Abstract

KARLE, VINAY NAGNATH. Genetic Algorithm for Paratransit Vehicle Routing. (Under the direction of Dr. John R. Stone)

This thesis presents ongoing research on paratransit vehicle routing. Paratransit is defined as dial-a-ride transportation. In our research the emphasis is on specialized transportation for the young, elderly and mobility challenged. The Vehicle Routing Problem with Time Windows (VRPTW) has been of interest to many researchers for at least three decades. In our case, the VRPTW has a precedence condition requiring that a passenger be dropped off at a destination only after being picked up at the origin. This problem is also known as the dial-a-ride problem in public transportation. The focus of the work is on routing algorithms. Given a predefined cluster of passenger requests, routing algorithms described in this thesis try to optimize the vehicle routes to serve them. The thesis introduces the vehicle routing problem and summarizes state-of-the-art approaches (http://www2.ncsu.edu/eos/service/ce/research/stone_res/vnkarle_res/www/research.html). Then it describes a few simple heuristics and a genetic algorithm (GA) developed and implemented in Java ™. The proposed modified crossover operator is based on an adjacency relationship. This modified crossover operator is necessary because the traditional crossover operator results in infeasible solutions because a few nodes may get eliminated while others are repeated. The proposed GA is demonstrated for one vehicle serving 25–passenger

requests. This GA implementation for the VRPTW problem shows promise compared to greedy routing algorithms. The new modified crossover operator can be tested and used in the implementation of genetic algorithm for a general pick-up and delivery problem. Future research directions include adding a GA for combined routing and scheduling problem.

Genetic Algorithm for Paratransit Vehicle Routing

by

Vinay Karle

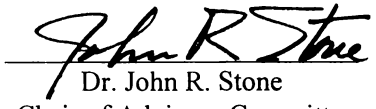
A Thesis Submitted to the Graduate Faculty of
North Carolina State University
In Partial Fulfillment of the Requirement for the Degree of Masters of Science

Department of Civil Engineering

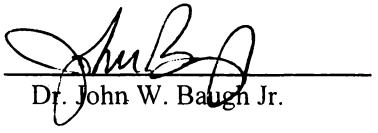
Directed by:
Dr. John R. Stone

Raleigh
May 1999

Approved by:



Dr. John R. Stone
Chair of Advisory Committee



Dr. John W. Baugh Jr.

Dr. Nagui M. Rouphail

Acknowledgments

I would like to express my heartfelt gratitude to my advisor, Dr. John R. Stone, for his instructions, thoughtful advice and support throughout the research. I am also grateful to Dr. John W. Baugh Jr. for his guidance and encouragement. In addition, I would like to express my gratitude to Dr. Nagui M. Rouphail.

I am also indebted to my mother for the courage and support she provided me after my father's sad and untimely demise. If it were not for her encouragement, I would not have been able to continue my studies. I am very thankful to Dr. Stone for all the help he provided during this time. I am also thankful to Dr. Rouphail and Dr. Baugh who both helped me finish my coursework after my return from home. I am very grateful to my friends Sujay Kumar for standing by me during my most trying times, and Praveen Mandava for his encouragement and help.

I appreciate the research funding provided by the Federal Transit Administration (FTA), North Carolina State Department of Transportation (NCDOT), and Winston-Salem Transit Authority. I want to thank the Southeastern Transportation Center (STC) for their additional support in my research work.

Thanks to Dr. Daniel Loughlin for his helping hand and meaningful insight into genetic algorithm. Thanks to Dr. Byungqyu Park for his time and help.

Thanks are due to Dr. David Johnston (Graduate Program coordinator), Dr. Harvey Wahls (former Graduate Program Coordinator), secretaries and staff in Mann Hall - Ms. Edna White, Ms. Barbara Nicholson, Ms. Pat Gay, Ms. Jeanne Phillip and the Office of International Students and Scholar Services at North Carolina State University for looking after my paperwork.

It is my sincere hope that work undertaken as a part of this research will have meaningful contributions. I am proud to be a part of the evolutionary process that constantly drives us to a better tomorrow.

Biography

Vinay Karle was born on January 27, 1975 at Latur in the state of Maharashtra, India. He is the eldest of the two sons and a daughter in the family headed by him after his father.

Mr. Karle received his primary education in different schools in the state of Maharashtra (India) because of the nature of his late father's job. He received his secondary and college education in Chikitsak Samuha Shirolkar High School and D. G. Ruparel College respectively, which are located in Mumbai, India. The author acquired Bachelor of Technology in Civil Engineering at Indian Institute of Technology Bombay in May 1996. In the fall of the same year, he entered the graduate school of Civil Engineering at North Carolina State University. He has worked as a research assistant in the Transportation Systems Engineering group at NCSU with Dr. John R. Stone to pursue a Master of Science in the same field.

Mr. Karle is looking forward to work in software development after completion of his M.S. degree.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

This thesis presents a solution to the paratransit vehicle routing problem with time windows. Paratransit can be defined as dial-a-ride transportation for physically challenged and elderly people. Vehicle routing and scheduling has always been a challenging and intriguing problem for the research community in transportation, operations research, and computer science. Vehicle routing is similar to the travelling salesman problem, a well-known routing problem in which a person must visit a certain number of nodes in an optimum way. The combined routing and scheduling problem is far more complex than the travelling salesman problem. The next section describes the vehicle routing and scheduling problem in general. In order to get an idea of the typical operations of a transit agency, Winston-Salem Transit Authority (WSTA) and its services are described in Section 1.3 of this chapter.

1.1 Vehicle Routing and Scheduling Problem

The conventional "dial-a-ride problem" has the following characteristics:

- size of available fleet – single vehicle or multiple vehicles
- type of available vehicle – homogeneous (same type of vehicle) or heterogeneous
- housing of vehicles - single depot or multiple depot
- nature of demand – recurring subscription, occasional demand-responsive
- location of demand – geographically constrained
- vehicle capacity restriction – small buses
- maximum ride times – about one hour per passenger
- operation – pickups and deliveries of passengers

- costs and objectives - minimum operational cost, minimum size of fleet, maximum passenger service based on convenience

For example, for a typical transit agency like WSTA, the problem is characterized as being a multi-vehicle, homogeneous fleet, single depot problem. In addition there will be some other constraints such as nature and location of demands, task precedence and time windows. Task precedence requires passenger pickup to precede passenger delivery and both tasks must be on the same vehicle. Time window constraints require performing the tasks within specified times. For example, the time window for the trip may be defined as an appointment time at the delivery end of the trip and a pickup time no earlier than one hour before the delivery time. In this way the passenger stays on the bus no longer than one hour (transit agency policy).

The vehicle routing and scheduling problem is "NP-hard" for which no polynomially bounded algorithm has been found (Kakivaya, 1996). This means that as the number of passengers increases, the time to optimally solve for a schedule increases exponentially. Thus, even with the fastest computers, solution time for a typical WSTA schedule approaches infinity.

As explained in Section 1.4, we can guess a few things from the way the current WSTA commercial scheduling software works. It starts with a skeleton schedule, which consists of subscription trips. The demand responsive calls are then inserted into the assumed optimal skeleton schedule, which is believed to have sufficient 'slack' to accommodate the insertion. It also calculates the pick up time for the customer, taking into consideration policies followed by the transit agency (for time window and maximum ride time). The biggest advantage of insertions is that calling back is not necessary as the reservationist can tell the customer the pickup time with time windows and the bus number. If instead all the requests (including subscriptions) are clustered together and optimized at the end of the day preceding the service day, a pick up time cannot be given to the customer at the time when he/she calls. Customers would need to

be contacted after the optimization algorithm clusters and routes all the requests and obtains the new solution.

1.2 Scope and Objectives

The forgoing discussion illustrates a variety of issues regarding the paratransit scheduling and routing problem. Particular issues of interest are generating schedules that require almost no modifications from either a master scheduler or a computer module. These schedules should be convenient to the passenger and efficient with regard to the transit system of vehicles and drivers. The computational resources and computation time should be appropriate to a small paratransit system of 20 to 30 vehicles that service 500 to 1,000 passenger requests per day. To simplify computational complexities, the vehicles are assumed to have the same capacity, and the drivers are assumed to have equal experience. Also demand-responsive requests are not considered, rather all routing and scheduling occurs before the day of service. Thus, this problem is a static dial-a-ride problem. Given this restricted problem statement, several specific research objectives may be defined:

- to develop a heuristics-based computational approach to vehicle routing problem
- to develop heuristics for vehicle routing problem
- to develop a genetic algorithm for routing of a vehicle
- to conceptualize a genetic algorithm for scheduling of vehicles
- to test the routing algorithms
- to use and showcase different developments in Java technology – like programming, graphical user interface using platform independent 'Swing' components, and web-browser enabled documentation

This project provides a unique opportunity to develop a new approach to vehicle routing and scheduling using state of the art programming language. By accomplishing these objectives, we will be able to advance the heuristics, specialized genetic operators, and platform independent programming.

1.3 Winston-Salem Transit Authority

Winston-Salem Transit Authority (WSTA) in Winston-Salem, North Carolina is an attractive case. WSTA provided us the required data and insights into the operations of a typical paratransit agency. The paratransit routing and scheduling problem is essentially a dial-a-ride problem. The explanation of these terms and model development are described in subsequent chapters. In order to get an idea of a paratransit agency and it's operations, this section describes the knowledge we obtained about WSTA.

