14BIT0051) SITE VIT, Vellore for the award of the degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by him under my supervision.

The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university. The Project report fulfils the requirements and regulations of VIT and in my opinion meets the necessary standards for submission.

Signature of the Guide	**Signature of the HoD**
(Prof. Vanmathi C)	(Dr. Dinakaran M)

Internal Examiner	**External Examiner**

ABSTRACT

Transportation is a major problem domain in logistics, and so it represents a substantial task in the activities of many companies. In some market sectors, transportation means a high percentage of value added to goods. Therefore, the utilization of computerized methods for transportation often results in significant savings ranging from 5% to 20% in the total costs. As per this day there is no deterministic approach invented which will give best result and solves the problem of Vehicle routing with time windows. In the Vehicle Routing Problem (VRP), the aim is to design a set of m minimum cost vehicle routes through n customer locations, so that each route starts and ends at a common location and some side constraints are also satisfied. Many heuristic approaches have been invented to approximate the solution and improve the solution previously achieved by different researchers. I have accepted this challenge which tends to address the delivery of grocery to various stores from warehouses and determine the best paths based on route evaluation, best available routes, determine optimum number of trucks required, profit, cost and efficiency analysis. Various comparative reports will be generated for various user and city data. More specifically, we consider the distance, fixed cost, time, together with risk simultaneously. To get a quicker and more accurate solution, several improvements are proposed in applying GA for optimization search. I am using multi- objective genetic algorithm (GA) to address the problem to obtain optimum solutions. I will be comparing an existing crossover operator and my new crossover operator. Genetic Algorithms are powerful and broadly applicable stochastic search and optimization techniques that work for many problems that are difficult to solve by conventional techniques.

Since multiple objective problems rarely have points that maximize all the objectives simultaneously, I am typically in a situation to maximize each objective to greatest extent possible. I have an interest in multiple objective transportation problems.

ACKNOWLEDGEMENT

I would like to express my gratitude to our **Chancellor, Dr. G. Viswanathan; Vice President, G. V. Selvam; Vice Chancellor, Dr. Anand A. Samuel; Pro Vice Chancellor, Dr. S. Narayanan;** and **Dr. Aswani Kumar Cherukuri, Dean, School of Information Technology and Engineering**, for providing with an environment to work in and for their inspiration during the tenure of the course.

In jubilant mood I express ingeniously my wholehearted thanks to **Prof. Dinakaran M., Associate Professor, Head of Department**, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is my pleasure to express with deep sense of gratitude to **Prof. Vanmathi C, Assistant Professor (Selection Grade)**, School of Information Technology and Engineering, Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavour. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of technology.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. Finally, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Vellore
Date:

Name of the student

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

BCRC	Best Cost Route Crossover
PMX	Partially Mapped Crossover

Chapter 1
INTRODUCTION

Transportation Service System for Grocery kart is a multi-objective optimization problem, which needs a suitable approach to obtain a global optimum solution. So, for this high complexity problem without any known sophisticated solution technique, a genetic approach is well suited. In the optimization process, due to the problem specific encoding, the genetic algorithm enables me to compute and optimize the routing quiet easily as compared to the other deterministic approaches. Genetic algorithm is a search heuristic that mimics the process of natural evolution. Genetic algorithms generate solutions to optimization problems using techniques inspired by natural evolution, such as selection, crossover and mutation.

1.1 Genetic Algorithms are powerful and broadly applicable stochastic search and optimization techniques that work for many problems that are difficult to solve by conventional techniques. A genetic algorithm (GA) is a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" towards an optimal solution.

Some basic terms related to Genetic algorithms:

- **Population** − It is a subset of all the possible (encoded) solutions to the given problem. The population for a GA is analogous to the population for human beings except that instead of human beings, we have Candidate Solutions representing human beings.
- **Chromosomes** − A chromosome is one such solution to the given problem.
- **Gene** − A gene is one element position of a chromosome.
- **Allele** − It is the value a gene takes for a chromosome.
- **Genotype** − Genotype is the population in the computation space. In the computation

space, the solutions are represented in a way which can be easily understood and manipulated using a computing system.

- **Phenotype** − Phenotype is the population in the actual real-world solution space in which solutions are represented in a way they are represented in real world situations.
- **Decoding and Encoding** − For simple problems, the **phenotype and genotype** spaces are the same. However, in most of the cases, the phenotype and genotype spaces are different. Decoding is a process of transforming a solution from the genotype to the phenotype space, while encoding is a process of transforming from the phenotype to genotype space. Decoding should be fast as it is carried out repeatedly in a GA during the fitness value calculation.
- **Fitness Function** − A fitness function simply defined is a function which takes the solution as input and produces the suitability of the solution as the output. In some cases, the fitness function and the objective function may be the same, while in others it might be different based on the problem.
- **Genetic Operators** − These alter the genetic composition of the offspring. These include crossover, mutation, selection, etc.

A Genetic Algorithm is a problem-solving method that uses genetics as its model of problem solving. It is a search and optimization tool to find approximate solutions, which work differently from classical search method. All form of possible solutions corresponding to a given problem constitute the Search Space.

So, the problem consists of finding out the solution that fits the best. But, when the search space becomes large, enumeration is no longer feasible as it will consume a large amount of time. So, a specific technique is used to find out the optimal solution. GA handles a population of possible solutions by applying a set of genetic operators directly on the population.

These operators include: -

Selection (Reproduction) is used to compare everyone in the population. It is done by using a fitness function. The fitness corresponds to an evaluation of how good the candidate solution is. The optimal solution is the one, which maximizes or minimizes the fitness function according to the given problem. In the Fitness Evaluation step, everyone’s quality is assessed. Then the Selection process helps to decide which individuals are to be used for reproduction and mutation to produce new search points.

Various selection methods are: -

a) Roulette Wheel Selection
b) Random Selection
c) Rank Selection
d) Tournament Selection
e) Boltzmann Selection
f) Elitism

Tournament Selection: It is a method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Crossover(Recombination) is the process of taking two parent solutions and producing from them a child (another solution). After the Selection process, the population is enriched with the better individuals. Then, this operator is applied on the mating pool (search space) to produce a better offspring. It proceeds in 3 steps: -

1) Reproduction operator selects at random a pair of two individual strings for mating.
2) A cross site is selected at random along the string length.
3) Finally, the position values are swapped between the two strings.

The various crossover techniques are:

a) Single point crossover
b) Two-point crossover
c) Multi-point crossover
d) Uniform Crossover

Mutation is used to maintain genetic diversity in the population. It introduces new genetic structures in the population by randomly modifying some of its building blocks. It plays the role of recovering the lost genetic materials as well as for randomly distributing the genetic information. Mutation can be generally done by:

a) Flipping
b) Interchanging

The basic Genetic Algorithm is as follows:

Step 1: [start] Selecting Genetic random population of ‘n’ chromosomes
(suitable solutions for the problem)

Step 2: [Fitness] Evaluate the fitness f(x) of each chromosome x in the population

Step 3:

[New population] Create a new population by repeating following steps until the new population is complete

[selection] select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to get selected).

[crossover] With a crossover probability, cross over the parents to form new offspring (children). If no crossover was performed, offspring is the exact copy of parents.

[Mutation] With a mutation probability, mutate new offspring at each locus (position in chromosome)

[Accepting] Place new offspring in the new population.

Step 4: [Replace] Use new generated population for a further sum of the algorithm.

Step 5: [Test] If the end condition is satisfied, stop, and return the best solution in Current population.

Step 6: [Loop] Go to step2 for fitness evaluation.

VRP: In the general Transportation problem, the objective is to minimize the total transportation cost. But this assumption is not always valid. As in the transportation problem there can be multiple objectives such as the fulfilment of transportation schedule contracts, balancing of load among the given number of vehicles, delivering the required quantity in every plant, minimization of transportation hazards and minimizing the cost. Generally, a multi-objective linear program is written as:

Max $z_1(x) = c^1(x)$

Max $z_2(x) = c^2(x)$

Max $z_3(x) = c^3(x)$

.

.

.

Max zq(x) = cq(x)

s.t. $x \varepsilon S$

Where q is the number of objectives, c^i is the vector of the i^{th} objective function coefficients(gradient), S the feasible region, and $z_i(x)$ the i^{th} objective function value (criterion value). Since multiple objective problems rarely have points that maximize all the objectives simultaneously, we try to maximize each objective to the greatest extent possible.

1.2 Applet Basics

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal. An applet is a Java class that extends the java.Applet class. A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment. The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime. The Figure 1.1 explains the life-cycle of an applet with help of flow-diagram.

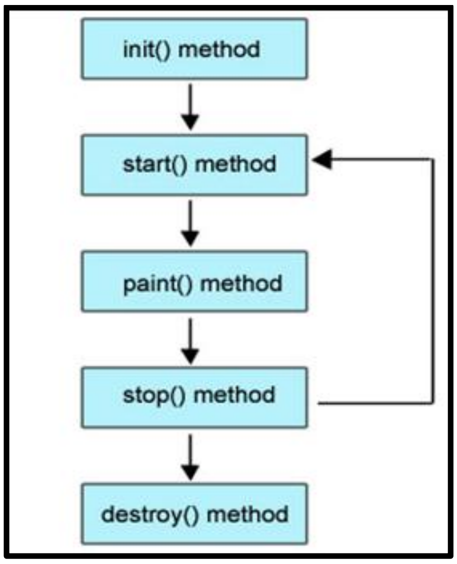


Fig. 1.1 Life Cycle of an Applet

The methods in the Applet class gives you the framework on which you build any serious applet −

- init − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- start − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- stop − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- destroy − This method is only called when the browser shuts down normally. Because

applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.

- paint − Invoked immediately after the start () method, and any time the applet needs to
- repaint itself in the browser. The paint () method is inherited from the java.awt.

These methods are known as Applet Life Cycle methods. These methods are defined in java.applet.Applet class except paint() method. The paint() method is defined in java.awt.Component class, an indirect super-class of Applet class.

1.3 Problem Statement

GroceryKart.com is responsible for supplying fresh vegetables and produce to all retail stores in Mumbai and adjoining suburbs. There are about 600 stores in their network. They have a fleet of old Matador vans that are being used to haul the goods from their central warehouse in Andheri. Given the perishable nature of the goods, all vans leave at 5:00 AM and all deliveries should happen before 8:00 AM. Each van can make multiple stop(s) but the union contract prevents any truck from making more than 20 stops or travelling more than 65 KM in any trip.

In effort to reduce costs and minimize greenhouse emissions, GroceryKart.com is planning to retire all the old Matador vans and purchase a new fleet of Mahindra vans. They believe the cost of purchasing the Mahindra vans will more/less be covered by the sale of Matador vans and the rest of the money will come from the existing savings fund. However, maintaining each truck costs Rs. 20,000 a month (including driver, etc.). Also, the variable cost/km of travel is around Rs. 30 for the first three years of the truck usage. Each Mahindra truck can carry up to 1000 kg at full capacity and at the end of the trip, all trucks return to the warehouse. Each stop at the store takes 5 minutes to download (irrespective of the weight) and every KM travelled takes 90 seconds on an average.

I should determine the number of Mahindra vans required to be purchased and the total KM that will be travelled to service the daily demand of all the stores to minimize the total cost over the first three years?

Also, determine

a) The number of trucks needed
b) Routing for each truck and
c) The total miles travelled each day

1.4 Literature Survey

The project focuses on few existing applications which partially accomplish what I am trying to do through the project “Transportation Service System for Grocery Kart”. The purpose of this project is to reduce the Time complexity and optimize the resources involved in routing techniques consisting of large data sets using Genetic Algorithm.

The problem of transport of goods, commodities and people is as relevant as when it was raised in 1959 by Dantzig and Ramser (1959), where it was considered as a generalization of the traveling salesman problem. The first article in which the phrase “Vehicle routing” appeared is attributed to Golden [1]. Other versions of Vehicle Routing Problem (VRP) appeared in the early 70s. Liebman and Marks [3] presented mathematical models that represent routing problem associated with the collection of wastes; Levin [4] posed the problem of fleet vehicles; Wilson [2] presented the problem of telephone request for transportation service for persons with disabilities and Marks and Stricker presented a model for routing public service vehicles. Some probabilistic concepts were associated with the problem by Golden and Stewart. In Solomon [5] constraints of time windows were included; Sariklis and Powell [6] proposed a problem where the vehicle does not return to the point where the journey begins. Recently, some specific conditions have been added to approach VRP to real-life problems, resulting in several variants which modify constraints or objective function of the basic VRP optimization model.

In its general form the objective of the VRP (Vehicle Routing Problem) is to design a set of minimum cost routes that serves many places, geographically dispersed, and fulfilling specific constraints of the problem. Since its first formulation in 1959, there have been many publications and has expanded its scope. Initial studies were conducted for analysing the distribution management. In the last decade, there have been significant advances in terms of the technical solution to resolve large instances. Another aspect that has gained interest is the inclusion of technological innovations in the VRP. These include global positioning systems, radio frequency identification and use of high-capacity computer information processing.

The Vehicle routing problem can be described as follows: given a fleet of vehicles with uniform capacity, a common depot, and several customer demands, finds the set of routes with overall minimum route cost which service all the demands [7]. All the itineraries start and end at the depot and they must be designed in such a way that each customer is served only once and just by one vehicle.

Genetic algorithms have been inspired by the natural selection mechanism introduced by

Darwin [8]. They apply certain operators to a population os solutions of the problem at hand, in such a way that the new population is improved compared with the previous one according to a prespecified criterion function. This procedure is applied for a preselected number of iterations and the output of the algorithm is the best solution found in the last population or, in some cases, the best solution found during the evolution of the algorithm. In general, the solutions of the problem at hand are coded and the operators are applied to the coded versions of the solutions. The way the solutions are coded plays an important role in the performance of a genetic algorithm. Inappropriate coding may lead to poor performance.

The operators used by genetic algorithms simulate the way natural selection is carried out. The most well-known operators used are the reproduction, crossover, and mutation operators applied in that order to the current population. The reproduction operator ensures that, in probability, the better a solution in the current population is, the more (less) replicates it has in the next population. The crossover operator, which is applied to the temporary population produced after the application of the reproduction operator, selects pairs of solutions randomly, splits them at a random position, and exchanges their second parts. Finally, the mutation operator, which is applied after the application of the reproduction and crossover operators, selects randomly an element of a solution and alters it with some probability. Hence genetic algorithms provide a search technique used in computing to find true or approximate solutions to optimization and search problems. The Figure 1.2 tells about various solution techniques for vehicle routing problems.

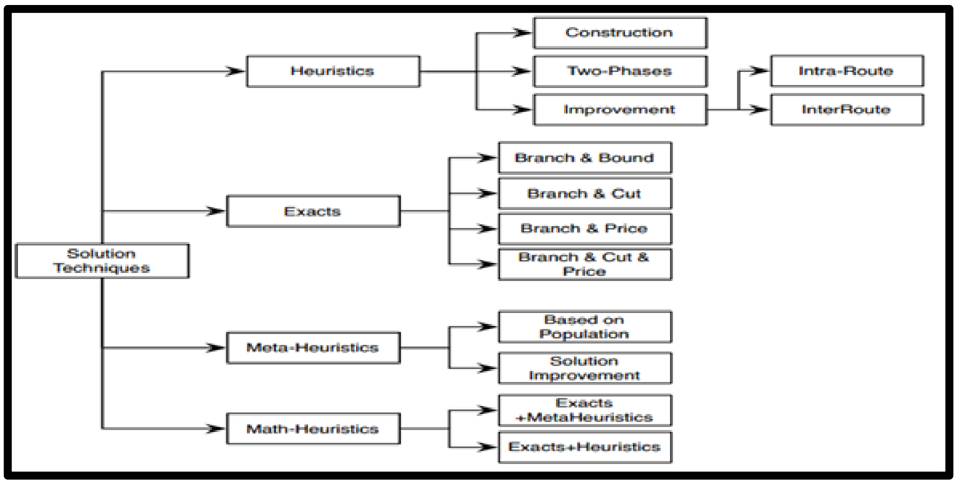


Figure 1.2 VRP Solution Techniques

The solutions techniques for vehicle routing problems can be depicted as follows:

1.4.1 Exacts

The VRP in its different forms, was initially addressed using exact techniques such as Branch and Bound algorithm proposed by Laporte and Nobert [9]; Fischetti, Toth and Vigo [10] and Baldacci and Branch and Cut and Price proposed by Baldacci et al. [11].

1.4.2 Heuristics

Heuristics for the VRP are divided into three classes: two phases,
construction and iterative improvement.
The **two-phase heuristics** divide the problem into two stages:
Customer allocation to route and determination the order of the visit. The iterative improvement receives an initial solution and then another heuristic by inter and intra route movements between customers to explore the solution space. Inside this classification Beasley [12] proposed the strategy Cluster First-Route second and Routed First-Cluster Second.
Construction heuristic, starts from an empty solution which is filled considering feasible alternatives. This kind of heuristics includes savings algorithm in two versions parallel and sequential posed by Clar and Wright [13]. Insertion heuristics, the solution is created by inserting customers who have not been assigned to a route proposed by Mole and Jameson [14]; and Laporte [9].
Iterative improvement heuristics, It is for the betterment of the individual routes, and use concepts such as Lambda exchange, Or - OPT, Van-Breedam Operators, Breedam, GENI and GENIUS, widespread insertions proposed by Gendreau, Hertz and Laporte [9], cyclic transfers, Cross-exchange neighbourhood, which is the generalization three neighbourhoods: insert, swap and Two - OPT. A detailed explanation of these techniques is found.