Winston-Salem is an excellent environment to develop, test and analyze transit innovations. The Winston-Salem Transit Authority (WSTA) is a total public transportation system with over 150 vehicles, 22 of which are small dial-a-ride buses. Its transportation services include fixed-route bus, modified fixed-route for the elderly, downtown circulators, park and shuttle, park and ride, dial-a-ride paratransit, contract paratransit, vanpools, carpool matching and vehicle brokerage. It is an integral unit of the

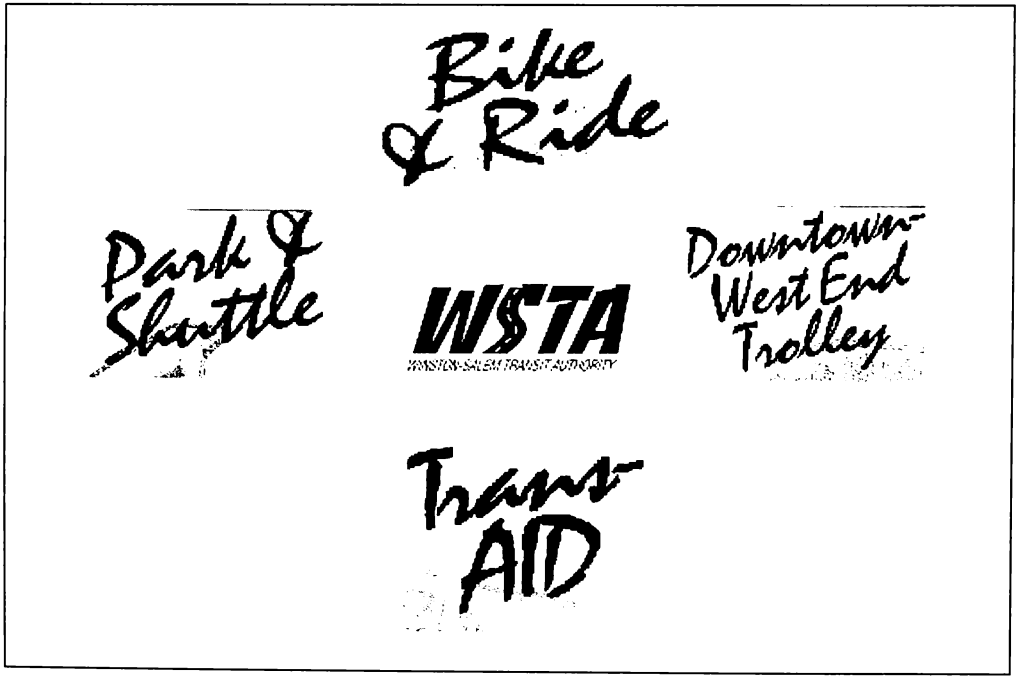


Figure 1.1: Winston-Salem Transit Authority

Winston-Salem Department of Transportation, which controls community transportation planning and construction, traffic operations and parking. As one of the nation's 15 demonstration sites for Advanced Public Transportation Systems (APTS), WSTA has demonstrated the capability to procure, install, manage and operate APTS. Trans-AID is the paratransit, dial-a-ride service for WSTA. It carries passengers in Winston-Salem and Forsyth County, a 400 square mile area with 550,000 people. Trans-AID passengers are eligible for transportation under the provisions of the Americans with Disabilities Act (ADA), Medicaid, and other social programs.

Before WSTA installed a computer-aided dispatch and scheduling system (CADS) in August 1994, almost 80% of the 118,050 annual demand-responsive Trans-AID passenger trips were recurring subscriptions needing no telephone reservation. The subscription trips provided a rarely changing "skeleton" schedule that a mini-computer prepared. The remaining 20 % were non-subscription passengers who reserved trips at least 24 hours in advance. Trans-AID provided virtually no same-day scheduling and dispatching. Drivers received a daily manifest each morning, and they received cancellations by voice radio during the day's operations. Before CADS the Trans-AID client base was 94% urban; very few rural passengers received transportation service. Additional rural funding during the evaluation period allowed Trans-AID to expand its rural client base and service significantly. By more efficiently scheduling passengers on vehicles and by better utilizing vehicle capacity, CADS allowed Trans-AID to serve the new rural passengers without purchasing any new vehicles.

After WSTA implemented CADS, Trans-AID passengers call a reservationist, who enters the relevant passenger information into a database. A commercial scheduling algorithm determines several alternatives for serving the caller's trip including estimated pick-up and arrival times for each alternative. The algorithm also identifies the "best" alternative on the basis of the existing schedule, that is a pre-CADS skeleton schedule of subscription trips. Ideally the skeleton schedule is optimized. However in practice it may not be the case as people move, cancel trips etc. The reservationist then negotiates with

the caller to determine which alternative is "best" with respect to the passenger. Travel requests can be handled in advance or real-time. Either way the new trip is inserted into the existing optimized skeleton schedule of subscription trips.

About 19 small buses equipped with voice radio carry the majority of the passengers. Drivers pick up and drop off passengers according to the schedules printed on conventional paper manifests. Drivers receive passenger cancellations or additions to the schedules from the WSTA dispatcher by voice radio. As each trip is completed, the drivers write down the mileage and other data concerning the trip for subsequent billing. Passengers pay no fares; agency contracts pay for all trips.

Three experimental vehicles carry mobile data terminals (MDTs), automated vehicle locators (AVL), and smart card readers. For these vehicles the assigned trip is transmitted digitally via radio to the vehicle, where the information is displayed on the MDT. The driver communicates back to the dispatcher through the MDT keypad acknowledging the request. When a passenger is picked-up or dropped-off, a smart card identifies the passenger and time stamps the trip. The AVL identifies the pick-up or drop-off location and the odometer marks the mileage. AVL tracks vehicle movements each time the MDT is used or every three minutes if the MDT has not been used. The scheduling algorithm uses the updated vehicle location for inserting demand-response trips (Stone, 1995).

More information about different services provided by WSTA is available on the World Wide Web (Appendix E). All the references to WSTA from here onwards refer to their paratransit. Also 'routing and scheduling' refers to 'paratransit routing and scheduling'.

1.4 WSTA Routing and Scheduling

With the assistance of a commercial scheduling program, schedulers prepare a day's schedule up to two weeks before it is performed. As new requests and cancellations are

telephoned to them from passengers, they make modifications usually several days before the actual service. They insert new requests into the computerized skeleton subscription schedule and make cancellations. The changes are guided by preferences from the passengers and suggestions from the commercial scheduling program. There is no re-optimization of the combination of the original skeleton schedule and the new insertions and cancellations. This process depends on a relatively wide pickup time windows and slack time in the schedule. The result is a fairly rigid schedule with the potential for system inefficiencies and passenger inconvenience. The process allows schedulers to promise a pickup time within a specified time window (usually ± 20 minutes of that time promised). If the schedule were re-optimized with the changes, significant computer time would be required and the schedulers would have to call the passengers back with the final pickup time.

In an attempt to improve system efficiency and passenger convenience without changing pickup time, a master scheduler may make final adjustments to the schedule the day before service. Such changes may include the following:

- Reallocating a passenger to a different route in order to save an extra trip to that area. For example customers living close to each other and having almost the same pick-up times might be on different routes because they called at different times to request the service. If they can be serviced together, the master scheduler would do the appropriate change.
- Adding a new customer to the system. The WSTA schedule for a particular day "opens" for scheduling two weeks in advance. If a new customer is added to the system during that time period, current scheduling software does not assign any route to this passenger.
- Adjusting travel time for long trips
- Swapping the drivers on different routes based on their familiarity with the area
- Replacing a broken down van.

The master scheduler can significantly improve a day's schedule based on experience and knowledge of the passengers, service area, and vehicles. Although some software products have automatic routing options to perform this task, many schedulers do not trust these options to give good schedules and instead rely on their own judgement. It is quite evident from the discussion above that the master scheduler plays an important role in day-to-day scheduling. Also, the skeleton subscription routes and schedules which are originally optimized may become 'stale' as passengers cancel trips, move to a new area, cancel the subscription, apply for subscription service etc. Hence in this research, efforts are made to produce schedules that are more close to optimality. Closeness to 'optimality' is again a qualitative term, at best we can make it a relative term and our efforts are focused in that direction. The explanation of the problem, terms and definitions, and solution approach are discussed later on in this thesis.

1.5 ITS Perspective of the Research

It is important to see how and where this research effort fits into the context of ITS. ITS is "the application of advanced information processing and communications, sensing, and control technologies to transportation". It is generally suggested that ITS applications may be considered in six interdependent system areas, three focused on technology and three on application (IVHS Primer, 1998).

Technology oriented:

- ATMS (Advanced Traffic Management Systems)
- ATIS (Advanced Traveler Information Systems)
- AVCS (Advanced Vehicle Control Systems)

Application oriented:

- APTS (Advanced Public Transportation Systems)
- CVO (Commercial Vehicle Operations)
- ARTS (Advanced Rural Transportation Systems)

Advanced public transportation systems are integral components of ITS. WSTA routing and scheduling problem falls under APTS (Figure 1.2). We are trying to 'optimize' the competing objectives- maximize the customer convenience and minimize the cost to the agency. Success in these system objectives will ultimately reflect on the overall progress of ITS.

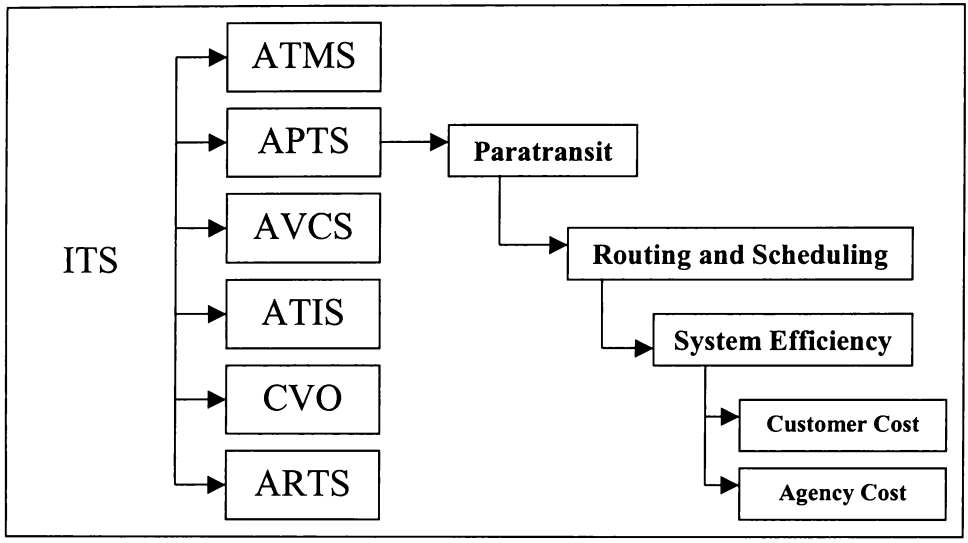


Figure 1.2: ITS Perspective

1.6 Organization of the Report

Chapter 1 introduced the general dial-a-ride problem based on our understanding of WSTA's operations. Chapter 2 presents some of the research efforts in the area of vehicle routing and scheduling problem and compares and contrasts a few heuristic methods. Chapter 3 develops a genetic algorithm with modified crossover operator for the multiobjective vehicle routing problem. It also describes a Nearest Neighbor routing algorithm and two heuristics algorithms. Chapter 4 presents results and performance of the genetic algorithm. In the final chapter, contributions and further research potential are discussed.

Chapter 2
Literature Review

The following sections present alternative approaches found in the literature to the dial-a-ride problem and their advantages and disadvantages. This information will help justify the genetic algorithm approach developed in this research.

2.1 Approaches to the Dial-A-Ride Problem

As discussed in Chapter 1 the dial-a-ride problem (DARP) is complicated and requires an innovative solution. Figure 2.1 shows different approaches to the routing and scheduling problem. The circular nature of the diagram indicates that combinations of these approaches are also useful. These approaches as discussed by Bott and Ballou (1986) are summarized next.