1.4.3 Metaheuristics

Metaheuristics have been adapted and intensively used for solving different variants of VRP. The main types of metaheuristics which have been applied to solve to the VRP are: Simulated Annealing, Determinist Annealing, Genetic Algorithms, Ant colony optimization, Iterative Location Search and Variable neighbourhood Search To apply such strategies is necessary to define a sufficiently clear and robust coding that allows representing an alternative solution where the value of the objective function and its feasibility is efficiently calculated for all the configurations.
It is also necessary to identify heuristics that can improve the performance of the technique

chosen. It is important to define strategies that deliver quality settings and local search mechanisms that allow for efficient and effective exploration of the solution space. Only the metaheuristics applied to VRP deserve space on multiple items, because many authors not mentioned in this document have made quite creative proposals to improve this methodology type where the performance is evident in the responses and times computer. So, the study of the variation of VRP interest requires special attention when a metaheuristic strategy is defined as an alternative solution.

1.4.4 Math-heuristics

This kind of methods consists in hybrid strategies that combine elements of heuristic techniques, metaheuristic techniques and methodologies exact solution as BB, BC and BCP. Here the sub-problems are identified and modelled in exact way then are solved by commercial integer linear programming solver. Dynamic programming or Lagrangean relaxation sometimes is used for this step.

To obtain best results and overcome disadvantages of other classical search methods, one needs to combine Genetic Algorithms with other optimization techniques.

Chapter 2
System Requirement Specification

The content and qualities of a good Systems Requirements Specification (SRS) are described and several sample SRS outlines are presented. This recommended practice is aimed at specifying requirements of application to be developed.

2.1 System Constraints

Following are the Hardware Constraints for the system:

1) CLIENT: Manufacturing companies, wall mart, food manufacturers, website users.
2) MACHINES: Processor- Pentium 4, RAM 1GB.

Following are the Software Constraints for the system:

1) CLIENT: The client should have at least, JVM.
2) DEVELOPER: JDK 1.8

According to the Design Standard Compliance the system shall be implemented in Java.

2.2 Software Environment

The Environment required to run the software consists of an OS, itextpdf library and JVM at client -side. The OS can be either Windows or Linux. The library itextpdf is required to generate PDF file for the solutions. JVM is required to run the Java application at client system.

2.3 System events and Data flow

a) System Activities

System will take data sets of the distance and the requirement of goods from each store. It will then compute the routing map based on various factors: -

a) Time
b) Distance to be travelled
c) Stores to be visited
d) Load capacity

- Profit, Cost and Efficiency analysis will be done to get optimal solutions.
- Generation of reports through comparative study of various cities and user data.
- Input Data Processing i.e. storing the data and processing it to provide alternative and optimized results.

- Comparative study of best solutions from future generations.

b) User Characteristics

The User should be technologically sound and be able to analyse the results.

2.4 Specific Requirements

a) Functional Objectives

1. System Use Case:

A user and an admin are required for working of proposed system. The user will provide all the necessary details like login information, demands details of each store, distance between stores etc. The admin will provide details after login is authenticated correctly. The admin will also provide a comparative study report generated by the system in form of .pdf document.

Figure 2.1 explains the Use-Case diagram for the implemented system. The User will provide the login details, the data-sets i.e. demand table and distance matrix. The admin will authenticate the login details and allows user to proceed. The system provides various comparative study report for input data.

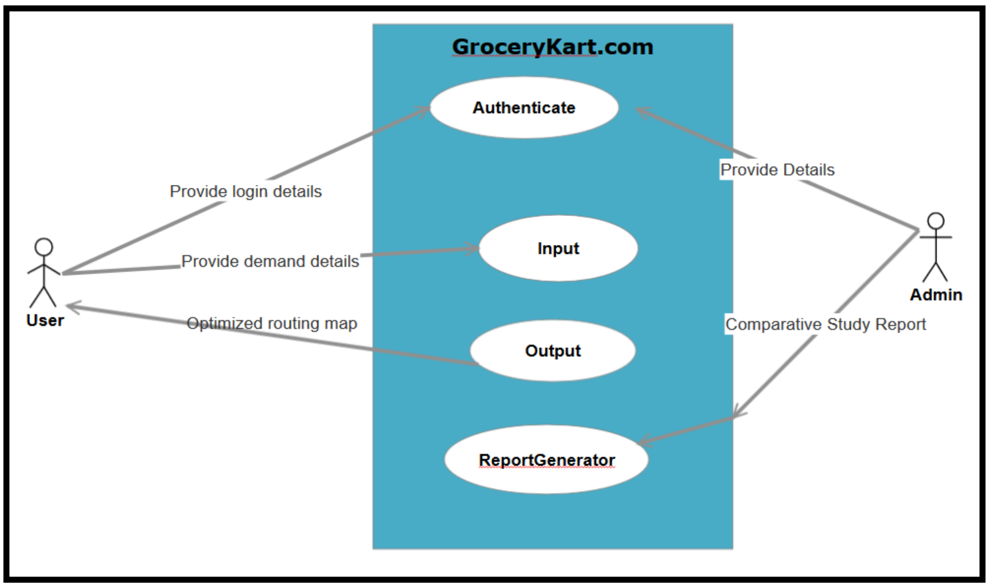


Fig 2.1 Use-Case Diagram

2. Report Generation:

Profit Analysis: Cost incurred by Best Solution-Cost incurred by Past solutions.

Cost Analysis:

1. No. Of Trucks
2. No. Of Km travelled Per day

3. Maintenance charges * (No. Of Trucks) *(years)
4. No. Ok Km travelled Per day * (cost per KM)
5. Efficiency of algorithm: Derived Solution cost / previous solution

Chapter 3
Software Design Specification

The purpose of this document is to outline the technical design of the Transportation Service System for Grocery Kart and provide an overview for the implementation of the model. Its main purpose is to –

- Provide the link between the Functional Specification and the detailed Technical Design documents
- Detail the functionality which will be provided by each component or group of components and show how the various components interact in the design
- Provide a basis for the Grocery Kart system's detailed design and development

As is true with any high-level design, this document will be updated and refined based on changing requirements.

3.1 Description of Problem

GroceryKart.com is responsible for supplying fresh vegetables and produce to all mom-and-pop retail stores in Mumbai and adjoining suburbs. There are about 600 stores in their network. They have a fleet of old Matador vans that are being used to haul the goods from their central warehouse in Andheri. Given the perishable nature of the goods, all vans leave at 5:00 AM and all deliveries must happen before 8:00 AM. Each van can make multiple stops, but the union contract prevents any truck from making more than 20 stops or travelling more than 65 KM in any trip.

To reduce costs and minimize greenhouse emissions, GroceryKart.com is planning to retire all the old Matador vans and purchase a new fleet of Mahindra vans. They believe the cost of purchasing the Mahindra vans will be covered by the sale of Matador vans and the rest of the money will come from the existing savings fund. However, maintaining each truck costs Rs. 20,000 a month (including driver, etc.). Also, the variable cost/km of travel is around Rs. 30 for the first three years of the truck usage. Each Mahindra truck can carry up to 1000 kg at full capacity and at the end of the trip, all trucks return to the warehouse. Each stop at the store takes 5 minutes to download (irrespective of the weight) and every KM travelled takes 90 seconds on an average.

The distance between each store and the warehouse, the daily demand for each store is provided.

We must determine the number of Mahindra vans required to be purchased and the total

KM that will be travelled to service the daily demand of all the stores to minimize the total cost over the first three years?

The output of the code should clearly mention:

a) The number of trucks needed
b) Routing for each truck
c) The total miles travelled each day

3.2 System Architecture

The figure 3.1 explains the implemented system using a block diagram with all the various modules like User Authentication, Route Evaluation, Grocery Kart Form, Report Generation, Truck Analysis, Store Analysis, Best Rotes etc.

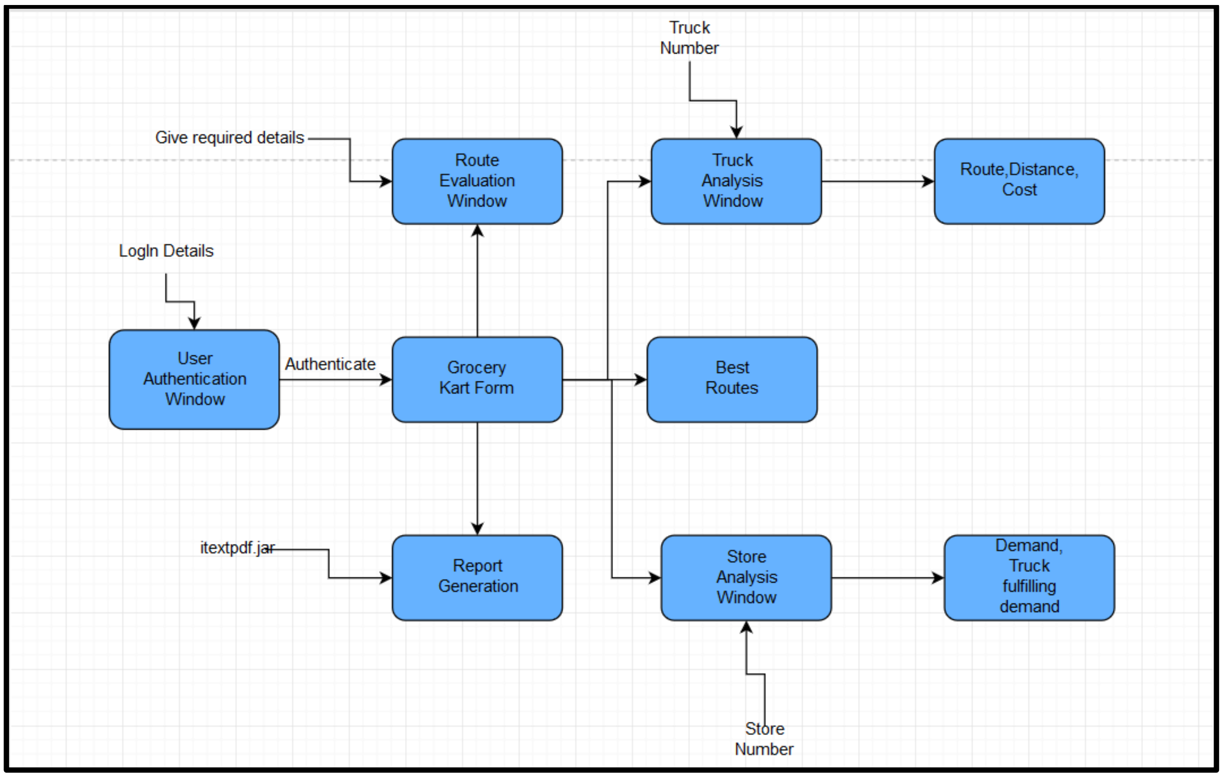


Figure 3.1 Architecture Block Diagram

Route Evaluation window: This window will provide various candidate generation with the total distance travelled and maximum number of trucks required the sorts the result according to the ranks.

The required inputs from the user are Population size, number of generations, Crossover Probability, Mutation Probability, Selection Percent, Type of Crossover to be used, number of solutions to be shown and two datasets: Demand table and Distance Matrix.

Truck Analysis Window: This window will provide the path of the specified truck (i.e.

which all stores it visits) and the distance travelled by that truck and the cost incurred. The required input is truck number for which analysis is needed to be done.

Store Analysis Window: This window will provide the demand of the specified store and which truck fulfils it. The input parameter store number is required.

Best Routes Window: This window will provide the best available routes for the distribution of grocery from the warehouse to various stores.

Report Generation Window: This window will provide 10 best solutions based on

a) Maintenance cost
b) Yearly cost
c) No. of trucks to buy
d) Total distance travelled

3.3 System Operation

A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

The figure 3.2 shows the overall information flow for the system. The input module takes various inputs like population size, crossover probability, mutation probability etc. along with two data-sets i.e Store Demand table and Distance Matrix. The processing module uses two crossovers to process the input data and hence generate optimum routes and minimum travelling costs. They are BCRC (Best Cost Routing Crossover) and PMX (Partially Mapped Crossover). The output module displays various possible routes to various stores with different combinations of trucks. It also generates a pdf of 10 best solutions with help of itextpdf library.

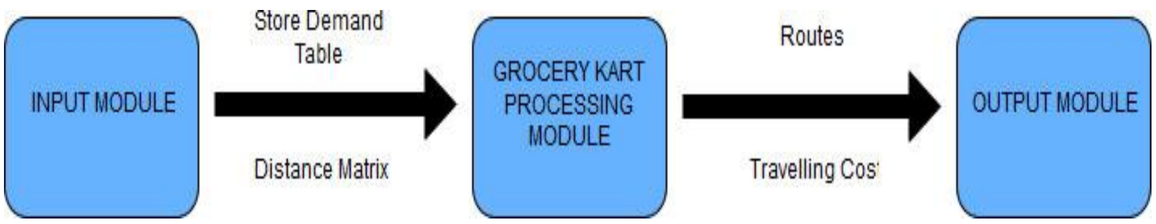


Figure 3.2 Data Flow Diagram

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent. Activity diagrams deal with all type of flow control by using different elements such as fork, join, etc. Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. The figure 3.3 shows an activity diagram for the implemented method.

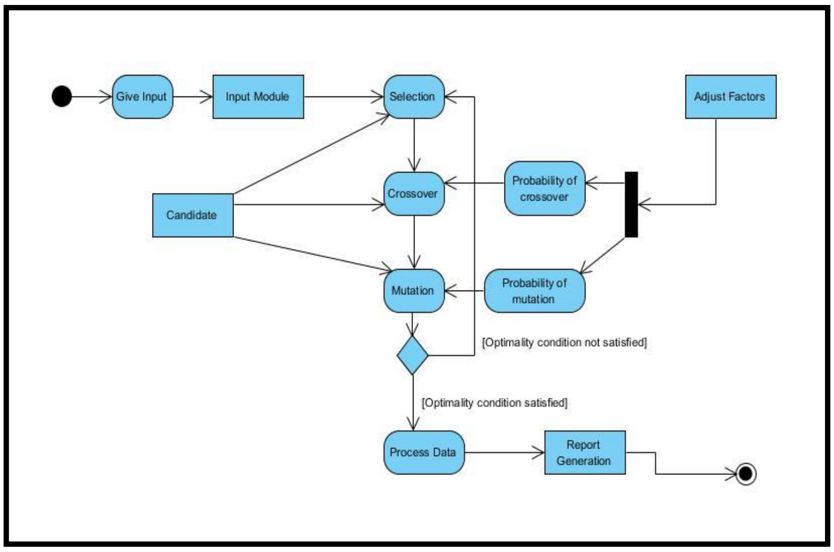


Figure 3.3 Activity Diagram

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. The figure 3.4 shows a sequence diagram for Admin Interaction with the system.

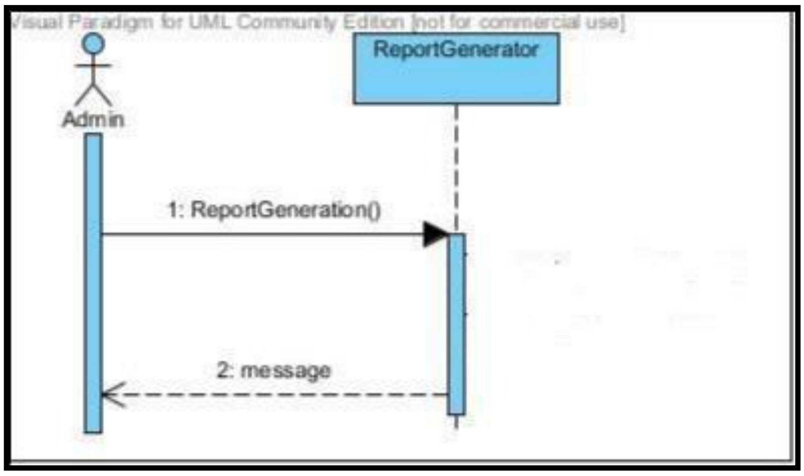

Visual Paradigm for UML Community Edition [not for commercial use]
Admin
ReportGenerator
1: ReportGeneration()
2: message

Figure 3.4 Sequence Diagram

I have chosen Object Oriented Designing for the Transportation Service System for Grocery Kart. **Object-oriented design** is the process of planning a system of interacting objects for solving a software problem. It is one approach to software design. An object contains encapsulated data and procedures grouped together to represent an entity.
The 'object interface', how the object can be interacted with, is also defined. An object-oriented program is described by the interaction of these objects. Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.
We have chosen this architecture because of the following reasons:

a) Easier maintenance.
b) Objects may be understood as stand-alone entities.
c) Objects are potentially reusable components.
d) For some systems, there may be an obvious mapping from real world entities to system objects

A **class diagram** in the UML is a type of static structure diagram that describes the structure of a system by showing the system's classes their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modelling. It is used for general concept-modelling of the systematic of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. Since, Object-oriented Programming is being used here, thus the functionality of the system is divided into various classes. In the diagram, classes are represented with boxes that contain three compartments: The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized. The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase. The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase. The figure 3.5 shows the class diagram of the implemented system.