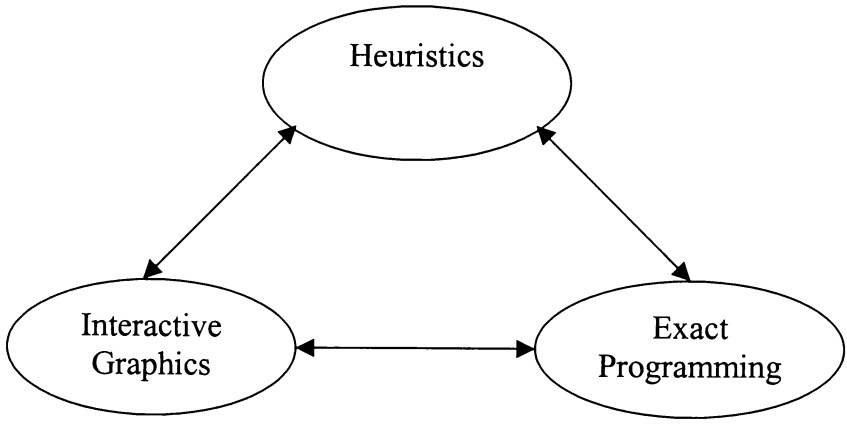


Figure 2.1: Solution Strategies for the Vehicle Routing and Scheduling Problem

Heuristic Approaches

As DARP is combinatorial in nature, heuristic techniques dominate the solution procedures. Researchers who have published heuristics approaches include Bodin, Golden and Assad (1983), Evans and Norback (1984), and Barker (1992). Four strategies in this category are:

- Cluster first, route-second procedures. First group or cluster customers and then determine the feasible routes for each group as a second step.
- Route first, cluster-second procedures. These methods build a large (usually infeasible) route first. Then the large route is partitioned into a number of smaller feasible routes.
- Savings or insertions procedures. These methods build a solution (possibly an infeasible one) by calculating the savings generated by a new configuration, or they determine the least expensive insertion of a customer into a route. Some of the earliest vehicle routing and scheduling heuristics were based upon this logic, and many variations have been studied.
- Exchange, or improvement, procedures. These methods maintain feasibility in the solution while modifying the routes step by step to reduce costs.

By using heuristic methods we settle for approximate solutions. In a worst case they can be far from the optimal solution (Solomon, 1986). Sometimes heuristics can be too general to solve a specific problem. One may need to add location-specific heuristics in order to speed up the solution procedure. Generally heuristic algorithms do not have a sound mathematical background, rather they have an experiential or intuitive basis. Heuristic algorithms include simulated annealing (Baugh, Kakivaya and Stone, 1995), tabu search (Gendreau, Hertz, and Laporte, 1994), and genetic algorithms (Uchimura and Sakaguchi, 1995, and Thangiah and Nygard, 1992). An introduction to genetic algorithms appears in Section 3.1. Heuristic methods are widely used for the dial-a-ride problem.

Exact or Optimal Approaches

Exact or optimal solution procedures are always desirable from the standpoint of unique solutions. Unfortunately vehicle routing and scheduling problems cannot be solved for exact solutions in real time even by the fastest available computers. The main difficulty with all exact procedures is the huge number of constraints and variables needed to represent even the basic vehicle routing and scheduling problem and their adverse effect on computation time and computer storage space. Recognizing such limitations no optimal approaches are used in this research (Kakivaya, 1996).

Interactive Approaches

The third solution strategy is the most simplistic, but in some respects it may be the most powerful of the strategies (Brill, Flach, Hopkins and Ranjithan, 1990). Interactive methods involve either a simulation, cost calculator approach, or some type of graphics capability to aid the decision-maker. Interactive approaches are merely aids to the decision-maker who still must use some intuitive methods to solve the problem. In the transit field schedulers use interactive approaches when they intuitively route buses to passengers using geographic displays of passenger origins and real-time bus locations. Again, the difficulty with this type of approach is that it is not guaranteed to produce optimal or even good schedules. This difficulty increases as the number of passengers and vehicles increase.

Combination Approaches

These strategies for solving the routing and scheduling problem combine the above three approaches. Researchers also have tried combinations of different heuristics for the routing and scheduling problem (Russell, 1995; Osman and Kelly, 1996).

Exact mathematical formulation of the problem and few heuristic approaches are described in detail in Section 2.2 and Section 2.3 respectively.

2.2 Exact Mathematical Formulation of the Problem

As described in Section 2.1, the exact or optimal approaches are not used to solve the combinatorial optimization problem. Kakivaya (1996) gave the exact formulation of the static dial-a-ride problem in his Ph.D. dissertation. The static dial-a-ride problem is characterized by the following:

- Routing constraint: each customer's origin and destination need to be visited exactly once
- Assignment constraints: each customer must be assigned to a trip
- Timing constraints: each customer's origin and destination have prescribed time windows
- Precedence constraints: a passenger be dropped off at a destination only after being picked up at the origin
- Capacity constraint: the capacity of the vehicle cannot be exceeded at any time
- Competing objectives: various objectives should be minimized, including the distance traveled by all vehicles, customer inconvenience, and the number of vehicles used

An infeasible route can be the one in which the precedence condition is violated, or a pick-up or a drop-off for a customer is missing or repeated, or capacity constraint is violated etc. In case of the hard time windows, violation of time windows may be considered as an infeasible solution.

Note that in reality the constraints can be very complex. For example, for a paratransit vehicle the capacity component may have additional dimensions such as wheel-chair capacity, etc. Timing constraints may vary based on the trip type and may not be well defined. For example the appointment time window would be followed more strictly than a home return trip.

A mathematical model based on following parameters was formulated by Kakivaya:

N number of customers

C capacity of the vehicle

V maximum number of trips

S maximum time violation permitted at a node

t_{ij} travel time between nodes i and j

ET_i earliest pick-up or drop-off time at node i

LT_i latest pick-up or drop-off time at node i

T a number larger than the latest drop-off time

where a node represents either an origin or a destination

Primary decision variables in the model are as follows:

x_{ij}^k 1, if travel from node i to node j (denoted as arc ij) is on trip k

0, otherwise

v number of trips

y_i departure time at node i

s_i amount of time window violation at node i

c_i number of customers in the vehicle at node i

The following arcs, x_{ij}^k, are infeasible and hence can be eliminated from the model when:

$i = j$

$i = j + N$

$i \leq N$ and $j = 0$

$i = 0$ and $j > N$

$ET_j > LT_i + t_{ij} + 2S$

$LT_j < ET_i + t_{ij} - 2S$

The objective function contains three objectives – cumulative travel time, degree of violation of time windows, and number of trips. The objective function can be stated as follows:

$$\text{Minimize } z_1 = \sum_{k=1}^{V}\sum_{i=0}^{2N}\sum_{j=0}^{2N} t_{ij}x_{ij}^{k}\ , z_2 = \sum_{i=1}^{2N} s_i\ , z_3 = v$$

In a cluster-first, route-second strategy, each cluster is to be routed optimally. Let us assume that there are 'M' customers in a cluster ($M \leq N$). Then the objective function for a single cluster can be stated as:

$$\text{Minimize } w_1 = \sum_{i=0}^{2M}\sum_{j=0}^{2M} t_{ij}x_{ij}\ , \quad w_2 = \sum_{i=1}^{2M} s_i$$

The objective function can also be modified to take into consideration the excess ride time (difference between actual ride time and minimum ride time).

$$\text{Minimize } w_1 = \sum_{i=0}^{2M}\sum_{j=0}^{2M} t_{ij}x_{ij}\ , \quad w_2 = \sum_{i=1}^{2M} s_i\ , \quad w_3 = \sum_{i=1}^{2M} (ert)_i$$

where $(ert)_i$ = actual ride time from pick-up to drop-off – minimum ride time

$(ert)_i = 0$, if i^{th} node is a pick-up node

In the overall objective function, a term to optimize hours of operation can also be included. A 'tour' or a 'route' may not operate for more than eight hours

$$\text{Minimize } z_1 = \sum_{k=1}^{V}\sum_{i=0}^{2N}\sum_{j=0}^{2N} t_{ij}x_{ij}^{k}\ , z_2 = \sum_{i=1}^{2N} s_i\ , z_3 = v, \qquad z_4 = \sum_{k=1}^{V}\sum_{i=0}^{2N} (ert)_i\ ,$$

$$z_5 = \sum_{k=1}^{v} (hop_k - 8hrs)$$

where hop_k = Hours of operation on 'route' (or 'tour') k

2.3 Heuristic Approaches

Many researchers in the past have used a local search procedure, a simulated annealing procedure, tabu search and genetic algorithms for solving the dial-a-ride problem (DARP). As stated earlier DARP is NP-hard, hence exact approaches, in general are not practicable. Solomon (1986) has described heuristics –savings heuristic, time oriented nearest neighbor heuristic, insertion heuristic, time oriented sweep heuristic, giant-tour heuristic - and their worst case performance. Tables 2.2, 2.3, 2,4, 2.5 summarize some heuristics that can be used for solving complex combinatorial problems. Table 2.1 gives relative advantages and disadvantages of different heuristics (Osman and Kelly, 1996; Baugh, Kakivaya, and Stone, 1995; Gendreau, Hertz, and Laporte, 1994; and Thangiah and Nygard, 1992). Table 2.1 lists and compares heuristic approaches. Tables 2.2, 2.3, 2.4, and 2.5 describe the local search descent, genetic algorithm, simulated annealing, and tabu search procedures.

2.4 Summary

As discussed earlier, there are different approaches to solve the dial-a-ride problem. It is a combinatorial optimization problem with multiple objectives. Due to the difficulty of this problem class, it is very improbable that an optimal solution could be found in polynomial time. Hence, heuristics which find an approximate optimal solution in polynomial time in 'n' (where n is the size of the problem) seem to offer attractive alternatives.

Genetic algorithms have a strong intuitive base even though they lack strong mathematical backgrounds. Genetic algorithms have been used for vehicle routing problem (Uchimura and Sakaguchi, 1995). They have also been used in combined vehicle routing and scheduling problem like school bus scheduling (Thangiah and Nygard, 1992). In this research we use GAs for vehicle routing problem with time windows and precedence condition. The cluster first – route second approach is taken to solve the

problem. It is not only important to come up with a good clustering strategy but also a good routing algorithm as efficiency and effectiveness of various routes will in turn represent overall efficiency and effectiveness of a day's schedule. GA is a stochastic random search technique. The search is guided by the 'fitness function' which is problem dependent. Chapter 3 describes in detail how GA works and it's implementation for the vehicle routing problem. It also describes the conceptual representation for the scheduling part of the problem, which essentially is a clustering algorithm.

Table 2.1: Comparison of Different Heuristics

Heuristic	Advantages	Disadvantages
Local Search Descent Procedure	(1) Simple and quick, (2) Works well if the "terrain" (solution space) is not smooth, (3) Some disadvantages can be overcome by using probabilistic guidance strategies.	(1) Can be very far from optimality as they perform blind search and solutions which reduce objective function value are accepted sequentially, (2) Do not use information gathered during the execution of the algorithm, (3) Depend heavily on initial solution and neighborhood generation mechanism.
Neural Networks (NN)	(1) Best for the problems that use past data e.g. pattern recognition, incident detection, prediction, and classification.	(1) Not suited for combinatorial optimization problems.
Simulated Annealing (SA)	(1) Can deal with arbitrary systems and cost functions, (2) It is relatively easy to code and generally gives a "good" solution, (3) Statically guarantees finding an optimal solution	(1) Appropriate cooling schedule is important for effective performance of the algorithm
Tabu Search	(1) Can guide a subordinate heuristic to continue the search beyond local minimum as tabu search chooses best move at each iteration even if results in degradation of the solution	(1) In order to be more efficient, it may need to be used with other heuristics
Genetic Algorithms	(1) Search is based on solution generation mechanism rather than attributes of a single solution by move-generation mechanism of local search methods, (2) Can be a powerful tool if used appropriately with other algorithms	(1) Initial solution which may be generated at random may not be of good quality, (2) The binary encoding of solutions may not be suitable for all problems, (3) Special repair schemes, crossover and mutation operators may be required

Table 2.2: A Local Search Descent Procedure

Step 1: Get an initial solution S and compute its objective $C(S)$.

Step 2: While there is an untested neighbor $S' \in \overline{N(S)}$, execute the following:

(a) Generate sequentially a trial $S' \in \overline{N(S)}$ and compute.

(b) If $C(S') < C(S)$ then, S' replaces S as a current solution. Otherwise, retain S and step 2 is repeated.

Step 3: Terminate the search and return S as the local optimal solution.

Table 2.3: A Simple Genetic Algorithm

Step 1: Generate an initial population of strings (genotypes).
Step 2: Assign a fitness value to each string in the population.
Step 3: Pick a pair of strings (parents) for breeding.
Step 4: Put offspring produced in a temporary population (mating pool)
Step 5: If the temporary population is not full, then go to step 3.
Step 6: Replace the current population with the temporary population and a portion of the current population.
Step 7: If the termination criteria are not met, go to step 2.

Table 2.4: Tabu Search Procedure

Step 1: Get an initial solution S, and initialize the short-term memory structures

Step 2: Select the best admissible solution, $S_{(best)} \in \overline{N(S)} \subseteq N(S)$; ($S_{best}$ is the best of all $S' \in \overline{N(S)} : S'$ is not tabu-active, or passed an aspiration test)

Step 3: Update the current solution $S = S_{best}$,

Step 4: Repeat Step 2 and Step 3 until a stopping criterion is satisfied.

Table 2.5: Simulated Annealing Procedure

Step 1: Generate an initial random or heuristic solution S'
Set an initial temperature T and a cooling schedule and simulated annealing procedure parameters.
Step 2: Choose randomly $S' \in N(S)$ and compute $\Delta = C(S') - C(S)$
Step 3: If
(i) S' is better than S ($\Delta < 0$), or
(ii) S' is worse than S but "accepted" by the randomization process at the present temperature T, i.e. $e^{\left(\frac{-\Delta}{\theta}\right)}$
Then replace S by S'
Else Retain the current solution S
Step 4: Update the temperature T depending on a set of rules, including:
(i) The cooling schedule is used
(ii) Whether an improvement was obtained in step 3 above,
(iii) Whether the neighborhood $N(S)$ has been completely searched
Step 5: If a "stopping test" is successful stop, else go to Step 2.

Chapter 3
Methodology

As described in Chapter 2, we looked at few methodologies. Genetic algorithms (GA) try to imitate the natural process of "selection", "crossover" and "mutation" to develop a solution. The search space for the routing and scheduling problem is very large. If there are '*n*' customers to be serviced by a route, we will have '*(2n)!*' permutations to deal with before finding an optimal route. Of course lots of permutations will be invalid, as they have to satisfy the precedence condition that a drop-off cannot precede a pick-up for a passenger. Appendix B presents a proof based on mathematical induction to give the exact number of 'brute force' computations required for vehicle routing problem.

Transit authorities like WSTA have to deal with many buses and passengers. Ideally the passengers would be assigned to a bus and then routed optimally. This chapter presents genetic algorithm and heuristics to solve routing problem. In Generic Algorithm, the initial "gene pool" consists of a number of solutions (unlike in Simulated Annealing where we have only one solution to deal with). As the solutions proceed through different "generations" their quality improves. In general a search space (in other words different solutions) can be imagined to be like a terrain with lots of valleys and mountains. Our aim is to reach the highest point, but we do not have any idea of the terrain (almost like a blind-folded person). Finally we settle for a relatively high point, as we do not know the absolute high point (or exact solution). Before proceeding further with the algorithm, Section 3.1 in short describes genetic algorithms and how they work.

3.1 Genetic Algorithms

A popular definition of a genetic algorithm is that it is a program that "evolves" in ways that resemble natural selection. It can thus solve complex problems by imitating the natural process of selection and reproduction.

Genetic algorithms make it possible to explore a far greater range of potential solutions to a problem than do conventional programs. Most organisms evolve by means of two primary processes - natural selection and reproduction. Natural selection determines which members of a population survive to reproduce, while reproduction ensures mixing and recombination among the genes of their offspring. Selection for crossover is simple. If an organism fails some test of fitness, such as recognizing a predator and fleeing, then it dies. Similarly poorly performing solutions are weeded out (Wyner, 1991).

The solution space can be viewed as a terrain. Valleys mark the location of strings that encode poor solutions and the highest point in the terrain corresponds to the best possible string. However, a genetic algorithm solution is acceptable if it is reasonably high (sub-optimal), the highest is not necessary. This description of the search terrain is valid for both maximization or minimization problems, which is referred to as optimization problems. Programming a computer to find a GA solution is challenging. The program maintains its own gene pool of solutions, and the search for the best one requires answers to the following questions:

- What do potential solutions look like?
- How do you initialize the gene pool?
- How do the solutions vary?
- What range of possible solutions might be in the pool?
- What is the difference between good and bad solutions?
- How is an acceptable solution quantified?
- How will breeding take place, and how will solutions crossbreed with each other?
- What is the mutation strategy?
- How does the GA guard against infeasibilities?
- How does the GA treat infeasibilities if they cannot be avoided?

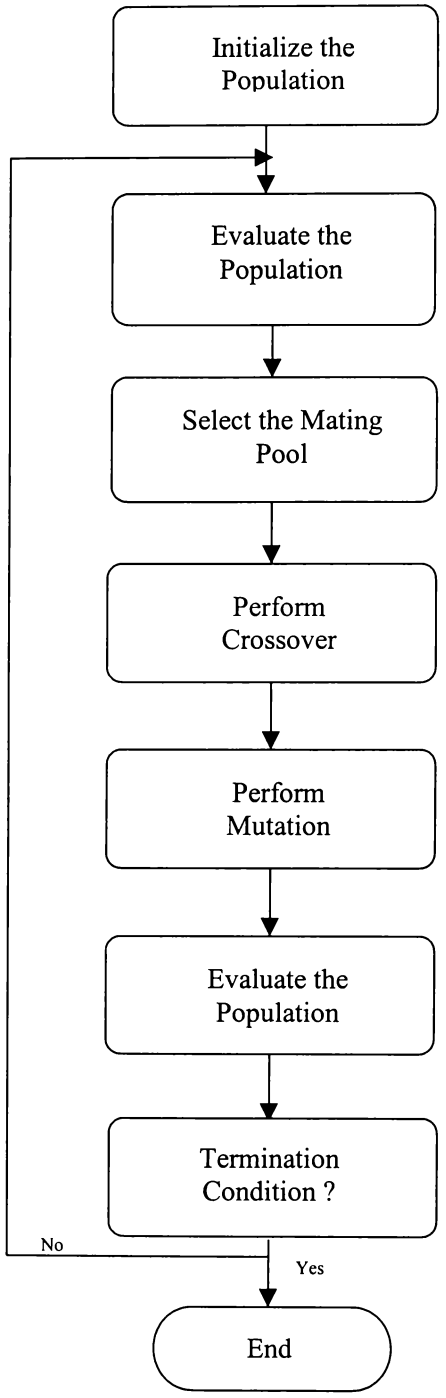


Figure 3.1: Flowchart of a Simple GA

The reproduction step will consist of creating a new generation in which more "fit" solutions are more likely to be represented than the unfit ones. The entire process of developing GA solutions is by nature empirical. One may need to tune the fitness function several times before success. Mutation might help move the process out of a niche. Even though fitness functions control the reproduction, crossover and mutation quickly destroy whatever coherence there is. GAs are part chance, part intelligent guidance, and part eager experimentation. Their advantage is that the endless recombination and mutation provide a certain amount of robustness and flexibility that are not part of many algorithms. A disadvantage is that randonmess gives the process an undeserved reputation of "black magic". Figure 3.1 shows a typical flowchart of a simple GA.

3.2 Vehicle Routing Problem – An Example

It is quite evident from the earlier discussions that the vehicle routing problem with time windows and precedence condition is a NP problem. So far as the vehicle does not violate the traffic regulations, the vehicle may pass the same tour point. On the other hand, in the travelling salesman problem (TSP), the above is not allowed. If a salesman, starting from his home city, is to visit each city on a given list exactly once and then return home, it is plausible for him to select the order in which he visits the cities so that the total distance traveled in his tour is as small as possible.

A Genetic algorithm (GA) is proposed to solve the vehicle routing problem. In this thesis the development of a GA is preceded by development of simple heuristics. These heuristics provide different operators for GA. The following sections develop the heuristics and the GA for routing. A section also describes conceptual work for using GA for the combined routing and scheduling problem.

Figure 3.2 shows a simple, two-customer vehicle routing problem. Each request consists of a pick-up and a drop-off. Hence, there are four 'nodes' in the problem. These

four nodes can be arranged in 4! ways. Because of the precedence conditions, some combinations (strictly speaking – permutations) do not form legal tours as they violate the precedence condition. Appendix B shows calculations for total number of tours possible. In the case of two-customer problem, of the twenty-four possible tours (both legal and illegal), only six are legal tours. They are as shown in Figure 3.2 in boldface type.