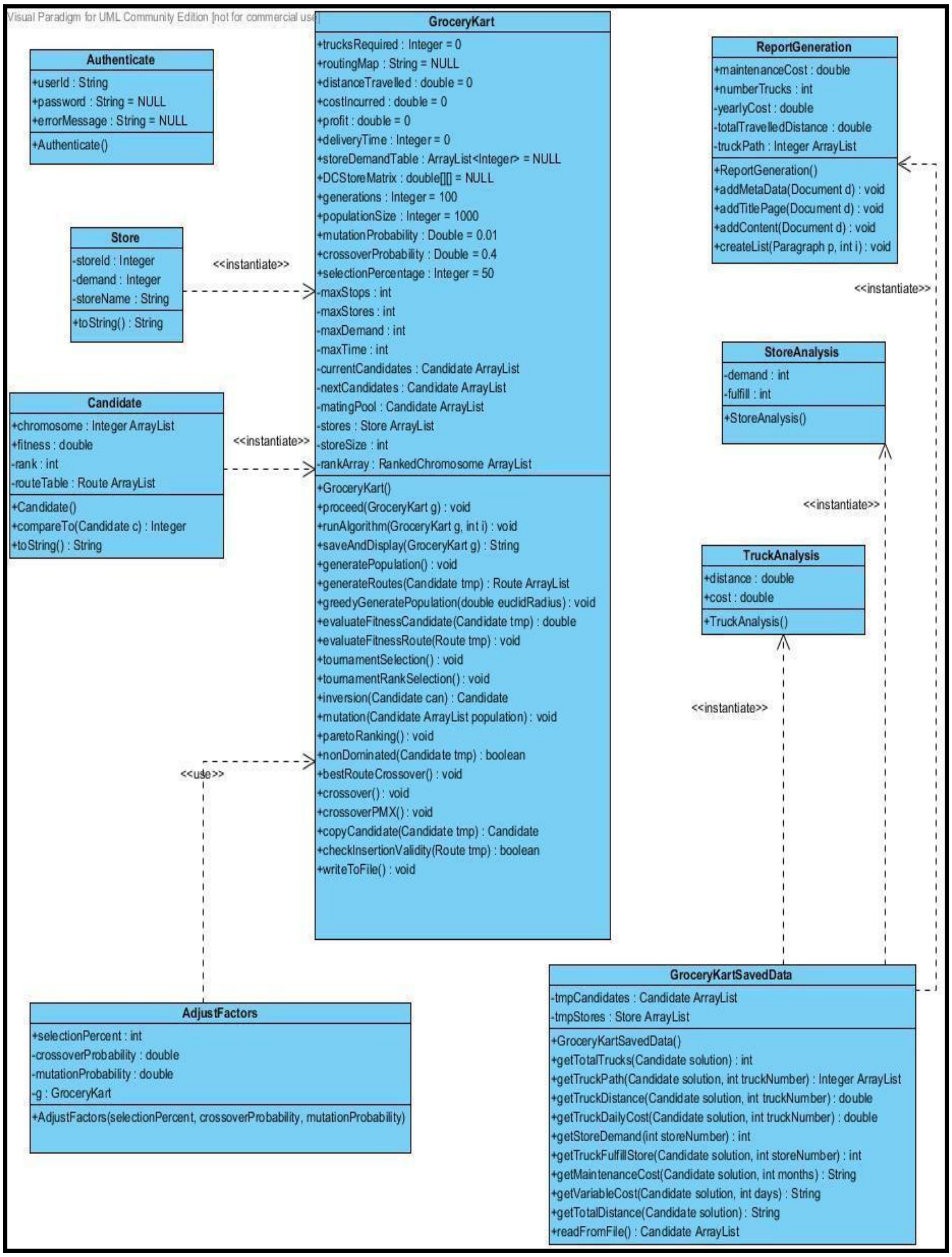

Visual Paradigm for UML Community Edition [not for commercial use]
Authenticate
+userId : String
+password : String = NULL
+errorMessage : String = NULL
+Authenticate()
Store
-storeId : Integer
-demand : Integer
-storeName : String
+toString() : String
<<instantiate>>
Candidate
+chromosome : Integer ArrayList
+fitness : double
-rank : int
-routeTable : Route ArrayList
+Candidate()
+compareTo(Candidate c) : Integer
+toString() : String
<<instantiate>>
<<use>>
GroceryKart
+trucksRequired : Integer = 0
+routingMap : String = NULL
+distanceTravelled : double = 0
+costIncurred : double = 0
+profit : double = 0
+deliveryTime : Integer = 0
+storeDemandTable : ArrayList<Integer> = NULL
+DCStoreMatrix : double[][] = NULL
+generations : Integer = 100
+populationSize : Integer = 1000
+mutationProbability : Double = 0.01
+crossoverProbability : Double = 0.4
+selectionPercentage : Integer = 50
-maxStops : int
-maxStores : int
-maxDemand : int
-maxTime : int
-currentCandidates : Candidate ArrayList
-nextCandidates : Candidate ArrayList
-matingPool : Candidate ArrayList
-stores : Store ArrayList
-storeSize : int
-rankArray : RankedChromosome ArrayList
+GroceryKart()
+proceed(GroceryKart g) : void
+runAlgorithm(GroceryKart g, int i) : void
+saveAndDisplay(GroceryKart g) : String
+generatePopulation() : void
+generateRoutes(Candidate tmp) : Route ArrayList
+greedyGeneratePopulation(double euclidRadius) : void
+evaluateFitnessCandidate(Candidate tmp) : double
+evaluateFitnessRoute(Route tmp) : void
+tournamentSelection() : void
+tournamentRankSelection() : void
+inversion(Candidate can) : Candidate
+mutation(Candidate ArrayList population) : void
+paretoRanking() : void
+nonDominated(Candidate tmp) : boolean
+bestRouteCrossover() : void
+crossover() : void
+crossoverPMX() : void
+copyCandidate(Candidate tmp) : Candidate
+checkInsertionValidity(Route tmp) : boolean
+writeToFile() : void
ReportGeneration
+maintenanceCost : double
+numberTrucks : int
-yearlyCost : double
-totalTravelledDistance : double
-truckPath : Integer ArrayList
+ReportGeneration()
+addMetaData(Document d) : void
+addTitlePage(Document d) : void
+addContent(Document d) : void
+createList(Paragraph p, int i) : void
<<instantiate>>
StoreAnalysis
-demand : int
-fulfill : int
+StoreAnalysis()
<<instantiate>>
TruckAnalysis
+distance : double
+cost : double
+TruckAnalysis()
<<instantiate>>
AdjustFactors
+selectionPercent : int
-crossoverProbability : double
-mutationProbability : double
-g : GroceryKart
+AdjustFactors(selectionPercent, crossoverProbability, mutationProbability)
GroceryKartSavedData
-tmpCandidates : Candidate ArrayList
-tmpStores : Store ArrayList
+GroceryKartSavedData()
+getTotalTrucks(Candidate solution) : int
+getTruckPath(Candidate solution, int truckNumber) : Integer ArrayList
+getTruckDistance(Candidate solution, int truckNumber) : double
+getTruckDailyCost(Candidate solution, int truckNumber) : double
+getStoreDemand(int storeNumber) : int
+getTruckFulfillStore(Candidate solution, int storeNumber) : int
+getMaintenanceCost(Candidate solution, int months) : String
+getVariableCost(Candidate solution, int days) : String
+getTotalDistance(Candidate solution) : String
+readFromFile() : Candidate ArrayList

Figure 3.5 Class Diagram

Chapter 4

IMPLEMENTATION

The proposed system is implemented using Java Applets and Core Java logic. Various features of Java were used in the process like various inbuilt methods, classes etc to create windows and to implement various stages of genetic algorithm.

The Genetic Algorithm has five basic stages:

a. Population Initialization
b. Selection
c. Fitness Evaluation
d. Crossover
e. Mutation

Population initialization, two methods are being used:

1. generatePopulation(): it generates a diverse population
2. greedyGeneratePopulation(): it generates optimum solutions

Selection is being performed by using Tournament Selection concept, which involves running several "tournaments" among a few individuals (or chromosomes) chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover.

Fitness Evaluation is done by using following two methods:

a) Weighted Sum approach
b) Pareto Ranking: Pareto is a state of allocation of resources from which it is impossible to reallocate to make any one individual or preference criterion better off without making at least one individual or preference criterion worse off.

For **Crossover**, user will be given two choices:

BCRC: Best Cost Route Crossover

BCRC not only maintains the candidates' good nature but improves it too, to
drive the force of obtaining better solutions by manipulating existing good candidates.

Algorithm:

```
Step 1: Mating pool of candidates/Population
Step 2: Each candidate has own routing scheme
```

Step 3: Random candidate selected (say c1)

Step 4: According to constraints, select another candidates (say c2i; i=1,2,3,4…)

Step 5: Mate c1, c2i

Step 6: Calculate fitness value

Step 7: i^{th} candidate with best fitness value selected

Step 8: Add selected candidate to new population

PMX: Partially Mapped Crossover

PMX aims at keeping as many positions from the parents. To get this, a sub-string is swapped like in two-point crossover and the values are kept in all other non-conflicting positions.

Then conflicting positions are replaced by values which were swapped to other offspring.

Algorithm:

Step 1: Mating pool of candidates/Population

Step 2: Keep copy of each candidate

Step 3: Randomly select 2 candidates.

Step 4: Assume position p-q are selected for swapping

Step 5: Offsprings generated are o1, o2

Step 6: Consider conflicting positions

For eg 1 and 4 were swapped. Therefore, we must replace the 1 in first position o1 by 4 & so on

Step 7: Calcuate fitness value

Step 8: Add selected candidate to new population

Example:

p1= (1 2 3 4 5 6 7 8 9)

p2= (4 5 2 1 8 7 6 9 3)

Assume the position 4-7 are selected for swapping.

Then the two offspring are given as follows if we omit the conflicting positions.

o1= (* 2 3|1 8 7 6|* 9)

o2= (* * 2|4 5 6 7|9 3)

Now we take the conflicting positions & in what was swapped to the other offspring.

For eg 1 and 4 were swapped. Therefore we have to replace the 1 in first position o1 by 4 & so on.

o1= (4 2 3 1 8 7 6 5 9)

o2= (1 8 2 4 5 6 7 9 3)

Mutation will be done by using Inversion Operator. The length of inversion will be fixed to 2 or 3 which gets selected randomly and assigned to get flipped.

Figure 4.1 shows the file hierarchy or structure of the project grocerykart. There is a package named grocerykart which has all the necessary files for the project. Two libraries have also been included named **itextpdf** which generates a pdf file with best solutions and **JDK 1.8** which help to develop as well as run the application.

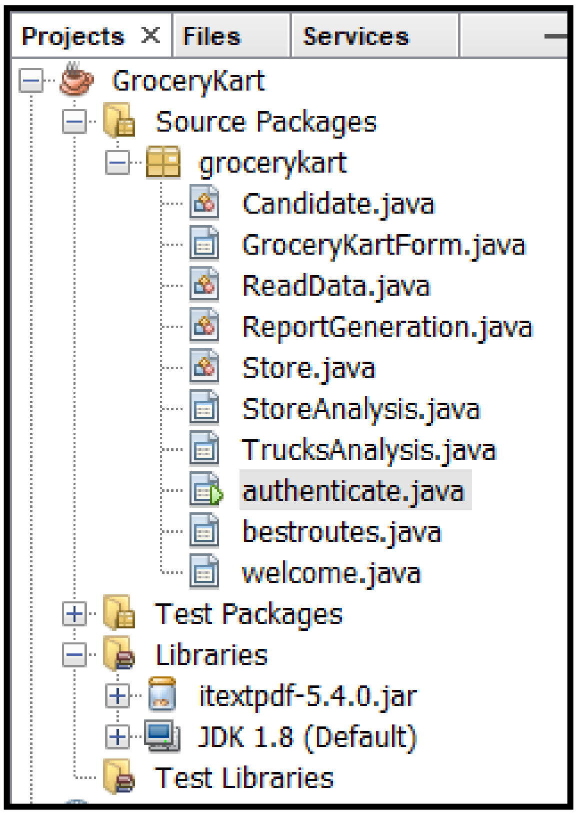


Figure 4.1: File Structure

4.1 USER AUTHENTICATION

Authentication involves the process of identifying an individual, which is usually based on a username and password.

Authenticate(): Default Constructor of Authenticate class which initializes a new User taking User Id and Password as input .The **message** is returned as output by this function.

4.2 SYSTEM DATA

System Data Contains Global Variables, with their Prototypes and default values already set. The Members Of this class are also inherited by Other Subclasses.

Globally declared variables are:

No of trucks

Routing maps

distanceTravelled

CostIncurred

Profit

Efficiency

deliveryTime

userId

storeDemandTable

DCStore Matrix

Generations

populationsize

crossoverProbability

mutationProbability

SelectionPercentage

Grocerykartform (): Default Constructor of **Super Class** Grocerykartform is used to initialize all the global variables mentioned above.

4.3 USER INTERFACE

This class is responsible for retrieving input from the user which mainly consists of Store Demand Table and DC Store Matrix, while all other factors are already containing default values, which can be altered using AdjustFactors class described in section. This class is a Subclass of System data.

Variables used in this module:

No of stores: Total number of stores.

Maxstops: maximum number of stops at which truck an stop

MaxDemand: total demand of all stores

MaxDistance: Total distance covered by all trucks in one day

MaxDeliveryTime : maximum Delivery time within which demand must be fulfilled.

Methods created and used in this module:

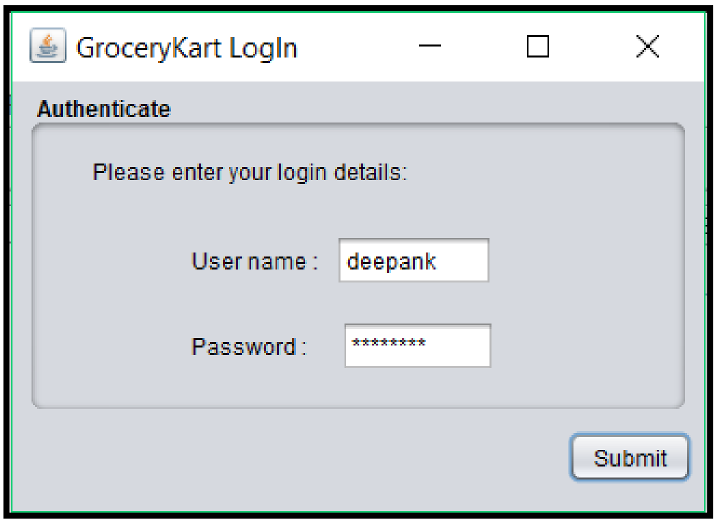
Read data(): default constructor of class Read data.

It invokes mainly two functions:

ReadDemandTable(): it provides the demand of each store.

ReadDistanceMatrix(): it provides the distance of each store from the warehouse and the distance among the stores.

It is the authentication window which requires the username and password as the input from the user to start the working of the grocery kart.


GroceryKart LogIn

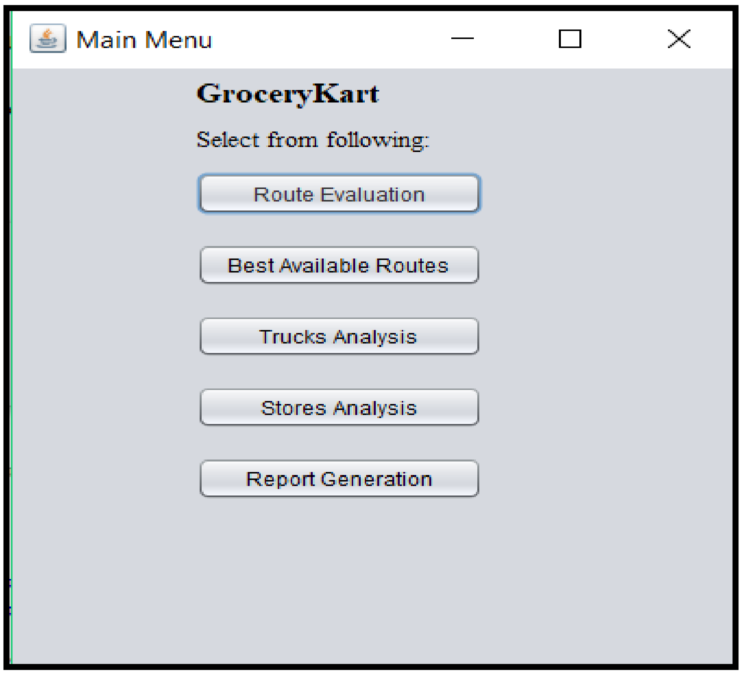
Authenticate

Please enter your login details:

User name : deepank

Password : ********

This window is the main welcome window in which the user chooses its options.