Customer 1, pick-up = P1, drop-off = D1
Customer 2, pick-up = P2, drop-off = D2

P1P2D1D2	D1P1P2D2
P1P2D2D1	D1P1D2P2
P1D1P2D2	D1P2P1D2
P1D1D2P2	D1P2D2P1
P1D2P2D1	D1D2P1P2
P1D2D1P2	D1D2P2P1
P2P1D1D2	D2P1P2D1
P2P1D2D1	D2P1D1P2
P2D1P1D2	D2P2P1D1
P2D1D2P1	D2P2D1P1
P2D2P1D1	D2D1P1P2
P2D2D1P1	D2D1P2P1

Figure 3.2: Possible Tours for a Two-Customer Pick-Up and Delivery Problem (legal tours are shown in boldface)

P1P2D1D2 represents the order customer1 pick-up, customer2 pick-up, customer1 drop-off and customer2 drop-off. This simple example illustrates the computational complexity of the vehicle routing problem. For a five-customer problem, 113,400 legal tours are possible. It is important that we cover the search space with 'good' search techniques so as to get a good estimate of the fitness of that route. For example a routing algorithm may 'mislead' the clustering algorithm by not performing the routing optimally. Note that in a combinatorial optimization problem the aim is not to come up with the best solution, but a better one. Many of the terms used in the explanation hence

tend to be qualitative. The development of the heuristic and genetic algorithm is discussed in next few sections.

Different types of 'calls' or 'requests' are handled by a typical transit agency. The call types are described in Appendix D. The customer may either request a pick up time (e.g. in case of a trip for shopping) or a drop-off time (e.g. in case of an appointment). It is assumed that a customer cannot request both pick-up and drop-off times for a particular trip. The problem is considered to be a single depot problem.

3.3 Evaluating a Route

A route can be defined as an ordered sequence of nodes. A sequence of nodes that is not ordered would be referred to as a cluster of nodes. Throughout this thesis, a node represents either a pick-up node or a drop-off node for a single customer. If two customers have the same drop off point (in space and time), it would still be represented as two different nodes. It is quite evident that a cluster would always have an even number of nodes. Each node has temporal and spatial characteristics.

How we evaluate a route is very important. Since our problem is a multi-objective optimization problem, changing the evaluation criteria (henceforth referred to as the evaluation function) would change the surface of the search space. For the purpose of optimization, we considered the following objective function:

Cost = W1 x (Time Window Violations) + W2 x (Distance Traveled in Miles)

W1 and W2 are weights that are set by the transit agency. Higher values of W1 reflect a transit policy of higher customer convenience, whereas higher values of W2 reflect a transit policy of minimizing the cost to the transit agency. The value of the evaluation function is calculated in miles by assuming a constant vehicle speed and multiplying the time term by an appropriate constant. For example, if we assume an average speed of

25mph and 180 minutes of time window violations, the time window violations would be translated into (180x25) / 60 = 75 miles of time window violations. Time window violations and distance traveled can be calculated in different ways, a few of which are described as follows.

Distance Calculations

Figure 3.3 illustrates two simple ways of calculating distance between two points.

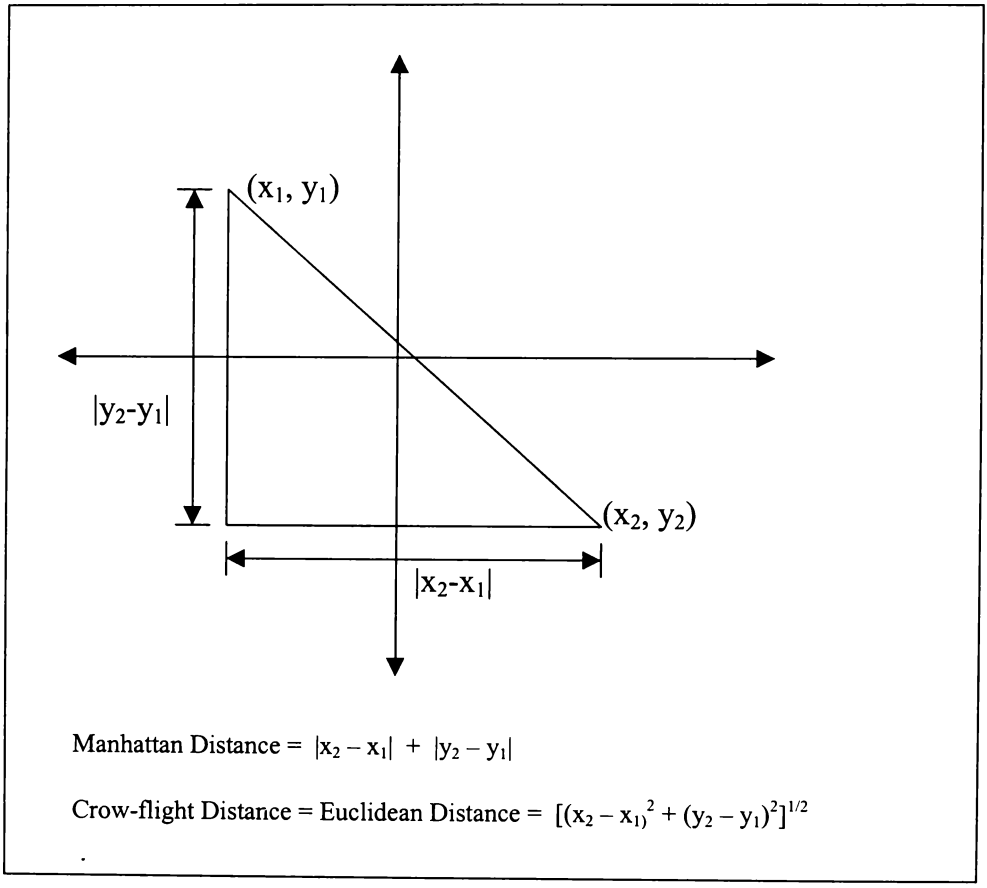


Figure 3.3: Simple Illustration of Manhattan Distance and Crow-Flight Distance

In our calculations we used crow-flight distance with a multiplication factor of 1.5 . This factor is inherent to the distance function that is encoded. In other words, W2 = 1.5 would mean distance between two points is (2.25) x Crow-Flight distance. The factor 1.5 is introduced to take into account the unknowns such as traffic delays, presence of signals etc. This factor is again the choice of the transit agency. A library of functions can be developed which would give distance by different formulae.

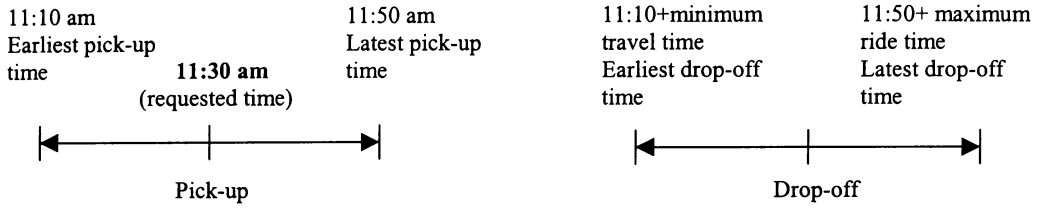
Time Window Violation Calculations

A time window can be defined as a fixed length of time in which a customer is promised to be picked-up, given that operating conditions remain predictable. There are different ways in which time windows can be calculated. Based on these time windows, different time window violation functions can be implemented. Figure 3.4 shows time window calculations. We considered three basic types of requests. A request consists of a pick-up and a drop-off. In a pick-up request, the customer specifies a pick-up time (for example, a trip to a grocery store) and lets the transit agency calculate the drop-off time. In a drop-off request, the customer specifies a preferred drop-off time and lets the agency find a suitable pick-up time. For an appointment drop-off request, the customer specifies the appointment time and the transit agency guarantees that the time window will not extend beyond the appointment time.

Figure 3.4 explains the time window calculations. In this example a time window of twenty minutes is assumed on either side of the requested time and forty minutes of total time window. Drop-off time window in case of pick-up request and pick-up time window in case of a drop-off request may look really very wide. In actual calculations latest drop-off time is dynamically calculated, once a pick up is performed. For example, if a customer is picked up at 10am, the latest drop-off time is modified dynamically. It is important to have an ordered list of nodes or a route before the time windows can be finalized. The time window violations are based on the dynamically modified latest drop-off time. Time windows that are calculated when the requests are first entered. These can be considered as 'absolute' time windows. A modified latest drop-off time would always

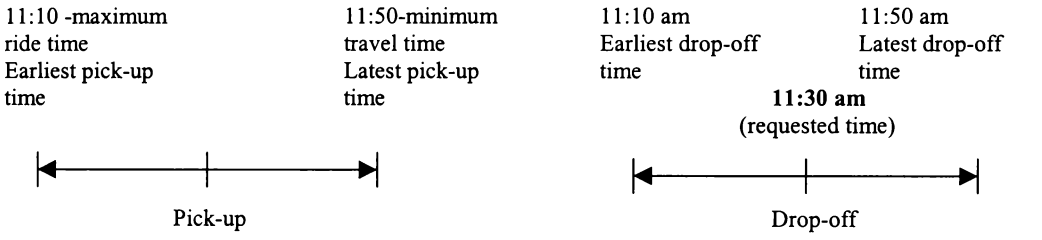
Pick-up Request

Time Window = 20 minutes on either side
Requested pick-up time = 11:30 am



Drop-off Request

Time Window = 20 minutes on either side
Requested drop-off time = 11:30 am



Appointment

Time window = 20 minutes on either side
Appointment time (drop-off time) = 11:30 am

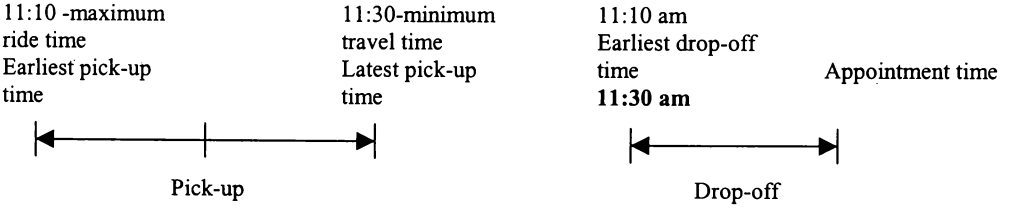


Figure 3.4: Illustration of Pick-Up and Drop-Off Time Calculations

be less than or equal to the initial latest drop-off time. Time window calculations are as explained below.

For a node, let

LT = Latest time a node can be visited

ET = Earliest time a node can be visited

LT is calculated dynamically for drop-offs only. A drop-off must follow its corresponding pick-up, but not necessarily immediately after its pick-up if the other passengers are served.