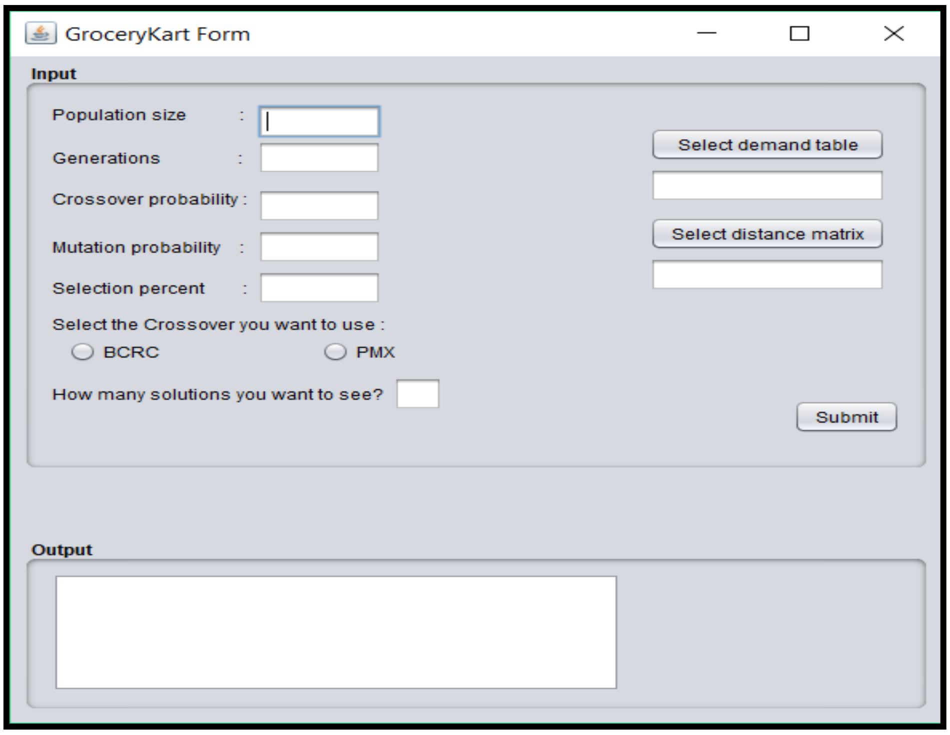

Main Menu

GroceryKart

Select from following:

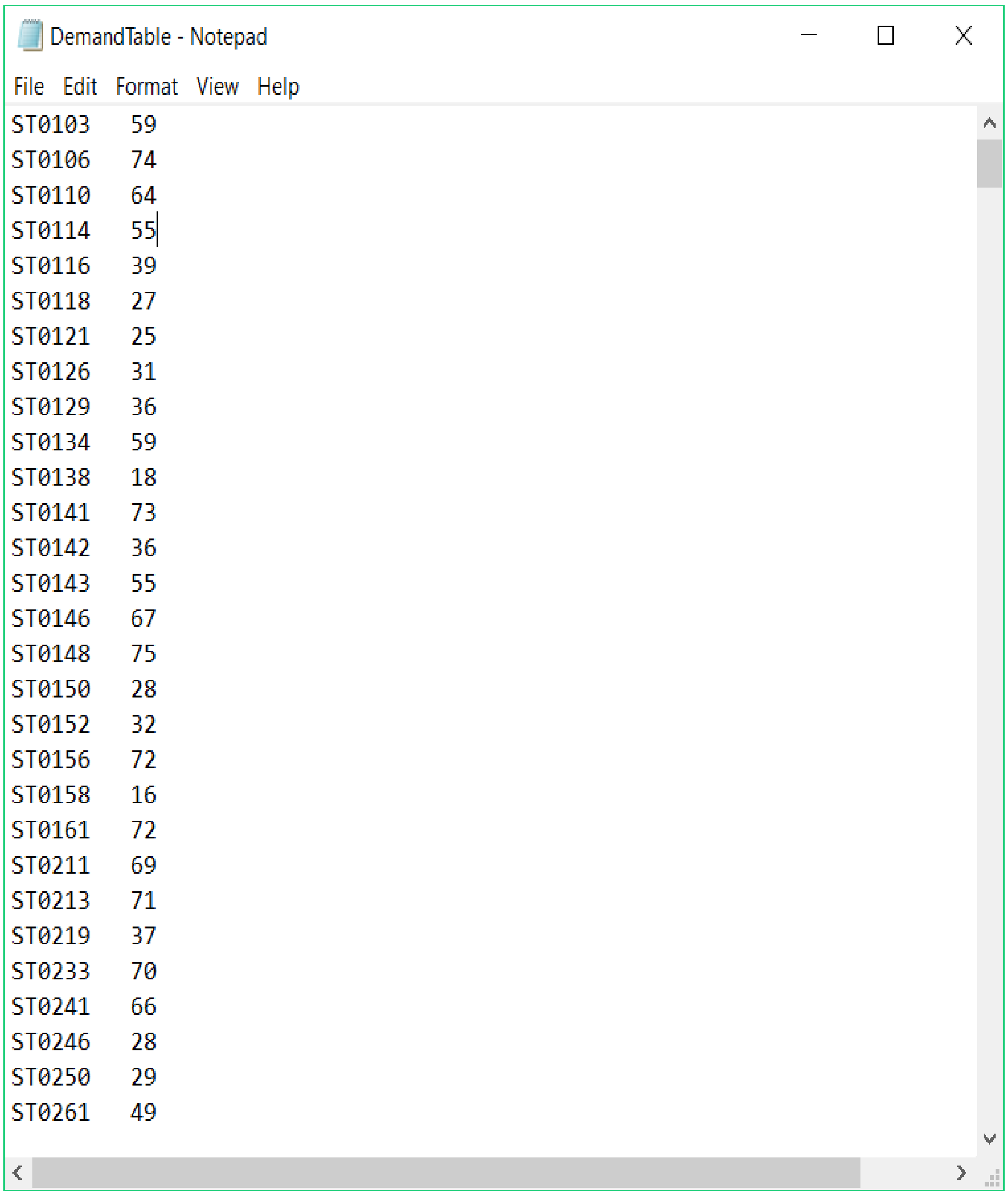
There are 5 choices for the user. The functionality of Best Available Routes, Trucks Analysis, Stores Analysis, Report Generation are dependent on Route Evaluation which takes the required inputs from the user.

This is the route evaluation window. The user inputs the various fields to calculate the various optimum results produced providing the trucks required and the routes and distance travelled by these trucks.



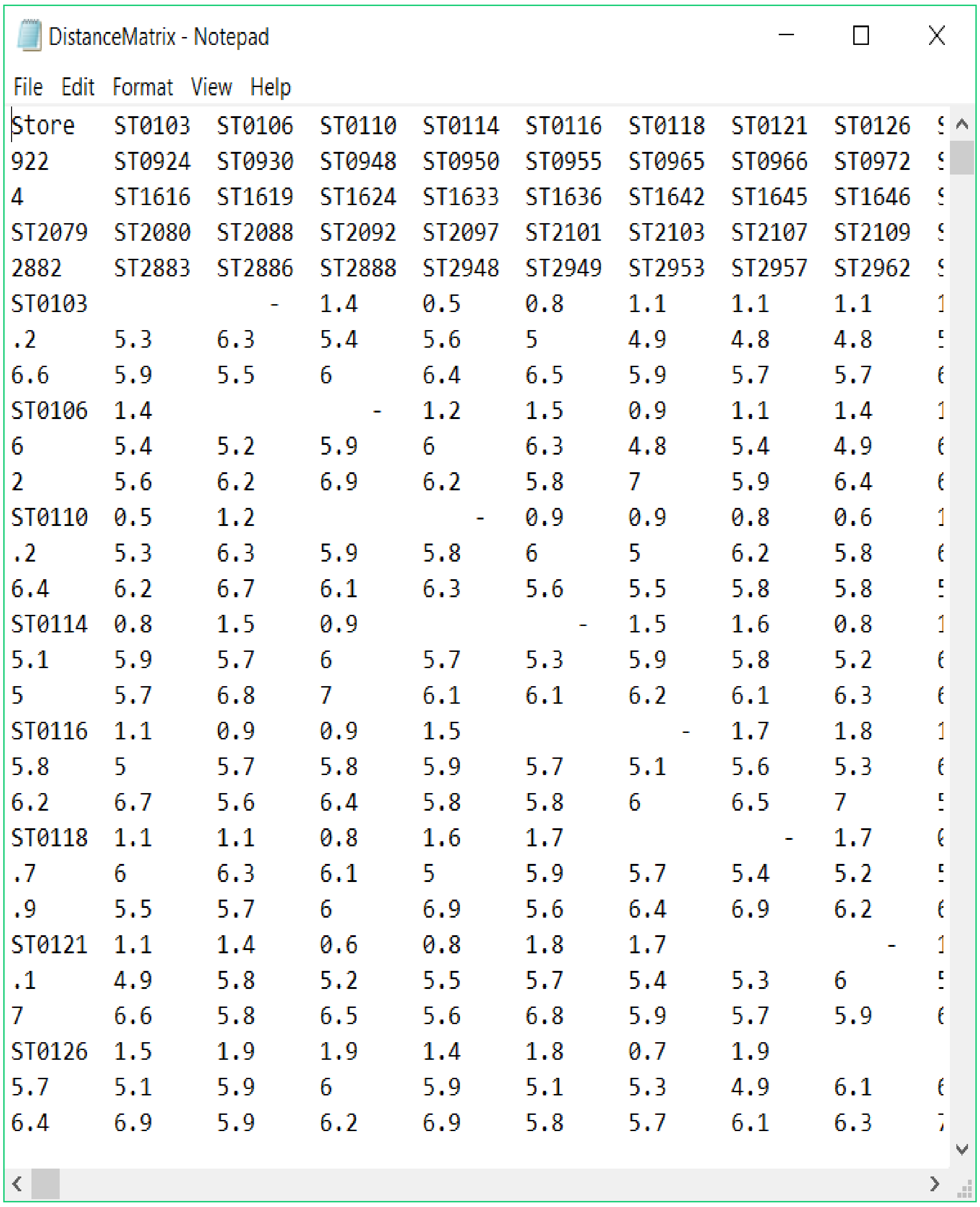
There are two **datasets** also which are required by this GroceryKart form as an input. These datasets are in form of text files which need to browsed from file explorer and imported into the grocerykart form.

a) Demand Table
b) Distance Matrix



```
ST0103    59
ST0106    74
ST0110    64
ST0114    55
ST0116    39
ST0118    27
ST0121    25
ST0126    31
ST0129    36
ST0134    59
ST0138    18
ST0141    73
ST0142    36
ST0143    55
ST0146    67
ST0148    75
ST0150    28
ST0152    32
ST0156    72
ST0158    16
ST0161    72
ST0211    69
ST0213    71
ST0219    37
ST0233    70
ST0241    66
ST0246    28
ST0250    29
ST0261    49
```

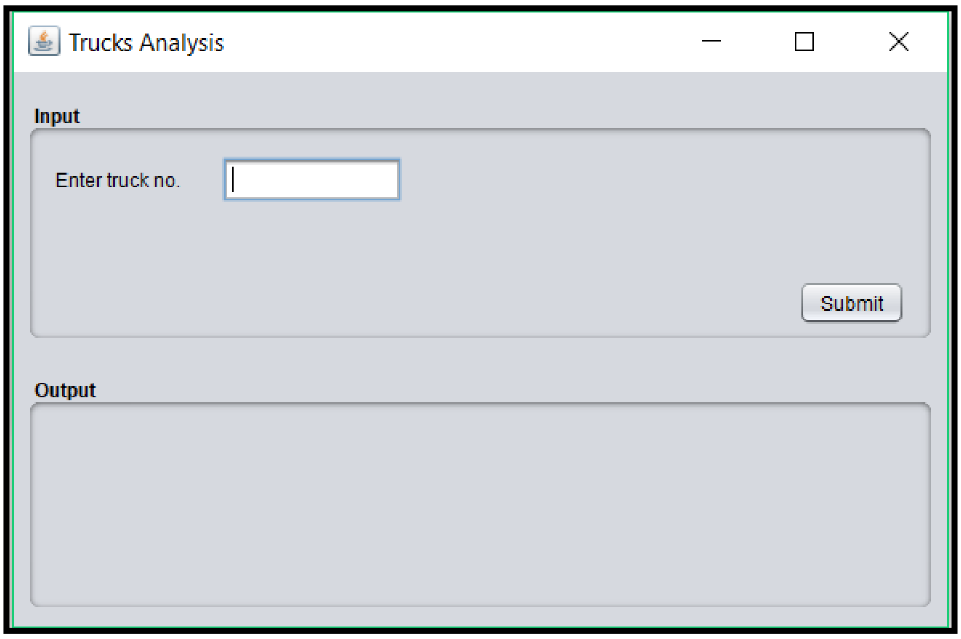
1. **Demand Table:** The Demand table consists of two columns. First column is Store Id and second column is demand of corresponding store.



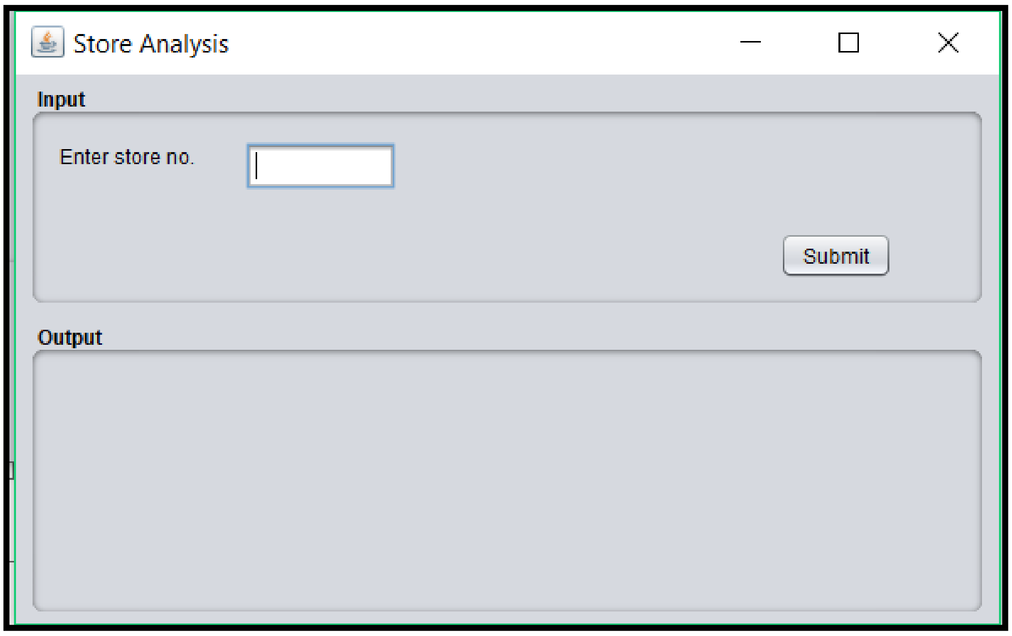
```
Store   ST0103  ST0106  ST0110  ST0114  ST0116  ST0118  ST0121  ST0126  9
922     ST0924  ST0930  ST0948  ST0950  ST0955  ST0965  ST0966  ST0972  9
4       ST1616  ST1619  ST1624  ST1633  ST1636  ST1642  ST1645  ST1646  9
ST2079  ST2080  ST2088  ST2092  ST2097  ST2101  ST2103  ST2107  ST2109  9
2882    ST2883  ST2886  ST2888  ST2948  ST2949  ST2953  ST2957  ST2962  9
ST0103              -   1.4     0.5     0.8     1.1     1.1     1.1     1
.2      5.3     6.3     5.4     5.6     5       4.9     4.8     4.8     5
6.6     5.9     5.5     6       6.4     6.5     5.9     5.7     5.7     6
ST0106  1.4                 -   1.2     1.5     0.9     1.1     1.4     1
6       5.4     5.2     5.9     6       6.3     4.8     5.4     4.9     6
2       5.6     6.2     6.9     6.2     5.8     7       5.9     6.4     6
ST0110  0.5     1.2                 -   0.9     0.9     0.8     0.6     1
.2      5.3     6.3     5.9     5.8     6       5       6.2     5.8     6
6.4     6.2     6.7     6.1     6.3     5.6     5.5     5.8     5.8     5
ST0114  0.8     1.5     0.9                 -   1.5     1.6     0.8     1
5.1     5.9     5.7     6       5.7     5.3     5.9     5.8     5.2     6
5       5.7     6.8     7       6.1     6.1     6.2     6.1     6.3     6
ST0116  1.1     0.9     0.9     1.5                 -   1.7     1.8     1
5.8     5       5.7     5.8     5.9     5.7     5.1     5.6     5.3     6
6.2     6.7     5.6     6.4     5.8     5.8     6       6.5     7       5
ST0118  1.1     1.1     0.8     1.6     1.7                 -   1.7     6
.7      6       6.3     6.1     5       5.9     5.7     5.4     5.2     5
.9      5.5     5.7     6       6.9     5.6     6.4     6.9     6.2     6
ST0121  1.1     1.4     0.6     0.8     1.8     1.7                 -   1
.1      4.9     5.8     5.2     5.5     5.7     5.4     5.3     6       5
7       6.6     5.8     6.5     5.6     6.8     5.9     5.7     5.9     6
ST0126  1.5     1.9     1.9     1.4     1.8     0.7     1.9
5.7     5.1     5.9     6       5.9     5.1     5.3     4.9     6.1     6
6.4     6.9     5.9     6.2     6.9     5.8     5.7     6.1     6.3     7
```

2. **Distance Matrix:** The distance matrix shows distance between various stores. For example, say D_{ij} is a value in the matrix. Then D_{ij} is the distance between store$_i$ and store$_j$.

This is the Truck Analysis window. It takes the truck number as the input value from the user to provide various analysis or result on that truck bought by the Grocery Kart.



This is Store Analysis window. It takes the Store number as the input value from the user to provide various analysis or other information on that store.



AdjustFactors Class

It is responsible for adjusting some factors according to the requirement of the user. Factors such as **mutation probability, crossover probability, and maximum number of stops** can be adjusted easily.

AdjustFactor(): default constructor reinitialises the values of factors to be adjusted.

4.4 Processing Module

The Processing module contains the working mechanism of the algorithm. It defines the flow of randomly generated possible solutions and optimising them to the best solution. It contains the following classes:
GroceryKart
Store
Candidate

4.4.1 GroceryKart Class

This class is the main class in the project, the main functions involved in calculating the optimum solution such as Selection, Mutation, Crossover are invoked by this class. Functions invoked by GroceryKart class are:

Generatepopulation() : This method generates candidates randomly. A permutation of 600 integers from 1 to 600 is shuffled every time. Each generated candidate is stored in currentCandidates after evaluating the fitness and routes.

greedyGeneratePopulation(): The method selects the node in a given euclidRadius. Based on greedy strategy it takes the nearest node in a given euclidRadius. It then keeps on adding nearest neighbours, finally appending the last. 600th node completing a candidate, generating routes and evaluating fitness. It then gets added to currentCandidates.

TournamentSelection() : When paretoRanking() is used , the selection procedure is Tournament Rank Selection as fitness is replaced by the ranks, candidatesget assigned during evaluation. It uses an 80% selection of good candidates and 20% selection of candidates randomly.

ParetoRanking() : paretoRanking is based on non-dominant solutions.

Crossover() : Here two types of Crossovers have been used:
BCRC: Best Cost Route Crossover
It helps not only maintaining candidates' good nature but also improves it too, to drive

the force of obtaining a better solution by manipulating existing good candidates

PMX: Partially Mapped Crossover

PMX Crossover aims at keeping as many positions from parents as possible. To achieve this goal, a sub-string is swapped like in two-point crossover and the values are kept in all other non-conflicting positions. The conflicting positions are replaced by the values which were swapped to other offspring.