For a pick-up request at time 't',

ET = t – time window

LT = t + time window

At the corresponding drop-off node,

ET = ET at pick-up + minimum travel time between pick-up and drop-off node

LT = LT at pick-up + maximum ride time

When simulating a route,

LT at drop off = departure time at pick up + max ride time only if departure time at pick up + max ride time < LT at pick up. This will ensure that absolute LT is not exceeded

For a drop-off request at time 't',

ET = t – time window

LT = t + time window

At the corresponding pick-up node,

ET = ET at drop-off - maximum ride time

LT = LT at drop-off – minimum travel time between pick-up and drop-off node

When simulating a route,

LT at drop off = departure time at pick up + max ride time

Only if, departure time at pick up + max ride time < LT at pick up

This will ensure that absolute LT is not exceeded

For an appointment request at time 't', calculations are similar to that of a drop-off request, except ET = t (at the drop-off)

Arrival time and departure times at a node are dynamically calculated as a route is simulated. Once time windows have been established, time window violations are calculated as the route is simulated via a computer program. Figure 3.5 represents various time window violation functions.

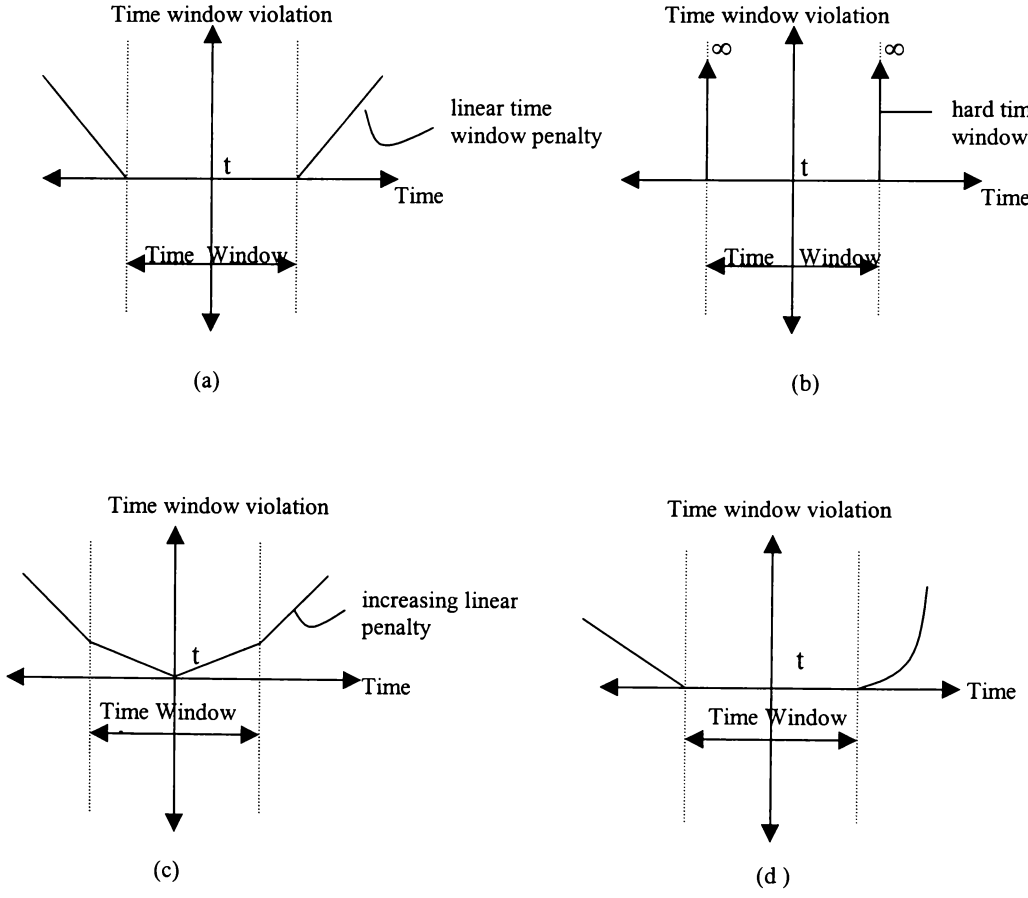


Figure 3.5: Example Functions for Calculating Time Window Violations

Different types of time window functions can be applied. In our model, we have linear (double) penalty for violating a normal time window. For an appointment time window violation, an exponential penalty function is used (square of the time window violation in minutes). Different methods can be implemented in a program to get the desired time window violation function.

3.4 Representing a Route

Section 3.3 described how a 'route' or a 'tour' or an ordered cluster of pick-up and drop-off nodes would be evaluated. The most natural way to represent a tour is through path representation. We represent a path with the help of linked list. In Java, Vector (a type of object) implements a list. A list or linked list is a special kind of data structure in which different objects are linked together. Let A, B, C, and D be the nodes in order. A linked list conceptually is like any other list and can be represented as follows.

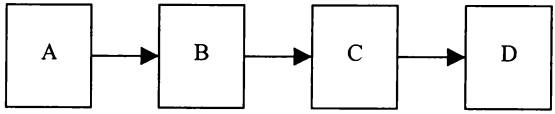


Figure 3.6 Linked List (Conceptual Representation)

3.5 Greedy Methods for Routing

Till now, we prepared the grounds for implementing routing algorithms. In this section we put the pieces together and describe greedy methods for the vehicle routing problem. The routing problem calls for ordering the nodes in a list such that an objective function is minimized. An algorithm or step-by-step recipe solves this problem. A greedy algorithm might also be called a "single-minded" algorithm or an algorithm that gobbles up all of its favorites first. The idea behind a greedy algorithm is to perform a single

procedure in the recipe over and over again until it can not be done any more and see what kind of results it will produce. It may not completely solve the problem, or, if it produces a solution, it may not be the very best one. However, a greedy algorithm is one way of approaching the routing problem and sometimes may yield good results. A node is either a pick-up or a drop-off node. A request consists of a pick-up and a drop-off node. And the necessary solution recognizes time window and precedence condition while seeking to minimize the objective function.

Triads Method

To start with we developed a routing algorithm just based on adjacency relationship firstly in time and then in space. The requests are ordered based on 'earliest time' using quicksort. Three "time ordered" nodes are clustered together and a minimum cost sequence of nodes is arrived at. Cost calculations are based on time window violations and distance as described in Section 3.4. Then the next three nodes are arranged. This is carried out until all the nodes to be routed are exhausted (Figure 3.7). This method is very fragile, works best in cases when requests are well separated in time, i.e., when there is sufficient time gap between two nodes. Three nodes at a time are considered as it is reasonable to calculate the cost of different permutations in a reasonable amount of computing time. For a 'n' node problem,
Number of computations = (quotient of n/3) x 6 + (1 or 2)

This method gives feasible results, but may be very expensive in terms of the cost of the route. The only advantage is that it takes very less computational time as compared to stochastic search techniques. The precedence condition and capacity constraints are checked once the ordering is complete. If they are not satisfied the "illegal" routes are dropped.

Best Node Method

This method is very similar to triads method described above. Only difference is, instead of arriving at a minimum cost sequence of nodes, only the first node in the minimum cost sequence is marked as visited. This is carried out till all the nodes have been considered for routing (Figure 3.8). This method may gives results that are far from optimality as it considers only three nodes at a time. The initial ordering of the route is based on 'earlier time' that introduces a bias towards time to start with. At the end precedence condition and capacity constraints are checked. For a 'n' node problem,

Number of computations $= (n-2) \times 6 + 2 \quad (n > 2)$

$= 2 \quad$ (for a two node problem)

Nearest Neighbor Algorithm

This algorithm starts a route by visiting the customer with the earliest pick-up time. At each location the cost of visiting three other locations is determined. These three nodes are determined by considering the space-time separation between the current location (which is already visited) and nodes that are yet to be visited. The minimum cost node is marked as visited and this procedure is repeated until all nodes are marked as visited. The space-time separation between two nodes is quantified based on an evaluation function similar to the one described in Section 3.3 (Kakivaya, 1995).

3.6 Genetic Algorithm for Routing

As stated earlier, a path representation is used for a tour. Figure 3.9 shows a GA with a modified crossover operator. Various functions or methods that are used to perform the "artificial life simulation" are described below. GA is a stochastic adaptive search technique.

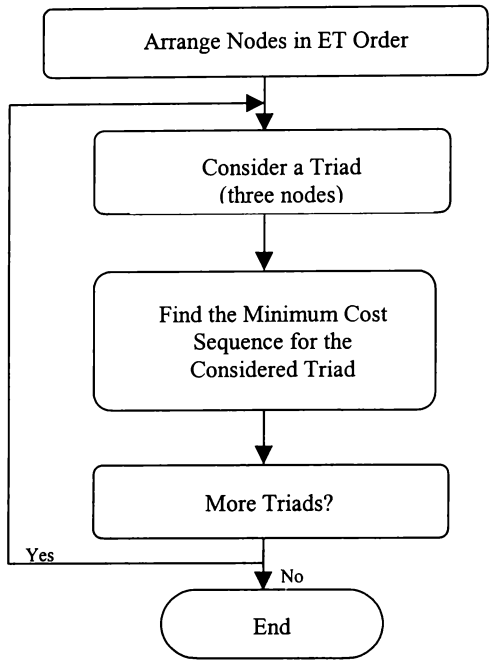


Figure 3.7: Triads Method

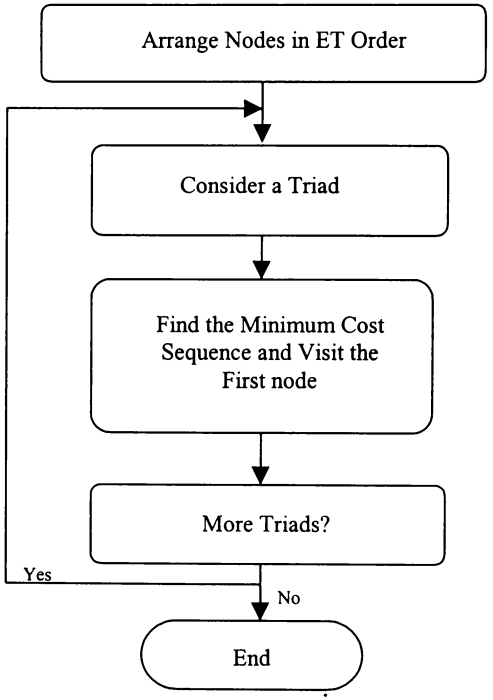


Figure 3.8: Best Node Method

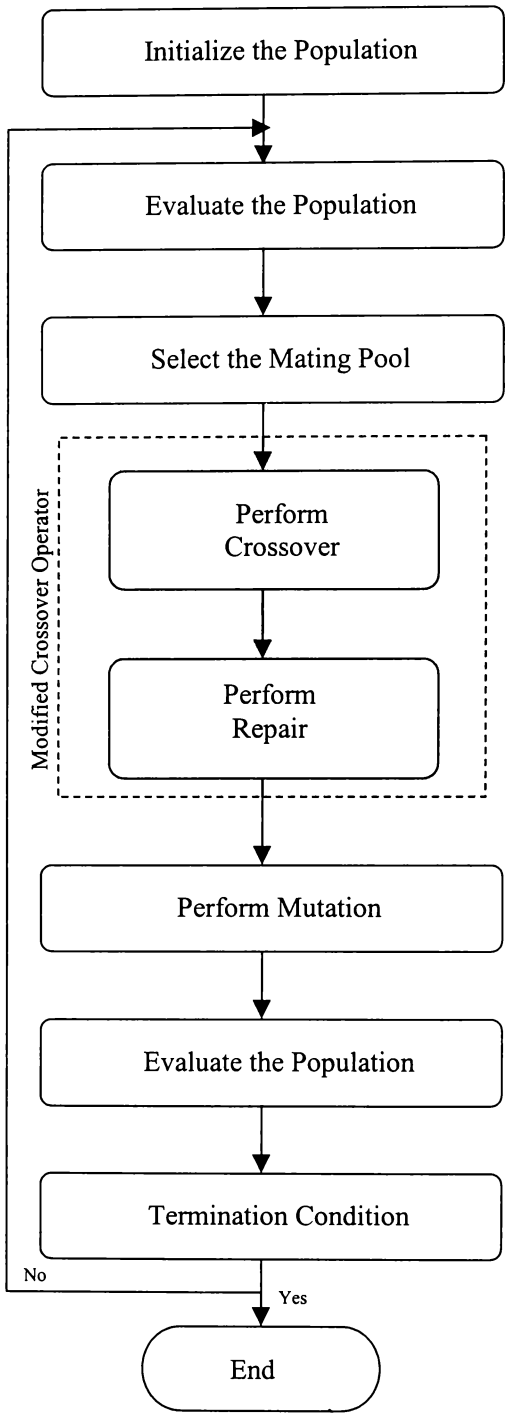


Figure 3.9: Genetic Algorithm with Modified Crossover Operator

Initialization: The initial population is arrived at randomly. Nodes are obtained in a list format and then are duplicated randomly until the population size is achieved. The precedence condition and capacity conditions are checked. If there is a precedence condition violation, it is corrected by exchanging the positions of the pick-up and the corresponding drop-off nodes. In the case of the capacity constraint, once the capacity of the vehicle has been reached, a drop off is forced before the next pick-up once. This initial population is evaluated by using evaluation criteria described in Section 3.3. The size of the population and the length of the chromosomes remain constant throughout the generations.

Tournament Selection: Different selection strategies can be implemented. Tournament selection is easy to implement and less time consuming. The current population is duplicated in random order and is compared to the original population letting better individuals to form the mating pool. This ensures that the best individual from the current population makes it to the mating pool and worst individual is eliminated (unless it is compared to itself by chance). Another selection strategy we implemented was to compare two randomly chosen individuals and accepting the better individual with some probability. Population here refers to a pool of solutions. 'Individual' or 'Chromosome' or 'String' or 'Genome' refers to a solution. In the routing problem, each route after evaluation will have a cost associated to it.

Crossover: A one point crossover function is implemented. Crossover is carried out with certain probability, which is specified as a parameter to GA. Crossover results in infeasible solutions. For this a repair mechanism is called upon. Crossover function with this repair mechanism is called modified crossover function.

Repair Mechanism: Based on observations about the triads method and the best node method and extending the logic presented by Uchimura and Sakaguchi (1995), we propose and implement a new modified crossover operator. Figure 3.10 shows the proposed modified crossover operator.

There are certain properties of a route that can be used to advantage. For example,

- The first node and the last node will always be a pick-up and a drop-off node, respectively.
- The length of the string is always even.
- A node cannot repeat itself more than twice after crossover.
- The number of missing nodes equals the number of repeated nodes.

As shown in Figure 3.10, Child 1 inherits properties from both the parents. First part of the resulting string (Child 1) is inherited from Parent 1, and second part from Parent 2. Repeated nodes are searched starting from the initial position until the crossover point by considering each node and checking if it occurs in the latter part of the string (child). Missing nodes are noted by comparing the latter part of Parent 1 string with Child 1. The same procedure is repeated for Child 2. The ordering is necessary as the nodes before crossover point in a Child 1 will be the same as nodes in Parent 1, before crossover point. Once repeated and missing nodes are identified, repeated nodes are eliminated by finding out the minimum cost sequence for each repeated node. This is based on the adjacency relationship. The cost of a node at a particular node is found by considering the nodes preceding and succeeding it and using a cost function similar to the one used to evaluate the entire route. Once repetitions are eliminated, a few 'spots' in the original list remain empty. For each 'empty spot', minimum cost substitution is searched for from the missing nodes list. After these substitutions, based on adjacency relationships are carried out, the resultant modified child is checked for the precedence condition and capacity constraint. In Figure 3.10, find the best node (one with minimum cost) from missing nodes to be placed at the position of repeated nodes, starting from left-hand side of the organism. In the case illustrated, missing node D3 can only take one position. The idea behind the proposed crossover operator is that the minimum cost substitutions help find the best 'spots' as it would represent substitution with minimum time distance and minimum time window violations. This in turn would result in lesser-cost solutions.

Mutation: Mutation causes to completely replace a randomly chosen organism with a legal tour. A common view in the GA community is that crossover is the major instrument of variation and innovation in GA, with mutation insuring the population against permanent fixation at any particular locus and thus playing a major role.

Elitist Operator: This function replaces the worst individual (costliest) organism with the best individual (over all the generations) if the best individual in the current population is not better compared to the best individual over all generations.

Feedback GA: It is assumed that there are lots of generations in an evolutionary process. We get the best individual at the end of an 'evolution' which is accepted as solutions. The evolutionary process is then repeated from the beginning (starting with Initialization) and the best individual is passed to it with the help of an elitist operator. Figure 3.11 shows a GA with feedback.

3.7 Genetic Algorithm for Scheduling – Conceptual Work

The combined routing and scheduling problem has two aspects – routing and scheduling. Routing involves servicing customers on a particular route in an optimum way. All the routes for a day make up the schedule of the day. So, in order for a schedule to be "optimal" it is important that all the routes are handled "optimally".

We developed a "bit-table" representation of the problem as shown in Figure 3.12. Each row in the table represents a route and each column represents a trip, which involves a pick-up and a drop-off for a particular passenger. If we have 'n' customers and they need a return trip too, there will be '2n' columns. A customer need not ask for a return trip, but that is typically the case. The column size remains fixed but the row size of the table may vary as the solutions proceed from one generation to other. In a column there can be at most one "1" and the remainder of the bits are zero. Specialized crossover and mutation operators can be defined. With this representation there will not be any infeasibilities after crossover is performed. Other parameters such as number of

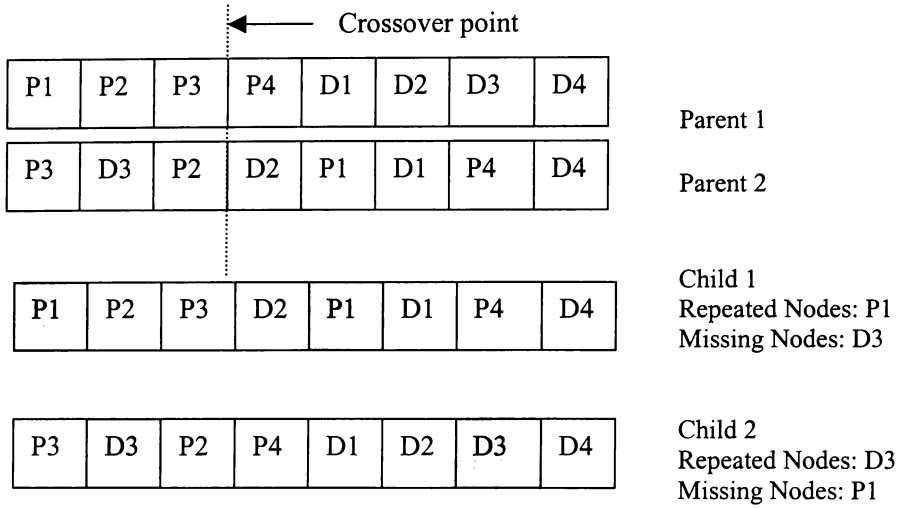
generations, number of solutions in a generation, crossover and mutation probabilities etc. will be elaborated later.

The Fitness function is very important as it guides different generations towards optimality. Inheritance and crossover depends on the fitness of a solution as GA follows a natural selection process. In our case a solution represents full days schedule. Three obvious, efficiency and effectiveness measures are number of miles traveled, time window violations (in units of time) and number of vehicles required. GA for scheduling would generate clusters and give those clusters to GA for routing. GA for routing would, in turn, return cost associated with that cluster after trying to route it sub-optimally. The solutions in the entire population can be ranked based on their fitnesses to carry out selection and crossover.

3.8 Summary

In this chapter we looked into development of three heuristic approaches and genetic algorithm for vehicle routing problem. It also described the new modified crossover operator and feedback GA. Section 3.7 described conceptual work about the representation of a schedule for a day. In the next chapter implementation details and experimentation with GA are discussed.

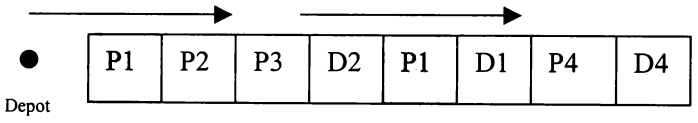
Crossover



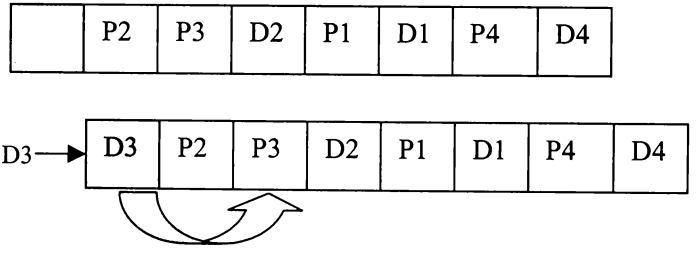
Repair

Child 1 (Repeated Nodes: P1, Missing Nodes: D3)

Cost (Depot – P1 –>P2) > Cost (D2 – P1 – D1)



Therefore, Child 1 with accepted position of P1 is as follows.