The figure 4.2 best explains the best-cost route crossover in a nutshell.

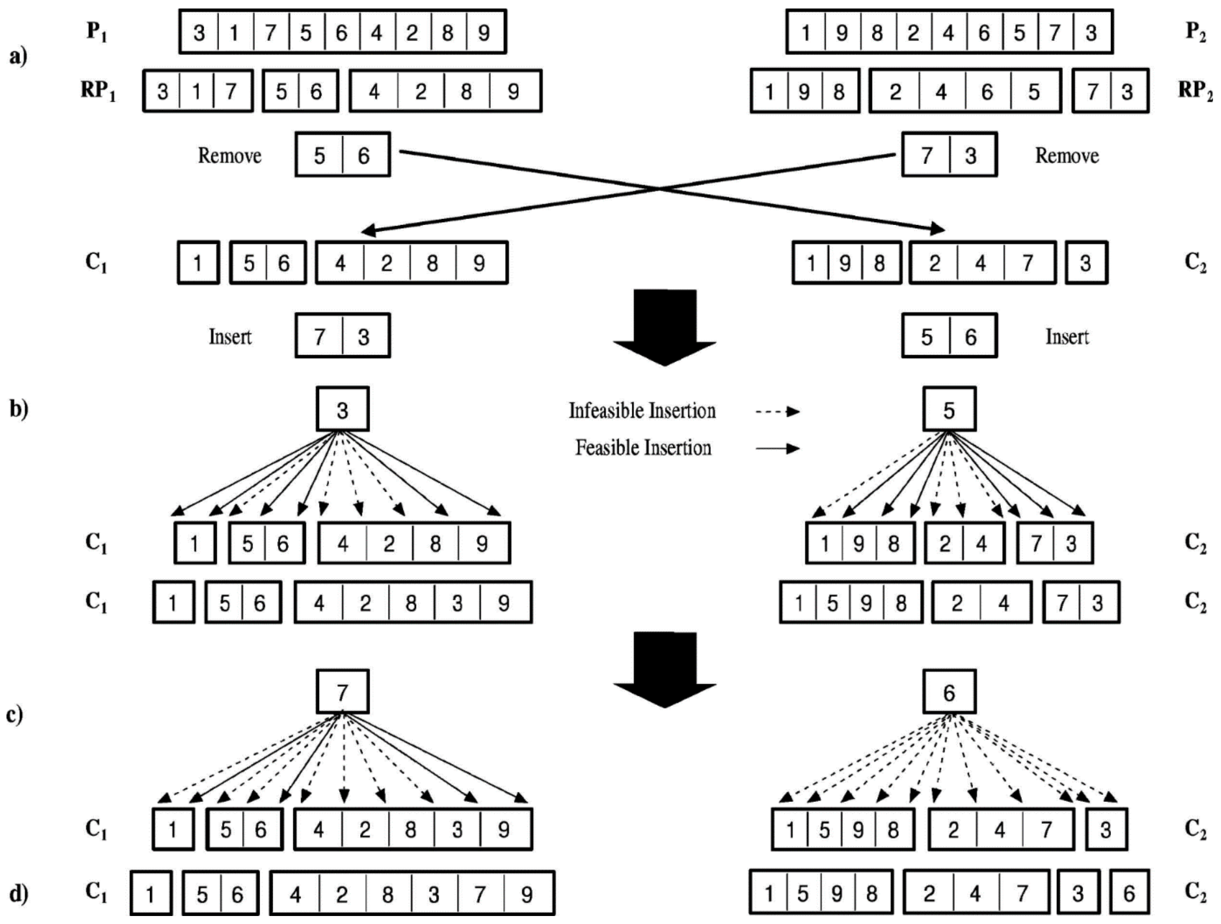


Figure 4.2 Best Cost Route Crossover

Mutation() : Mutation is being achieved by using Inversion operator. The length has been fixed to 2 or 3. This operator chooses a candidate randomly and some part of it gets flipped.

4.4.2 Store class

This class contains all the detailing about all the stores in the city where the grocery is to be delivered. It serves as an entity and describes all the demand of a store.

The methods which are invoked in this class are**:**

toString() : to show the representation of store.

4.4.3 Candidate Class

This class models the solution to be generated. It includes the best routes available and the fitness.

4.5 Output Module

Output Module provides the required results for the user.

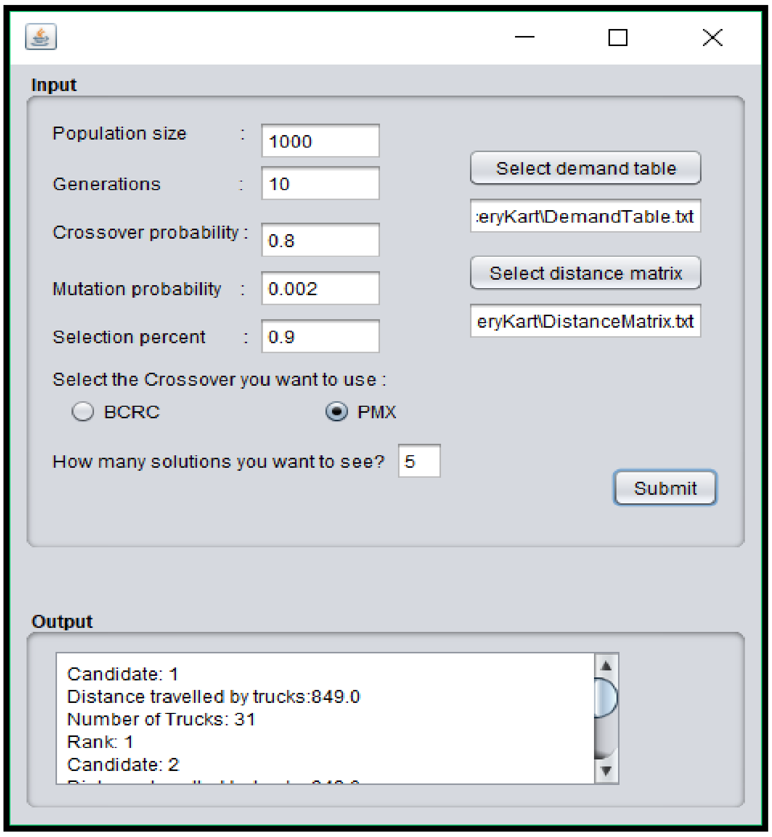
Classes used in this module are:

Store analysis: It includes all the demands of a store and provides which truck satisfies it.

Truck Analysis: It provides the best path for a truck and total distance travelled by it.

Route Evaluation window: This is window provides various candidate generation with the total distance travelled and maximum number of trucks required the sorts the result according to the ranks.

The required inputs from the user are Population size, number of generations, Crossover Probability, Mutation Probability, Selection Percent, Type of Crossover to be used, number of solutions to be shown and two datasets: Demand table and Distance Matrix.



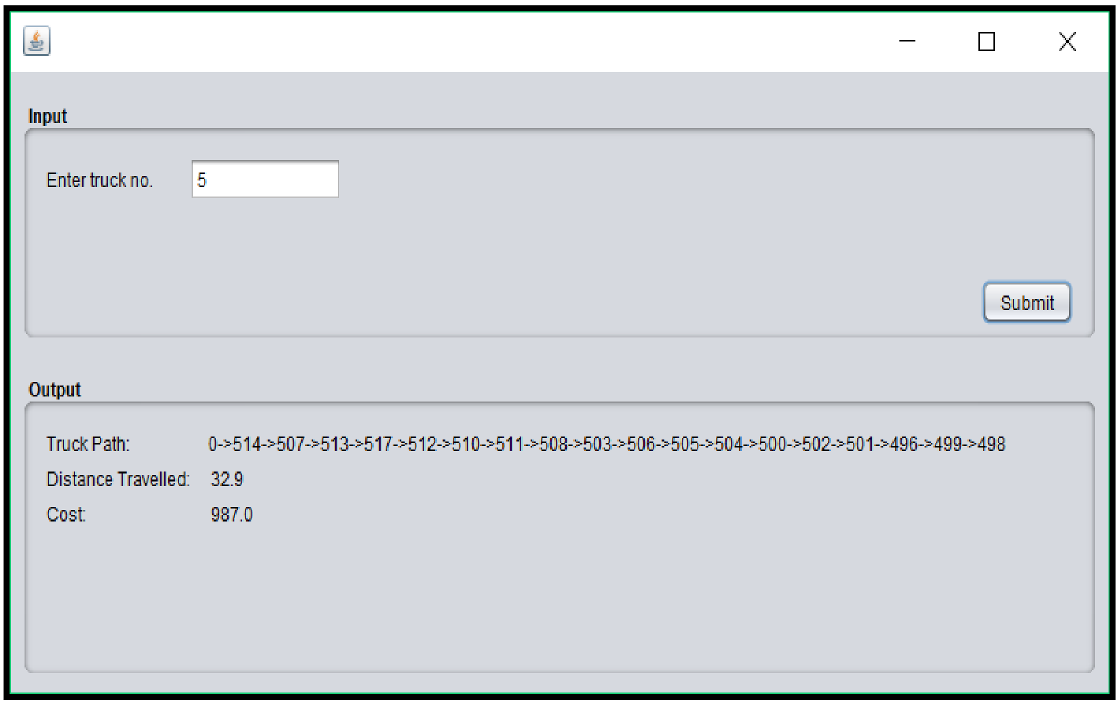
Input

Population size : 1000
Generations : 10
Crossover probability : 0.8
Mutation probability : 0.002
Selection percent : 0.9
Select the Crossover you want to use :
BCRC [✓ PMX]
How many solutions you want to see? 5

Output

```
Candidate: 1
Distance travelled by trucks:849.0
Number of Trucks: 31
Rank: 1
Candidate: 2
```

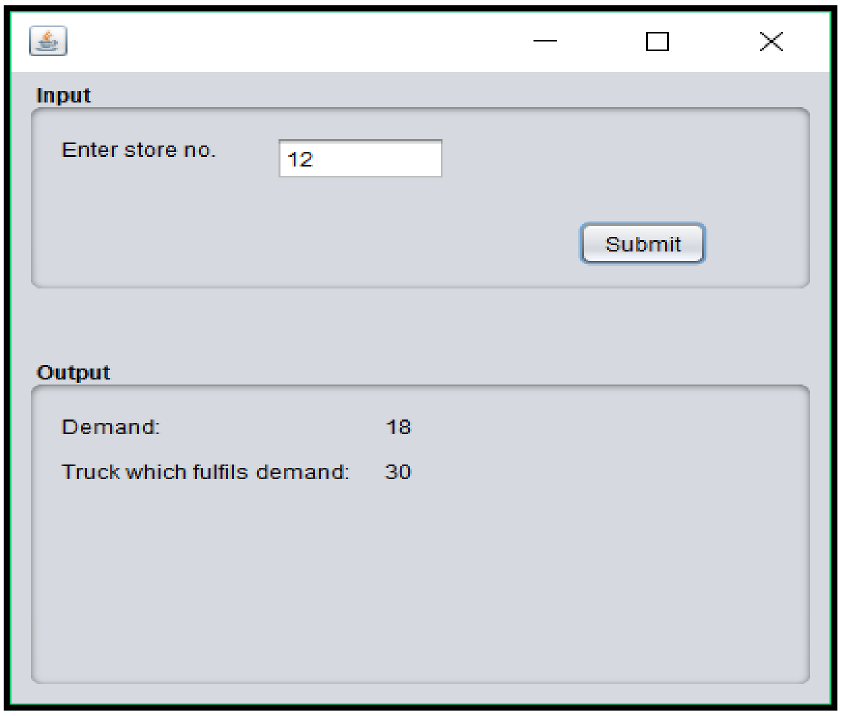
Truck Analysis: This window shows the result for the truck number specified by the user. It provides the path of that truck (i.e. which all stores it visits) and the distance travelled by that truck and the cost incurred. The required input is truck number for which analysis is needed to be done.



```
Enter truck no.    5

Truck Path:          0->514->507->513->517->512->510->511->508->503->506->505->504->500->502->501->496->499->498
Distance Travelled:  32.9
Cost:                987.0
```

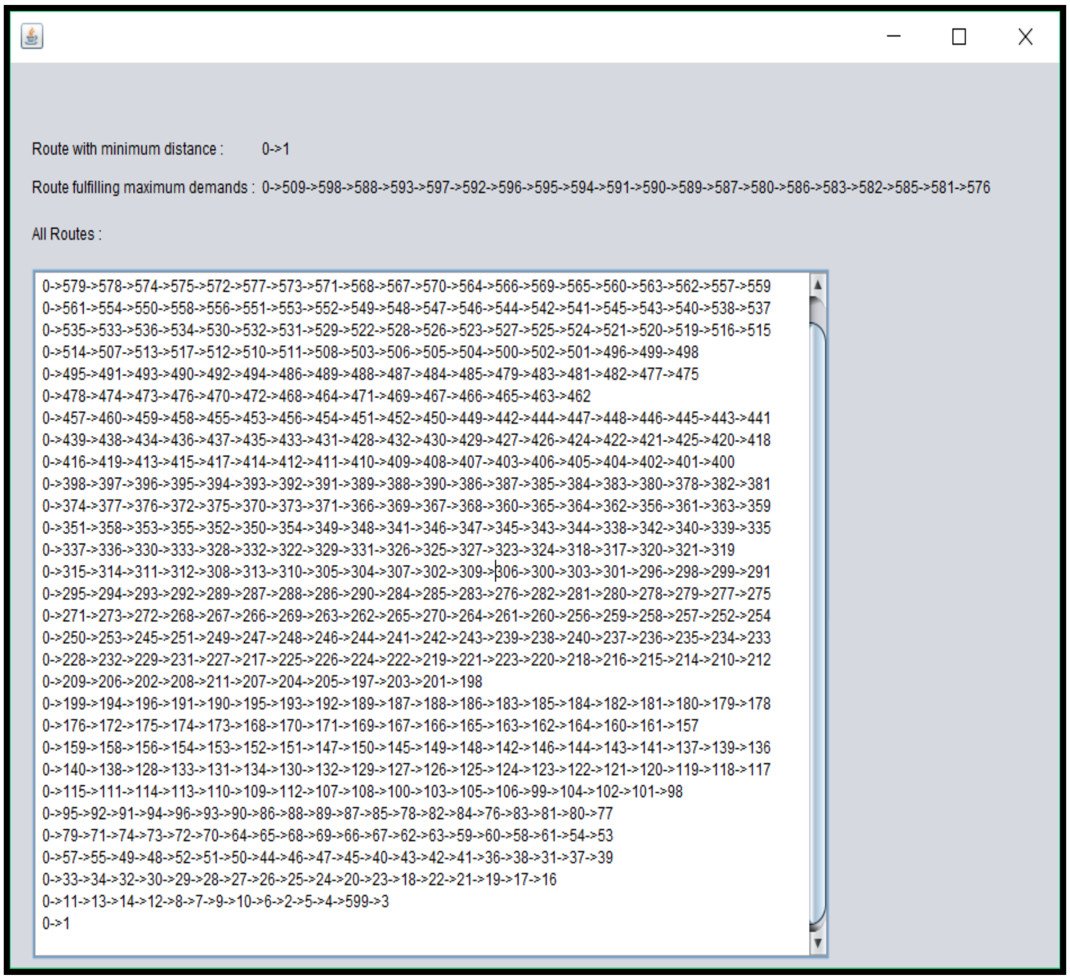
Store Analysis: This window shows the result for the store number specified by the user. It provides the demand of that store and which truck fulfils it. The input parameter store number is required.



```
Enter store no.    12

Demand:                      18
Truck which fulfils demand:  30
```

Best Available Routes: This window provides us the best available routes for the distribution of Grocery from the warehouse to various stores.



```
Route with minimum distance :      0->1

Route fulfilling maximum demands :  0->509->598->588->593->597->592->596->595->594->591->590->589->587->580->586->583->582->585->581->576

All Routes :

0->579->578->574->575->572->577->573->571->568->567->570->564->566->569->565->560->563->562->557->559
0->561->554->550->558->556->551->553->552->549->548->547->546->544->542->541->545->543->540->538->537
0->535->533->536->534->530->532->531->529->522->528->526->523->527->525->524->521->520->519->516->515
0->514->507->513->517->512->510->511->508->503->506->505->504->500->502->501->496->499->498
0->495->491->493->490->492->494->486->489->488->487->484->485->479->483->481->482->477->475
0->478->474->473->476->470->472->468->464->471->469->467->466->465->463->462
0->457->460->459->458->455->453->456->454->451->452->450->449->442->444->447->448->446->445->443->441
0->439->438->434->436->437->435->433->431->428->432->430->429->427->426->424->422->421->425->420->418
0->416->419->413->415->417->414->412->411->410->409->408->407->403->406->405->404->402->401->400
0->398->397->396->395->394->393->392->391->389->388->390->386->387->385->384->383->380->378->382->381
0->374->377->376->372->375->370->373->371->366->369->367->368->360->365->364->362->356->361->363->359
0->351->358->353->355->352->350->354->349->348->341->346->347->345->343->344->338->342->340->339->335
0->337->336->330->333->328->332->322->329->331->326->325->327->323->324->318->317->320->321->319
0->315->314->311->312->308->313->310->305->304->307->302->309->306->300->303->301->296->298->299->291
0->295->294->293->292->289->287->288->286->290->284->285->283->276->282->281->280->278->279->277->275
0->271->273->272->268->267->266->269->263->262->265->270->264->261->260->256->259->258->257->252->254
0->250->253->245->251->249->247->248->246->244->241->242->243->239->238->240->237->236->235->234->233
0->228->232->229->231->227->217->225->226->224->222->219->221->223->220->218->216->215->214->210->212
0->209->206->202->208->211->207->204->205->197->203->201->198
0->199->194->196->191->190->195->193->192->189->187->188->186->183->185->184->182->181->180->179->178
0->176->172->175->174->173->168->170->171->169->167->166->165->163->162->164->160->161->157
0->159->158->156->154->153->152->151->147->150->145->149->148->142->146->144->143->141->137->139->136
0->140->138->128->133->131->134->130->132->129->127->126->125->124->123->122->121->120->119->118->117
0->115->111->114->113->110->109->112->107->108->100->103->105->106->99->104->102->101->98
0->95->92->91->94->96->93->90->86->88->89->87->85->78->82->84->76->83->81->80->77
0->79->71->74->73->72->70->64->65->68->69->66->67->62->63->59->60->58->61->54->53
0->57->55->49->48->52->51->50->44->46->47->45->40->43->42->41->36->38->31->37->39
0->33->34->32->30->29->28->27->26->25->24->20->23->18->22->21->19->17->16
0->11->13->14->12->8->7->9->10->6->2->5->4->599->3
0->1
```

4.6 Report Generation

It is a segment of Output module which concludes the results. It includes the function reportgeneration() which generates the best 10 solutions providing –

- Maintainance cost
- Yearly cost
- No. of trucks to buy
- Total distance travelled

4.7 Sample Coding

//Authenticate

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
username= text.getText();

password=pass.getText();
if(username.equals("deepank") &&
password.equals("kartikey")){
welcome w = new welcome();
w.main();
jLabel5.setText("");
}
else
if(username.equals(""))
{
jLabel5.setText("Enter Username");
}
else
if(password.equals(""))
{
jLabel5.setText("Enter Password");
}
else
if(!username.equals("deepank"))
{
jLabel5.setText("Enter correct Username");
}
else
if(!password.equals("kartikey"))
{
jLabel5.setText("Enter correct Password");
}
```

```
else
{
jLabel5.setText("Enter correct login details");
}
}

public class GroceryKartForm extends javax.swing.JFrame
{
public static int p = 0;
public static int g = 0;
public static double c = 0;
public static double m = 0;
public static double s = 0;
public static int n = 0;
public static File file;
public static File file1;
public static boolean flag=false;
public static boolean flag1=false;
public static int val=0;
private void groupButton(){
ButtonGroup bg = new ButtonGroup();
bg.add(jRadioButton1);
bg.add(jRadioButton2);
}

public GroceryKartForm() {
initComponents();
groupButton();
}

private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
```

```
if (text.getText().isEmpty() || text1.getText().isEmpty()
|| text2.getText().isEmpty() || text3.getText().isEmpty()
|| text4.getText().isEmpty() || text5.getText().isEmpty()
|| textdemand.getText().isEmpty() ||
textdistance.getText().isEmpty()) {
jLabel7.setText("ERROR: Please enter all the required
details");
}

p = Integer.parseInt(text.getText());
g = Integer.parseInt(text1.getText());
c = Double.parseDouble(text2.getText());
m = Double.parseDouble(text3.getText());
s = Double.parseDouble(text4.getText());
n = Integer.parseInt(text5.getText());
flag=jRadioButton1.isSelected();
flag1=jRadioButton2.isSelected();
//GroceryKart.proceed();
GroceryKart gr = new GroceryKart();
gr.proceed(gr);
//int progressValue = 100/gr.generations;
// System.out.println(progressValue);
for (int i = 0; i < gr.generations; i++) {
gr.runAlgorithm(gr,i);
//System.out.println((i+1)*progressValue);
}
String display = gr.saveAndDisplay(gr);
jTextArea1.setText(display);
}

class AdjustFactors {
/*
* Used to set the Genetic operator values as fetched from
the user.*/
```

```
GroceryKart g;
int selectionPercent;
double crossoverProbability;
double mutationProbability;

AdjustFactors(int selectionPercent, double
crossoverProbibility, double mutationProbability) {
g = new GroceryKart();
this.selectionPercent = selectionPercent;
this.crossoverProbability = crossoverProbibility;
this.mutationProbability = mutationProbability;
}
}
```

Store Class

```
package grocerykart;
import java.io.Serializable;
public class Store implements Serializable {
int storeId;
String storeName;
int demand;
@Override
public String toString() {
String ret = "Store Id: " + storeId + "\t";
ret += "Store Name: " + storeName + "\t";
ret += "Demand: " + demand;
return ret;
}
}
```

Candidate Class

```
package grocerykart;
class Candidate implements Comparable<Candidate>,
Serializable {
```

```
ArrayList<Integer> chromosome;
double fitness;
int rank;
ArrayList<Route> routeTable;

public Candidate() {
chromosome = new ArrayList<>();
routeTable = new ArrayList<>();
}

/*
* toString() method overriden to get a nice formatted
output of a given candidate
*/
@Override
public String toString() {
int[] tmp = new int[chromosome.size()];
String ret = "";
for (Integer i : chromosome) {
ret += (i + " ");
}
return ret;
}
@Override
public int compareTo(Candidate tmp) {
if(this.fitness < tmp.fitness) {
return -1;
}
else if(this.fitness == tmp.fitness) {
return 0;
} else {
return 1;
}
}}
```

```
public void generatePopulation() {
for (int i = 0; i < populationSize; i++) {
ArrayList<Integer> tmp = new ArrayList<Integer>();
tmp.add(0);
for (int j = 1; j < storeSize; j++) {
tmp.add(j);
}
Random r = new Random();
Candidate temp = new Candidate();

int tmpncustomers = storeSize;
temp.chromosome.add(tmp.get(0));
tmp.remove(0);
tmpncustomers--;
for (int j = 1; j < storeSize; j++) {
int tmpindex = r.nextInt(tmpncustomers);
int tmpadd = tmp.get(tmpindex);
tmp.remove(tmpindex);
temp.chromosome.add(tmpadd);
tmpncustomers--;
}
temp.routeTable = generateRoutes(temp);
temp.fitness = evaluateFitnessCandidate(temp);
currentCandidates.add(temp);
}
}

public void greedyGeneratePopulation(double euclidRadius) {
for (int i = 0; i < populationSize; i++) {
Candidate temp = new Candidate();
ArrayList<Integer> tmp = new ArrayList<Integer>();
tmp.add(0);
for (int j = 1; j < storeSize; j++) {
tmp.add(j);
```

```
}
temp.chromosome.add(0);
tmp.remove(0);
Random r = new Random();
int rcustomer = r.nextInt(storeSize - 2) + 1;
int custVal = tmp.get(rcustomer);
tmp.remove(rcustomer);
temp.chromosome.add(custVal);
int tmpncustomers = storeSize;
for (int j = 2; j < storeSize; j++) {

// check the nearest neighbour to rcustomer in euclid
radius specified
int minindex = -1;
double mindist = Double.MAX_VALUE;
for (int k = 0; k < tmpncustomers - 2; k++) {
int l = tmp.get(k);
if (DCStoreMatrix[rcustomer][l] != 0 &&
DCStoreMatrix[rcustomer][l] < mindist &&
DCStoreMatrix[rcustomer][l] < euclidRadius) {
mindist = DCStoreMatrix[rcustomer][j];
minindex = k;
}
}
if (minindex != -1) {
custVal = tmp.get(minindex);
tmp.remove(minindex);
temp.chromosome.add(custVal);
rcustomer = minindex;
} else {
rcustomer = r.nextInt(tmpncustomers);
custVal = tmp.get(rcustomer);
tmp.remove(rcustomer);
temp.chromosome.add(custVal);
}
tmpncustomers--;
}
temp.routeTable = generateRoutes(temp);
temp.fitness = evaluateFitnessCandidate(temp);
currentCandidates.add(temp);
}
}

public void paretoRanking() {
rankArray = new ArrayList<RankedChromosome>();
int currRank = 1;
int N = populationSize;
int m = populationSize;
while (N != 0) {
RankedChromosome temp = new RankedChromosome();
temp.rank = new ArrayList<Candidate>();
for (int i = 0; i < m; i++) {
if (nonDominated(currentCandidates.get(i)) == true) {
currentCandidates.get(i).rank = currRank;
}
}

for (int i = 0; i < m && i < N; i++) {
if (currentCandidates.get(i).rank == currRank) {
temp.rank.add(currentCandidates.remove(i));
N--;
i--;
}
}
rankArray.add(temp);
currRank++;
m = N;
}}
```

Following is the procedure to find that a given candidate is non-dominated in a given population:

```
private boolean nonDominated(Candidate tmp) {
for (Candidate temp : currentCandidates) {
if (temp == tmp) {
continue;
}
if ((temp.fitness <= tmp.fitness && temp.routeTable.size() < tmp.routeTable.size()) || (temp.fitness < tmp.fitness && temp.routeTable.size() <= tmp.routeTable.size())) {
// tmp is dominated by temp
return false;
} else if (temp.fitness > tmp.fitness && temp.routeTable.size() > tmp.routeTable.size()) {
// tmp is non-dominated
} else {
// not comparable
}
}
return true;
}

public void tournamentRankSelection() {
matingPool = new ArrayList<Candidate>();
for (RankedChromosome r : rankArray) {
for (Candidate c : r.rank) {
currentCandidates.add(c);
}
}
ArrayList<Integer> populationSet = new
ArrayList<Integer>();
for (int i = 0; i < populationSize; i++) {
populationSet.add(i);
}
```

```
int tmpncustomers = populationSize;
for (int k = 0; k < SELECTIONSIZE / 2; k++) {
ArrayList<Integer> tournamentSet = new
ArrayList<Integer>();
ArrayList<Integer> fitnessTournamentSet = new
ArrayList<Integer>();
for (int i = 0; i < 4; i++) {
Random r = new Random();

int index = r.nextInt(tmpncustomers);
int tmpnum = populationSet.get(index);
populationSet.remove(index);
tournamentSet.add(tmpnum);
fitnessTournamentSet.add(currentCandidates.get(index).rank)
;
tmpncustomers--;
}
Random r = new Random();
double gutter = r.nextDouble();
if (gutter < 0.8) {
double minFit = Double.MAX_VALUE;
int minindex = -1;

for (int i = 0; i < 4; i++) {
if (fitnessTournamentSet.get(i) < minFit) {
minFit = fitnessTournamentSet.get(i);
minindex = i;
}
}
Candidate tmp =
currentCandidates.get(tournamentSet.get(minindex));
matingPool.add(tmp);
tournamentSet.remove(minindex);
}
else {
int index = r.nextInt(4);
Candidatetmp=currentCandidates.get(tournamentSet.get(index)
);
matingPool.add(tmp);
tournamentSet.remove(index);
}
populationSet.addAll(tournamentSet);
tmpncustomers += 3;
}
}

private void bestRouteCrossover() {
Random r = new Random();
ArrayList<Integer> temp = new ArrayList<Integer>();
int candidatesSize = matingPool.size();
for (int i = 0; i < candidatesSize; i++) {
temp.add(i);
}
int parent1Index = temp.remove(r.nextInt(candidatesSize));
int parent2Index = temp.remove(r.nextInt(candidatesSize -
1));
Candidate parent1 = matingPool.get(parent1Index);
Candidate parent2 = matingPool.get(parent2Index);

Candidate copyParent1 = copyCandidate(parent1);
Candidate copyParent2 = copyCandidate(parent2);
// Selecting key for Parent 1
int tmp = r.nextInt(copyParent1.routeTable.size());
Route keyParent1 = new Route();
for (int i : copyParent1.routeTable.get(tmp).route) {
keyParent1.route.add(i);
}
```

```
// Selecting key for Parent 2
tmp = r.nextInt(copyParent2.routeTable.size());
Route keyParent2 = new Route();
for (int i : copyParent2.routeTable.get(tmp).route) {
keyParent2.route.add(i);
}
ArrayList<Route> newalroutesParent2 =
removeSelectedElements(keyParent1, copyParent2.routeTable);
ArrayList<Route> newalroutesParent1 =
removeSelectedElements(keyParent2, copyParent1.routeTable);

for (int i : keyParent1.route) {
if (i == 0) {
continue;
}
boolean flag = false;
ArrayList<Candidate> tmpCandidates = new
ArrayList<Candidate>();
for (int k = 0; k < newalroutesParent2.size(); k++) {
Route jroute = newalroutesParent2.get(k);
for (int j = 1; j < jroute.route.size(); j++) {
Candidate tmpCopyCandidate = copyCandidate(copyParent2);
tmpCopyCandidate.routeTable.get(k).route.add(j, i);
if
(checkInsertionValidity(tmpCopyCandidate.routeTable.get(k))
) {
flag = true;
tmpCopyCandidate.fitness =
evaluateFitnessCandidate(tmpCopyCandidate);
tmpCandidates.add(tmpCopyCandidate);
}
}
}
Collections.sort(tmpCandidates);
```

```
if (tmpCandidates.size() != 0) {
copyParent2 = tmpCandidates.get(0);
}
}
for (int i : keyParent2.route) {
if (i == 0) {
continue;
}
boolean flag = false;
ArrayList<Candidate> tmpCandidates = new
ArrayList<Candidate>();
for (int k = 0; k < newalroutesParent1.size(); k++) {
Route jroute = newalroutesParent1.get(k);
for (int j = 1; j < jroute.route.size(); j++) {
Candidate tmpCopyCandidate = copyCandidate(copyParent1);
tmpCopyCandidate.routeTable.get(k).route.add(j, i);

if
(checkInsertionValidity(tmpCopyCandidate.routeTable.get(k))
) {
flag = true;
tmpCopyCandidate.fitness =
evaluateFitnessCandidate(tmpCopyCandidate);
tmpCandidates.add(tmpCopyCandidate);
}
}
}

if (flag == false) {
copyParent1.routeTable.get(copyParent1.routeTable.size()-
1).route.add(i);
copyParent1.fitness =
evaluateFitnessCandidate(copyParent1);
tmpCandidates.add(copyParent1);
```

```
}
Collections.sort(tmpCandidates);
copyParent1 = tmpCandidates.get(0);
}
nextCandidates.add(copyParent1);
nextCandidates.add(copyParent2);
}

private void crossover() {
nextCandidates = new ArrayList<Candidate>();
int crossoverPopulationSize = matingPool.size();
for (int i = 0; i < crossoverPopulationSize; i++) {
bestRouteCrossover();
}
Collections.sort(matingPool);
for(int i=0;i<matingPool.size();i++) {

nextCandidates.add(matingPool.get(i));
}
}

private void crossoverPMX() {
Collections.sort(matingPool);
Random r = new Random();
nextCandidates = new ArrayList<Candidate>();
for (int k = 0; k < populationSize / 2; k++) {
double probability = r.nextDouble();
if (probability < crossoverProbability) {
int candidateSize = matingPool.size();
int parent1Index = r.nextInt(candidateSize);
int parent2Index = r.nextInt(candidateSize);
while (parent1Index == parent2Index) {
parent2Index = r.nextInt(candidateSize);
}
```

```
Candidate parent1 = matingPool.get(parent1Index);
Candidate parent2 = matingPool.get(parent2Index);
Candidate copyParent1 = copyCandidate(parent1);
Candidate copyParent2 = copyCandidate(parent2);
int chromosomeSize = matingPool.get(0).chromosome.size();
int cuttingPoint1 = r.nextInt(chromosomeSize);
int cuttingPoint2 = r.nextInt(chromosomeSize);
while (cuttingPoint1 == cuttingPoint2) {
cuttingPoint2 = r.nextInt(chromosomeSize);
}
if (cuttingPoint1 > cuttingPoint2) {
int swap;
swap = cuttingPoint1;
cuttingPoint1 = cuttingPoint2;
cuttingPoint2 = swap;
}
ArrayList<Integer> replacement1 = new ArrayList<Integer>();
ArrayList<Integer> replacement2 = new ArrayList<Integer>();
for (int i = 0; i < chromosomeSize; i++) {
replacement1.add(-1);
replacement2.add(-1);
}
ArrayList<Integer> offspring1Vector = new
ArrayList<Integer>();
ArrayList<Integer> offspring2Vector = new
ArrayList<Integer>();
for (int i = 0; i < chromosomeSize; i++) {
offspring1Vector.add(i);
offspring2Vector.add(i);
}
for (int i = cuttingPoint1; i <= cuttingPoint2; i++) {
offspring1Vector.remove(i);
offspring1Vector.add(i, copyParent2.chromosome.get(i));
offspring2Vector.remove(i);
offspring2Vector.add(i, copyParent1.chromosome.get(i));
int index = copyParent2.chromosome.get(i);
replacement1.remove(index);
replacement1.add(index, copyParent1.chromosome.get(i));
index = copyParent1.chromosome.get(i);
replacement2.remove(index);
replacement2.add(index, copyParent2.chromosome.get(i));
}
for (int i = 0; i < chromosomeSize; i++) {
if ((i >= cuttingPoint1) && (i <= cuttingPoint2)) {
continue;
}
int n1 = copyParent1.chromosome.get(i);
int m1 = replacement1.get(n1);
int n2 = copyParent2.chromosome.get(i);
int m2 = replacement2.get(n2);
while (m1 != -1) {
n1 = m1;
m1 = replacement1.get(m1);
}
while (m2 != -1) {
n2 = m2;
m2 = replacement2.get(m2);
}
Integer element = new Integer(offspring1Vector.get(i));
offspring1Vector.remove(element);
offspring1Vector.add(i, n1);
element = new Integer(offspring2Vector.get(i));
offspring2Vector.remove(element);
offspring2Vector.add(i, n2);
}
Candidate offspring1 = new Candidate();
Candidate offspring2 = new Candidate();
offspring1.chromosome = offspring1Vector;
```

```
offspring2.chromosome = offspring2Vector;
offspring1.routeTable = generateRoutes(offspring1);
offspring2.routeTable = generateRoutes(offspring2);
offspring1.fitness = evaluateFitnessCandidate(offspring1);
offspring2.fitness = evaluateFitnessCandidate(offspring2);
nextCandidates.add(offspring1);
nextCandidates.add(offspring2);
} else {
nextCandidates.add(matingPool.get(k % 10));
}
}
int tmpSize = populationSize - nextCandidates.size();
for (int i = 0; i < tmpSize; i++) {
nextCandidates.add(matingPool.get(i));
}
}


private void mutation(ArrayList<Candidate> population) {
Random r = new Random();
int popSize = population.size();
for (int i = 0; i < popSize; i++) {
if (r.nextDouble() > GroceryKart.mutationProbability) {
Candidate temp = inversion(population.remove(i));
population.add(temp);
}
}
}

private Candidate inversion(Candidate candidate) {
int chromosomeSize = candidate.chromosome.size();
Random r = new Random();
int windowSize = (r.nextDouble() > 0.5) ? 2 : 3;
int pos = r.nextInt(chromosomeSize - 5) + 1;
```

```
if (windowSize == 2) {
int temp = candidate.chromosome.remove(pos);
candidate.chromosome.add(pos,
candidate.chromosome.remove(pos + 2));
candidate.chromosome.add(pos + 2, temp);
} else {
int temp = candidate.chromosome.remove(pos);
candidate.chromosome.add(pos,
candidate.chromosome.remove(pos + 3));
candidate.chromosome.add(pos + 3, temp);
temp = candidate.chromosome.remove(pos + 1);
candidate.chromosome.add(pos,
candidate.chromosome.remove(pos + 2));
candidate.chromosome.add(pos + 2, temp);
}
candidate.routeTable = generateRoutes(candidate);
candidate.fitness = evaluateFitnessCandidate(candidate);
return candidate;
}

private void writeToFile() {
try {
FileInputStream finbest = new
FileInputStream("bestsolutions.kart");
ObjectInputStream oisbest = new ObjectInputStream(finbest);

FileOutputStream fout = new
FileOutputStream("solutions.kart");
FileOutputStream foutstores = new
FileOutputStream("stores.kart");
ObjectOutputStream oos = new ObjectOutputStream(fout);
ObjectOutputStream oosstores = new
ObjectOutputStream(foutstores);
ArrayList<Candidate> bestCandidates = new ArrayList<>();
int i=0;
for (Candidate c : currentCandidates) {
i++;
oos.writeObject(c);
bestCandidates.add(c);
if (i == 10) {
break;
}}
for (Store s : stores) {
oosstores.writeObject(s);
}
// Read Best Candidates
int j=0;
while(true) {
j++;
Candidate tmpBestCandidate = (Candidate)
oisbest.readObject();
bestCandidates.add(tmpBestCandidate);
if(j==10) {
break;
}}
Collections.sort(bestCandidates);
FileOutputStream foutbest = new
FileOutputStream("bestsolutions.kart");
ObjectOutputStream oosbest = new
ObjectOutputStream(foutbest);
int k=0;
for (Candidate c : bestCandidates) {
k++;
oosbest.writeObject(c);
if (k == 10) {
break;}}
} catch (IOException | ClassNotFoundException e) {}
}}
```

Chapter 5
TESTING

Once the source code has been developed testing is required to uncover the error before it is implemented. To perform software testing, a series of test cases is designed. Since testing is a complex process hence, to make the process simpler, testing activities are broken into similar activities. Due to this, for a project incremental testing is generally preferred. In this testing process the system is broken in set of subsystems and their subsystems are tested separately before integrating them to form the system for system testing.

Testing means the process of analysing a software item to detect the difference between existing and acquired conditions and to evaluate the feature of the software item.

5.1 BLACK-BOX TESTING

BLACK BOX TESTING, also known as Behavioural Testing, is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.

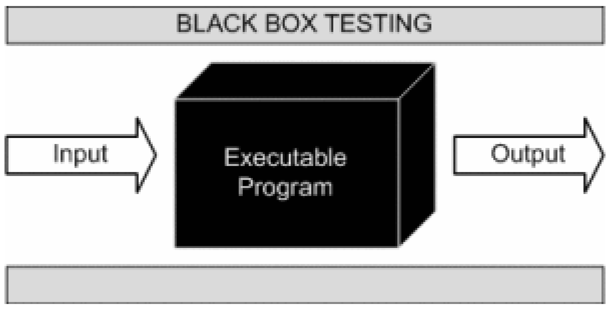


Figure 5.1 Black Box Testing

This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors related to Incorrect or missing functions, Interface errors, Errors in data structures or external database access, Behaviour or performance errors, Initialization and termination errors.

If the testing component is viewed as a **"black box",** the inputs must be given to observe the behaviour (output) of the program. In this case, it makes sense to give both valid and invalid inputs. The observed output is then matched with expected result (from specifications). The advantage of this approach is that the tester need not to worry about the internal structure of the program. However, it is impossible to find all errors using this approach. For instance, if one tried three equilateral triangle test cases for the triangle program, we cannot be sure whether the program will detect all equilateral triangles. This is an astronomical task – but still not exhaustive. To be sure of finding all possible errors, we not only test with valid inputs but all possible inputs.

5.2 WHITE-BOX TESTING

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system. This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees. The table 5.1 compares black-box and white-box testing and differentiates them.

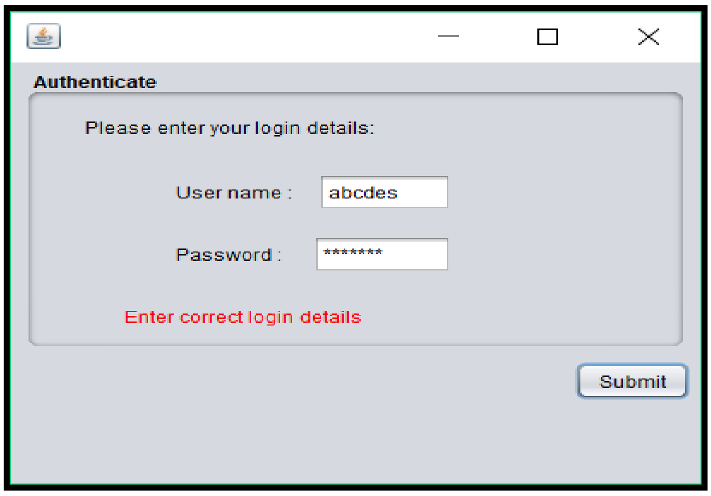
Black-Box Testing	**White-Box Testing**
The internal structure, design, implementation of the module is NOT known to the tester.	The internal design, structure, implementation of module being tested needs to be known to tester.
It is mainly applicable to higher levels of testing like System testing.	It is mainly applicable to lower levels of testing like Unit Testing.
It is generally done by independent software testers.	It is generally done by Software developers.
No programming or implementation knowledge is required.	Programming or Implementation knowledge is required.
The basis for black-box testing are the requirement specifications.	The basis for white-box testing is detailed design.

Table 5.1 Black-Box versus White-Box Testing

5.3 INTEGRATION TESTING

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software engineering in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

LogIn Window: Generates error if the username and password field is entered wrong or kept empty.



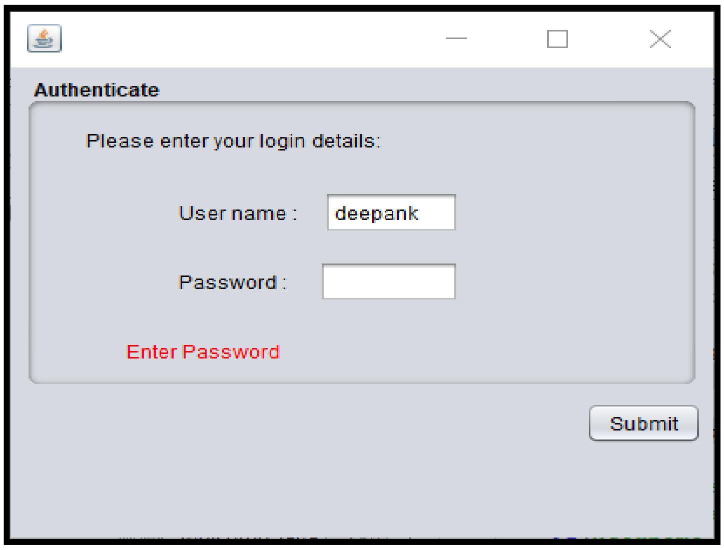
Authenticate

Please enter your login details:

User name : abcdes

Password : *******

Enter correct login details



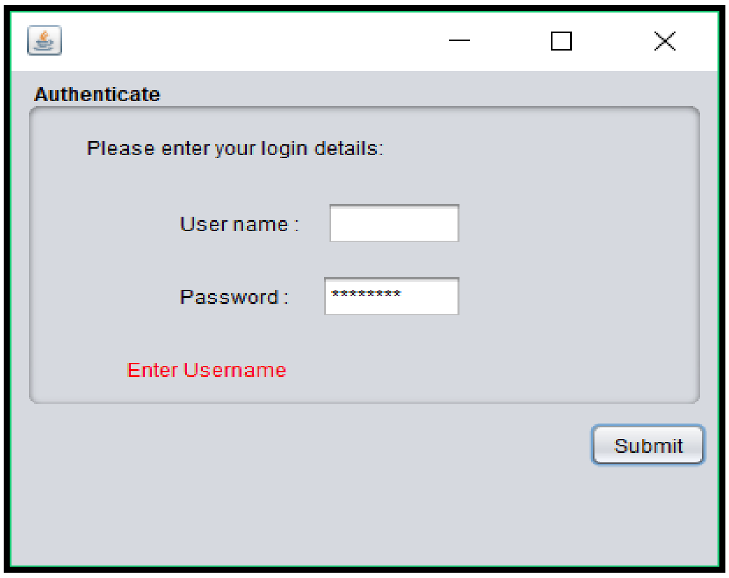
Authenticate

Please enter your login details:

User name : deepank

Password :

Enter Password

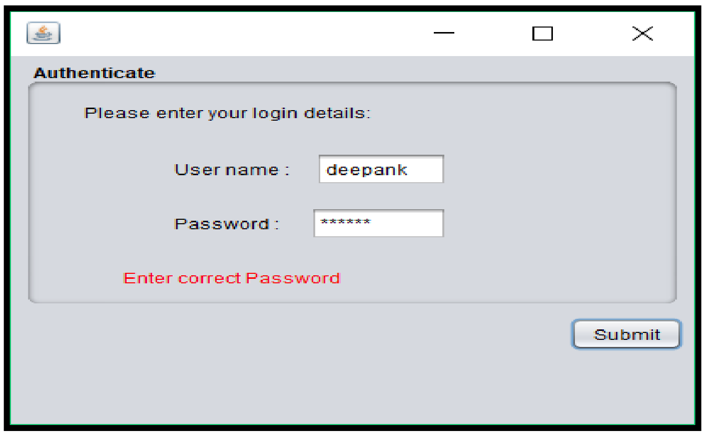

Authenticate

Please enter your login details:

User name :

Password : ********

Enter Username

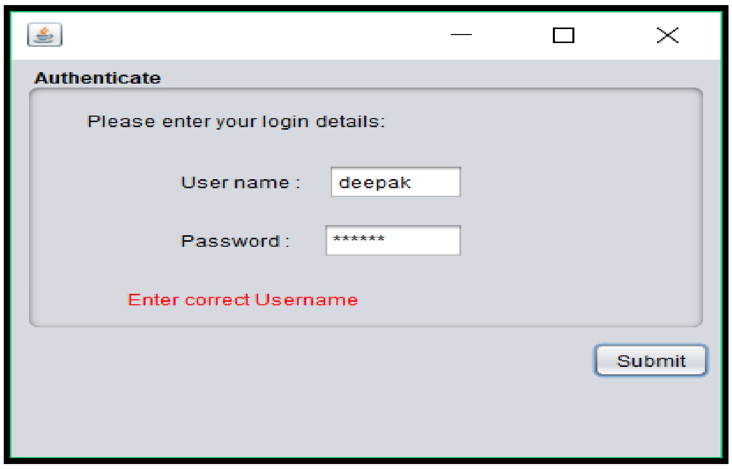

Authenticate

Please enter your login details:

User name : deepank

Password : ******

Enter correct Password


Authenticate

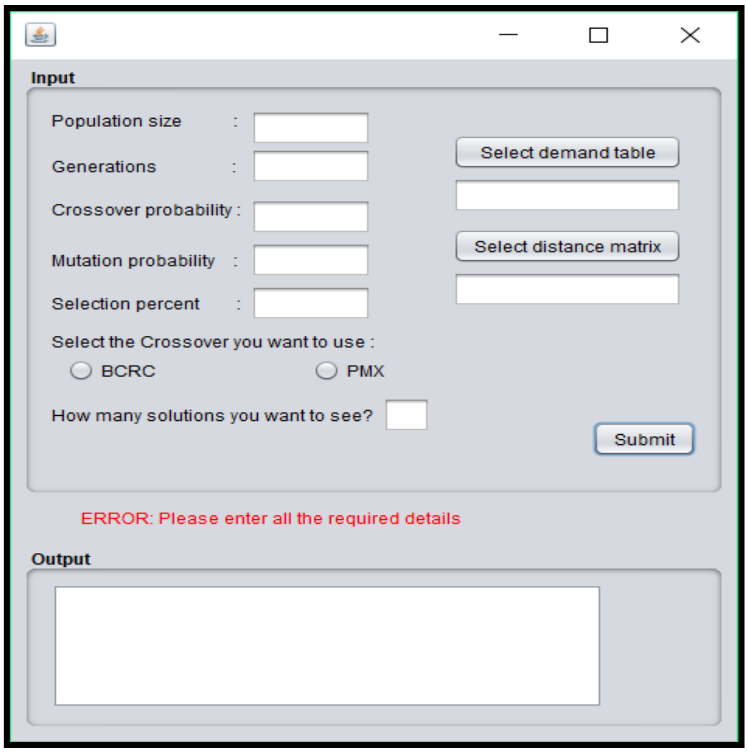
Please enter your login details:

User name : deepak

Password : ******

Enter correct Username

Grocery Kart Form: Generates error if any or all the field in route evaluation window kept empty or filled wrong. All the fields are required here.



Input

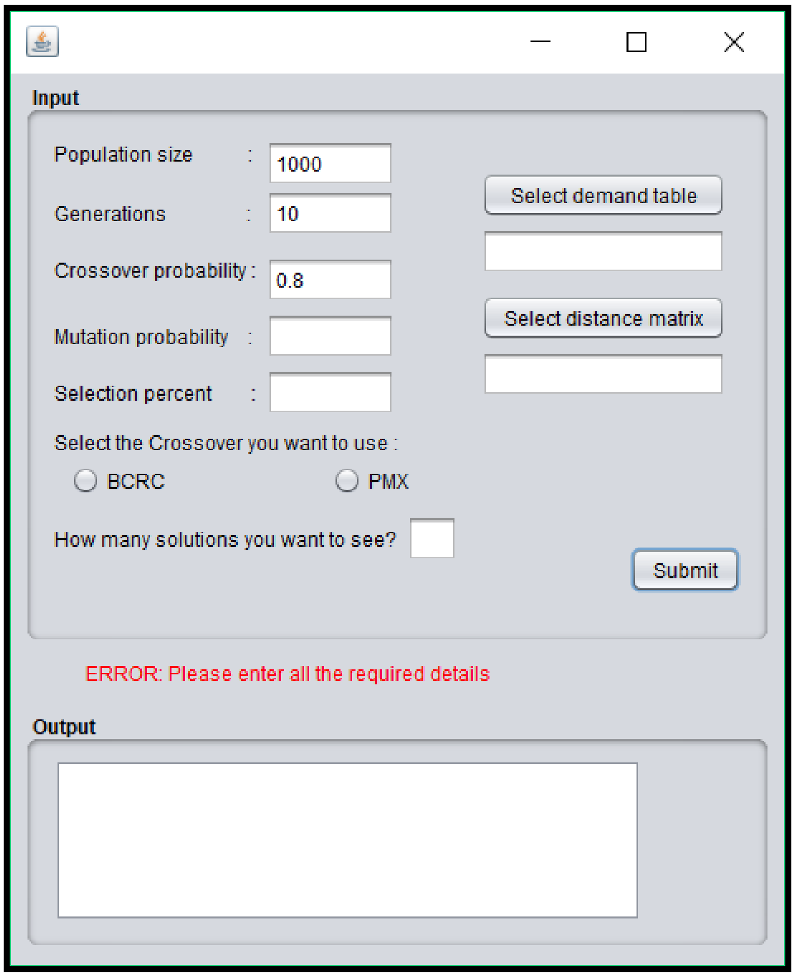
Population size :	
Generations :	
Crossover probability :	
Mutation probability :	
Selection percent :	

Select the Crossover you want to use :
( ) BCRC ( ) PMX

How many solutions you want to see?

ERROR: Please enter all the required details

Output



Input

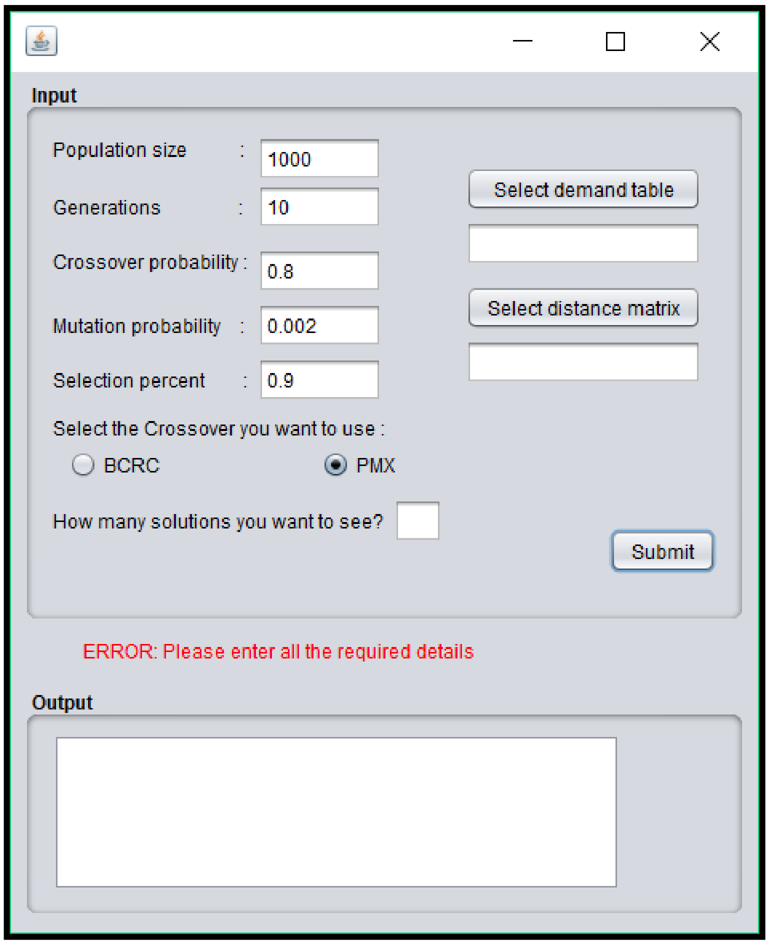
Population size :	1000
Generations :	10
Crossover probability :	0.8
Mutation probability :	
Selection percent :	

Select the Crossover you want to use :
( ) BCRC ( ) PMX

How many solutions you want to see?

ERROR: Please enter all the required details

Output



Input

Population size : 1000

Generations : 10

Crossover probability : 0.8

Mutation probability : 0.002

Selection percent : 0.9

Select the Crossover you want to use :

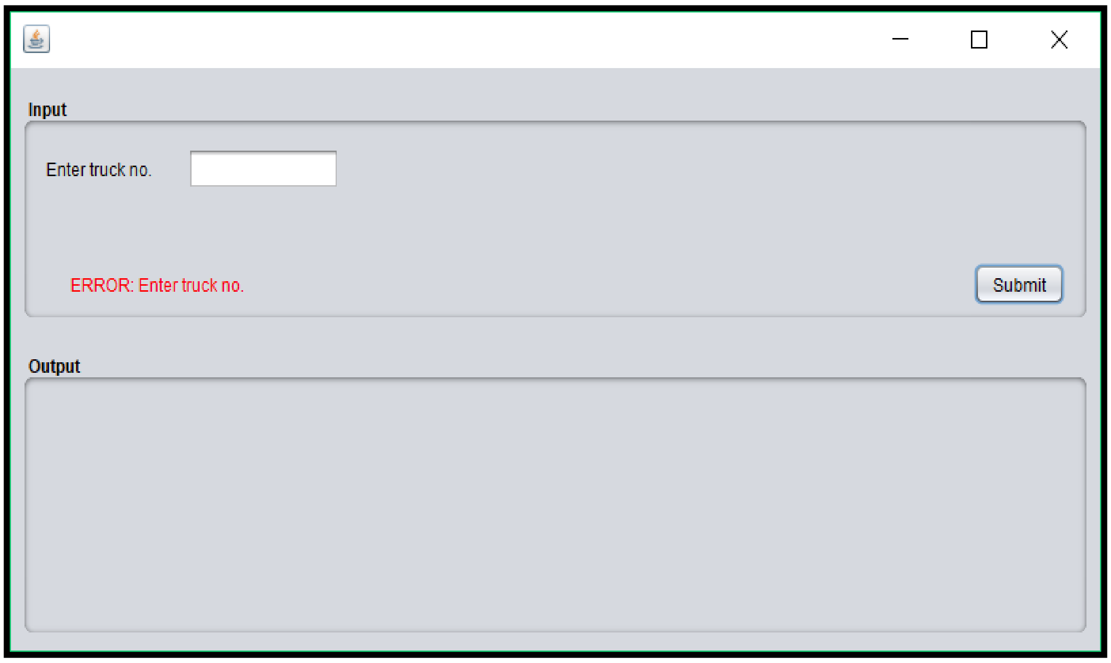
BCRC [✓ PMX]

How many solutions you want to see?

ERROR: Please enter all the required details

Output

Truck Analysis Window: Generates error if truck number is entered wrong or kept empty.

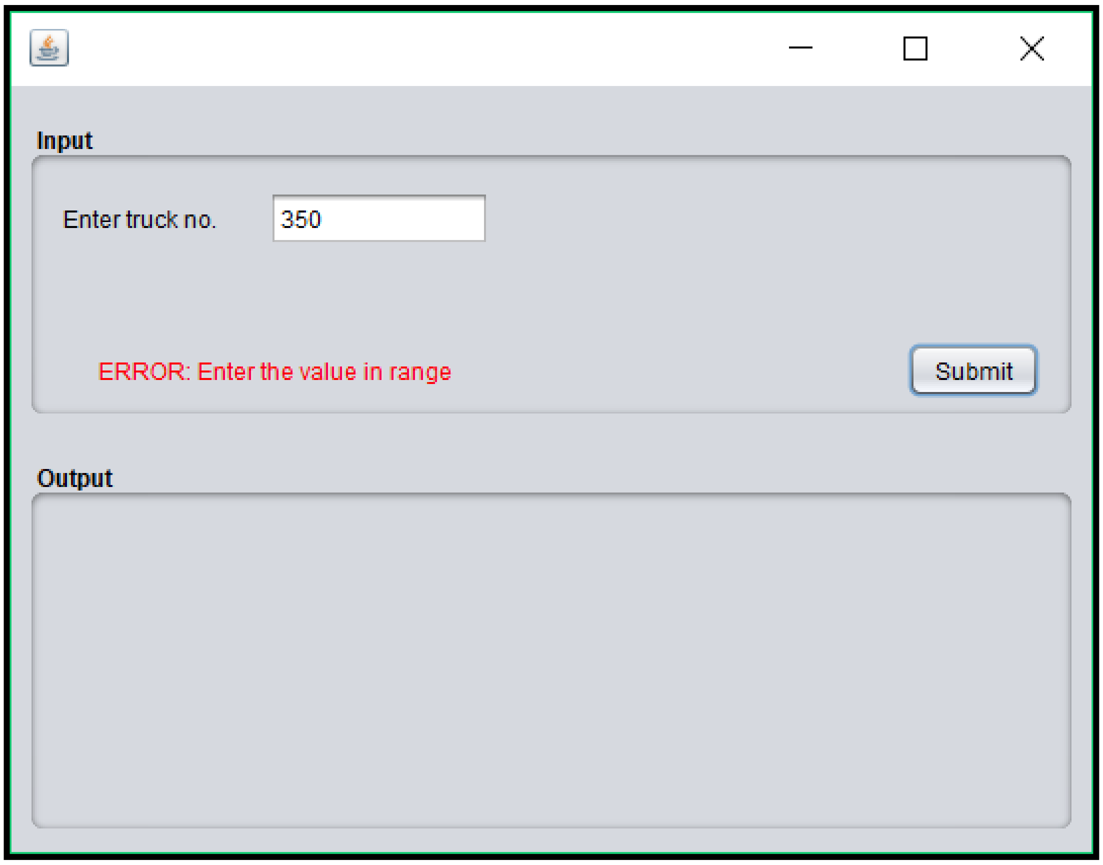


Input

Enter truck no.

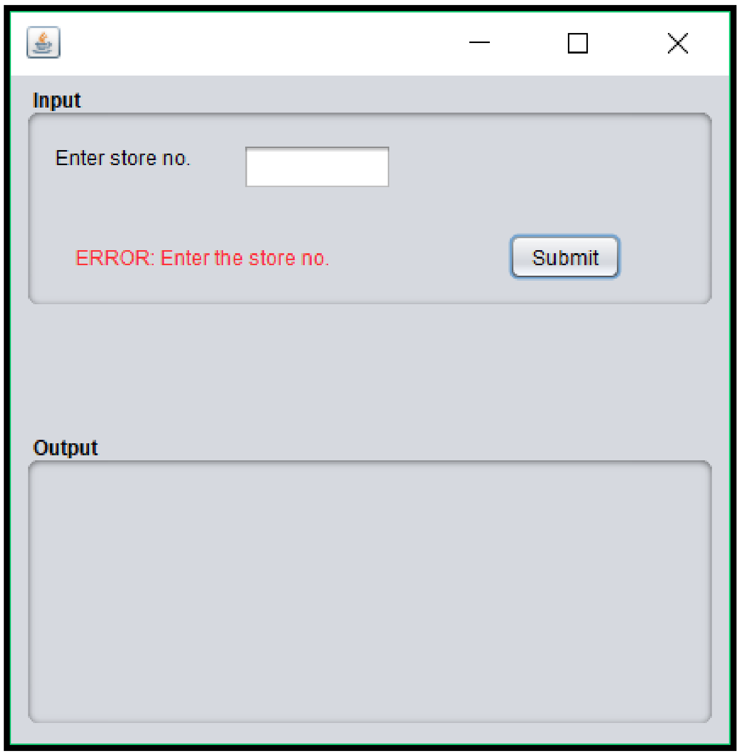
ERROR: Enter truck no.

Output

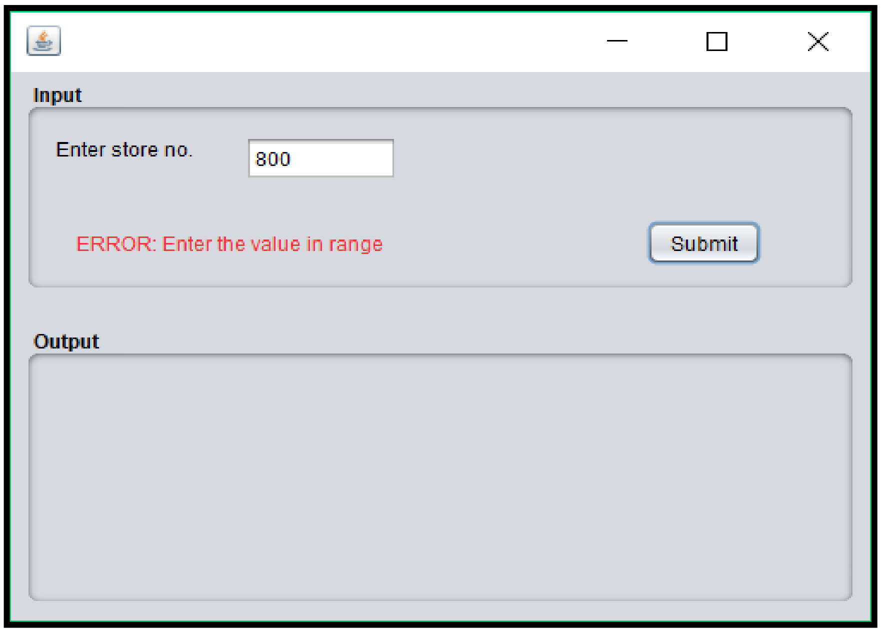


ERROR: Enter the value in range

Store Analysis Window: Generates error if store number is entered wrong or field is kept empty.



ERROR: Enter the store no.



ERROR: Enter the value in range

Chapter 6
RESULT ANALYSIS

The two types of crossover were the basis of analysis and for the comparison of the results.

Table 1 explains BCRC distance covered versus time-taken for the that distance.

BCRC	
Distance	**Time Taken**
799.88	20 min
807.8	12 min
812.1	2 min

Table 7.1. BCRC distance versus time taken

Table 2 shows comparison of implemented crossover i.e. PMX versus BCRC crossover.

PMX		BCRC	
Distance	**Time Taken**	**Distance**	**Time Taken**
818.1	1 min	799.8	20 min
819.4	40 Sec	807.8	12 min
821.5	30 Sec	812.1	2 min

Table 7.2. PMX versus BCRC

From the results shown above which were compiled after the execution of the program we concluded that BCRC gives more optimum distance and takes more time for the execution and the population taken is less. Whereas for PMX gives less optimum distance and takes less time for the execution and the population taken is more.

From these conclusions we cannot judge the best out of the two but can be done in a aspect.

- BCRC is better than PMX in providing more optimum solution.
- PMX is better than BCRC as the execution time is less.

Chapter 7
CONCLUSIONS

Transportation Service System for Grocery kart is a multi-objective optimization problem, which needed a suitable approach to obtain a global optimum solution. So, for this high complexity problem without any known sophisticated solution technique, a genetic approach was well suited. In the optimization process, due to the problem specific encoding, the genetic algorithm can compute and optimize the routing quiet easily as compared to the other deterministic approaches. Genetic algorithm is a search heuristic that mimics the process of natural evolution. They generate solutions to optimization and searching problems using techniques inspired by natural evolution, such as selection, crossover and mutation.

For this problem, initially generated a Random Population to maintain diversity and a Greedy Population to get optimal solutions. Then for the Selection used Tournament Rank selection which further optimised our dataset. Weighted Sum approach and Pareto Ranking (to get the alternative solutions of same nature) are the Fitness Evaluation procedure used.

The BCRC (Best Cost Route Crossover), maintains and improves the candidates' good nature to drive the force of obtaining a better solution by manipulating existing good candidates.

Partially mapped crossover (PMX) aims at keeping the most number of possible positions from the parents. To achieve this goal, a sub string is swapped like in two-point crossover & the values are kept in all other non-conflicting positions. The conflicting positions are replaced by the values which were swapped to the other offspring. With help of the genetic algorithm the system implemented was able to find the optimized routes as well as minimum costs to reach other stores from a specified store or a warehouse.

Chapter 8
FUTURE SCOPE

Genetic Algorithms are of major significance to the development of solutions to multi-objective and high complexity problems. The potential which they offer over existing techniques is enormous. They find application in biogenetics, computer science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields. And the list will continue to grow especially if Genetic Algorithms are combined with other optimization methods.

The method used to solve this Transportation Service System for Grocery Kart problem can be referred to solve:

- Location Allocation Problems
- Real Coded Genetic Algorithm (RCGA) for Integer Linear Programming in Production Transportation problems.
- Network Designing and routing problems.
- Planning of Packet Switched networks.

The further working on optimization can be obtained by:

- We can use roulette wheel selection and rank selection also as mentioned in (Ombuki et.al 2006) in references. The paper described that as compared to tournament selection this selection method was slow and gives a nearly same answer.
- There are various other heuristic approaches mentioned in (Tan et.al 2000) LSD, Tabu search, Simulated Annealing and NSGA-II.
- We can also us PFIH approach for generating a seed for population generation (Tan et.al 2000).
- Insertion mutation is another mutation approach which gives the same result as inversion.
- The best population can be passed as seeds and can improve the running time of Best Cost Route Crossover.
- The best till date solutions can be saved and pass them as initiators to BCRC Crossover and it will further improve the result tremendously.

- There can be a new algorithm which can be invented and will run on updating the solutions only if any minimal changes occur in demand and distance matrix.
- Merging the two heuristic approaches like LSD and MOGA can also improve results but it will take a lot of running time.

REFRENCES

[1] Golden, B., Magnanti, L. and Nguyan, H. (1972). Implementing Vehicle Routing Algorithms. Networks, 7(2), 113-148

[2] Wilson, N., Sussman, J., Wang, H. and Higonnet, B. (1971). Scheduling algorithms for dial-a-ride Systems. Massachusetts Institute of Technology, Cambridge.Technical report USL TR-70-13.

[3] Liebman, J. and Marks, D. (1970). Mathematical analysis of solid waste collections. Public Health Service. Publ No 2065. Ms Depot of Health. Education and Welfare

[4] Levin, A. (1971). Scheduling and Fleet Routing Models for Transportation Systems. Transportation Science 5(3), 232-256.

[5] Solomon, M. (1987). Algorithms for Vehicle Scheduling Routing Problem with Time Windows. Operations Research, 35(2), 354-265

[6] Sariklis, D. and Powell, S. (2000). A Heuristic Method, The Open Vehicle Routing Problem. JORS, 51, 564-573.

[7] K.C.Tan, L.H.Lee, Q.L.Zhu, K.OU,2000. Heuristic methods for vehicle routing problem with time windows, Artificial Intelligence in Engineering, 1-15

[8] Mitsuo Gen and Runwei Cheng,2000. Genetic Algorithms and Engineering Optimization, Wiley Series, 340-400

[9] Laporte, G. and Nobert, Y. (1987). Exact algorithms for the Vehicle Routing Problem. Discrete Mathematics, 31, 147-184.

[10] Fischetti, M., Toth P. and Vigo, D. (1994). A branch and bound algorithm for the capacitated vehicle routing problem on direct graphs. Operations Research, 42, 846-859.Fukasawa, R., Lysgaard, J., de Aragão, M. P., Reis, M., Uchoa, E. and Werneck, R. F. (2004). Robust branch-and-cut-and-price for the capacitated vehicle routing problem.

[11] Baldacci, R., Toth, P. and Vigo, D. (2010). Exact Algorithms For Routing Problems Under Vehicle Capacity Constraints. Annals of Operations Research, 175(1), 213-245

[12] Beasley, J. (1983). Route first-cluster second methods for VRP. Omega, 1, 403-408

[13] Clarke, G. and Wright, J.(1964). Scheduling of Vehicles from a Central Depot a Number of Delivery Points. Operations Research, 12, 568-581.

[14] Mole, R. and S. Jameson, S. (1976). A sequential route-building algorithm employing a generalized saving criterion. Operation Research Quarterly, 11, 503-511

[15] Randy L.Haupt ,2004. Practical Genetic Algorithms -2nd Edition , Wiley Series, 200-250

[16] Beatrice Ombuki,B.J.Ross and Franklin Hanshar,2006. Multi Objective Vehicle Routing Problem with Time Windows, Applied Intelligence, 1-14

[17] S.M.SIVANANDAM, S.N.DEEPA,2008. Introduction to Genetic Algorithms, Springer, 170-200

[18] Mitchell Melanie,1999. An introduction to Genetic Algorithm, MIT Press, 160-165

[19] http://www.mu-sigma.com/muphoria/