Check **precedence** condition and then **capacity** constraint

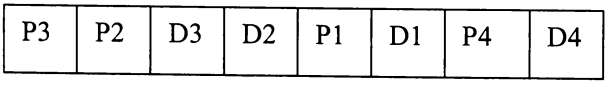


Figure 3.10: An Illustration of Modified Crossover Operator

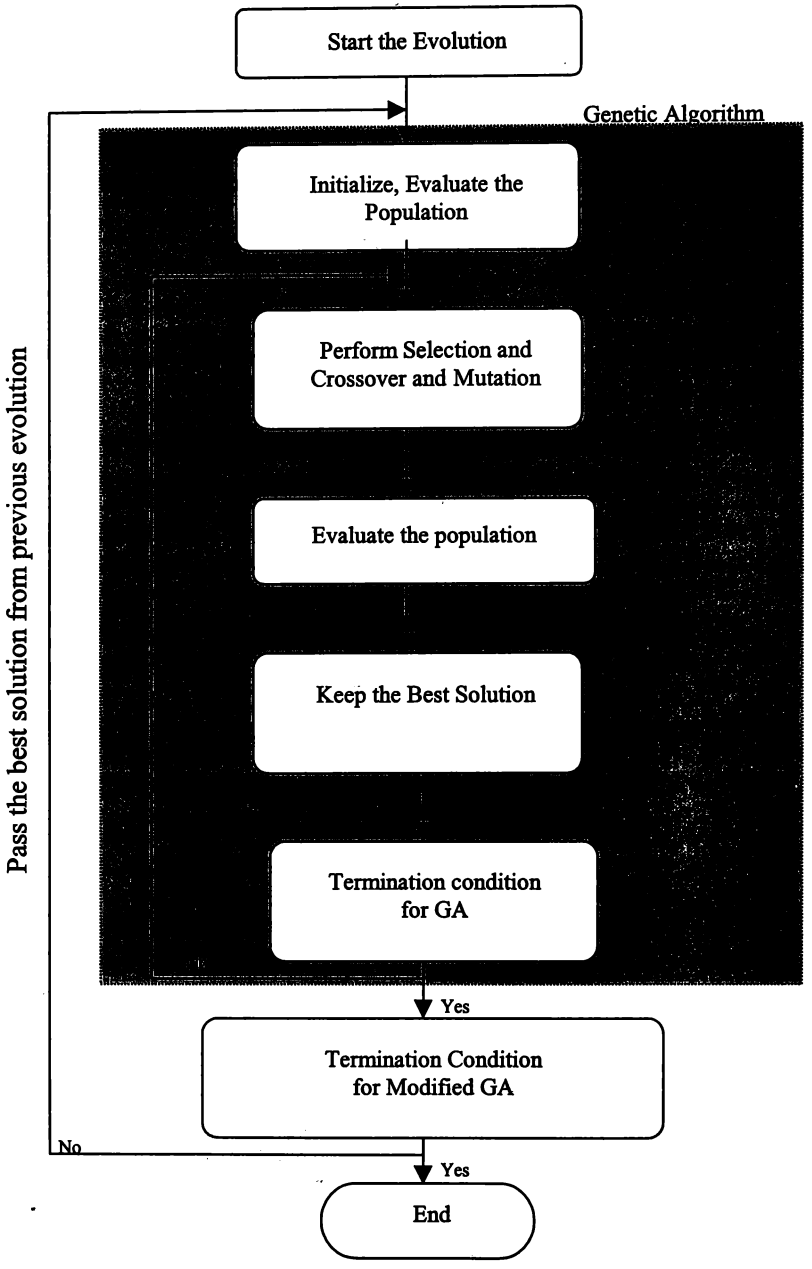


Figure 3.11: Flowchart of Feedback GA

	Pass 1	Pass1 Return Trip						Pass n	Pass n Return
Route999	0	0						0	0
Route 0	0	0						0	0
Route 110	1	0						0	1
Route 719	0	1						0	0
Route 781	0	0						1	0
Route m	0	0						0	0

Figure 3.12: Representation of a Schedule

Chapter 4
Results and Implementation

4.1 Results

In this section we are going to look at the performance of the triads method, best node method, nearest neighbor algorithm (NN) and GA for routing five, eight and twenty-five customers respectively. As mentioned in the Chapter 3, there is no guarantee about the quality of the solutions that heuristic methods provide. Researchers have used the nearest neighbor algorithm as a routing technique (Kakivaya, 1996). A brief recap of terms explained in earlier chapters is also presented below for ease of reading. Each customer has a 'request' which consists of a pick-up and a drop-off. Each pick-up and drop-off is referred to as a separate node. Hence in a five-customer problem there are ten nodes. In the traveling salesman problem (TSP), a salesman tries to minimize the distance traveled. Vehicle routing problem is similar to TSP. In a vehicle routing problem, the vehicle must:

- Visit all the nodes.
- Follow the precedence condition, which is visiting a customer's origin before visiting drop-off.
- Take into account the capacity constraint.
- Observe time windows.
- Minimize the travel distance.
- Start and end a route at the depot.

Some of the measures of performance of a route and their definitions are listed in Table 4.1.

Table 4.1: Measures of Performance of a Route

Measure of Performance	Explanation
Distance Traveled	Total distance traveled on a route after visiting all the nodes. Route begins and ends at a depot.
Time Window Violation	Summation of all the time window violations at each node.
Hours of Operation	Total hours of operation on this route. This is the time period between when the vehicle leaves depot and returns to the depot at the end of the route.
Excess Ride Time	It is defined for a pair of pick-up and drop-off nodes. Excess ride time is any ride time above the minimum travel time between these two nodes.

There are different ways of implementing the distance and time window violation functions (Chapter 3). This decision is typically a policy decision to be made by a transit agency. For the test problems in this section the following functions are assumed.

Distance: Distance between two points is assumed to be Euclidean distance multiplied by 1.5. The factor may be justified by taking into consideration geography of the region or to account for the traffic uncertainties.

Speed: An average of 25mph is assumed for the vehicle serving a route.

Time Window: A time window of 20 minutes on either side of the scheduled pick-up or drop-off is assumed.

Maximum Ride Time: A passenger cannot be on a vehicle for more than an hour.

Load Time: A load time of 3.5 minutes at each node is assumed. This is for both loading (pick-up) and unloading (for a drop-off) of a customer. Any immediate pick-up or drop-off at this point is considered to have additional load time of one minute.

Capacity: Capacity of the vehicle is assumed to be 25. In our problems, because of the problem size capacity is not exceeded. For larger size problems, a drop-off is forced before pick-up in case of capacity constraint violation.

Time Window Violation: There is no penalty associated for arrival before ET at a node. The vehicle is assumed to idle until the ET is reached. As explained in Chapter 3, ET and LT calculations are based on a maximum ride time policy. For a delayed pick-up or a

drop-off a linear penalty is associated with slope of the line equaling two. For example, five minutes of time window violation will be counted as ten minutes of penalty. For an appointment time violation, the penalty is the square of the time window violation, that is five minutes of time window violation counts as $5^2 = 25$ minutes of time window violations. This time window violation is then converted into miles by assuming the same speed of 25 mph.

Cost Function: Cost associated with a route is calculated as,

(W1 x Time Window Violations + W2 x Total Distance Traveled)

where W1 = 2 and W2 = 1 for the purpose of calculations.

All the algorithms are tested on Intergraph TD-225 (Pentium II, 300 MHz) computer in the Transportation Systems Laboratory of North Carolina State University. CPU runtime could not be measured, as Java does not provide a way to measure CPU time. Actual execution time may be measured. This may not be a good measure of the performance as the observed execution time may vary depending on the system load.

Five Customer Problem: All five customers have the same origin and destination. These requests are separated one hour each in time. A taxicab like service is expected, otherwise the time window violations would be very high. Table 4.2 tabulates the requests to be routed. Figure 4.1 shows the spatial distribution of the nodes. Table 4.2 summarizes the results obtained from different algorithms.

Table 4.2: Five Customer Data

Customer	Origin	Destination	Trip Type and Time
Depot	(0,0)	(0,0)	Start/End
Customer1	(5,5)	(10,10)	Pick-up at 9:00
Customer2	(5,5)	(10,10)	Pick-up at 10:00
Customer3	(5,5)	(10,10)	Pick-up at 11:00
Customer4	(5,5)	(10,10)	Pick-up at 12:00
Customer5	(5,5)	(10,10)	Pick-up at 13:00

TWV, HOP, and ERT stand for time window violations, hours of operations, and excess ride time respectively. This nomenclature is followed for subsequent tables in this chapter.

Now we conduct GA experiments to simulate the 'artificial life' using modified crossover operator. Note that the length of a string representing a solution remains constant as it represents nodes to be served and all the nodes are to be served. GA parameters used are given below.

Population size	: 10
Number of Generations	: 50
Selection	: Tournament selection
Crossover	: One point modified crossover, crossover percentage 75
Mutation	: Mutation percentage 3

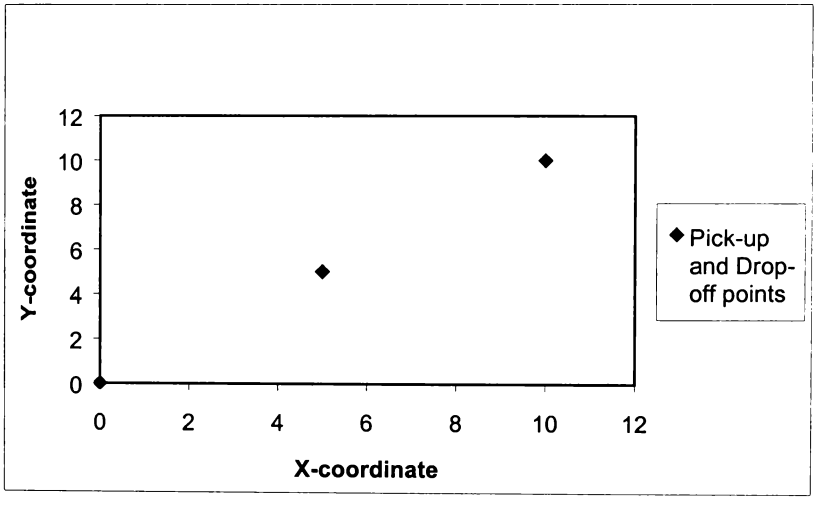


Figure 4.1: Spatial Distribution of the Nodes (five customer problem)

Table 4.3: Result for Five Customer Problem

Method	Runtime (seconds) & (Computations)	Distance (miles)	TWV (minutes)	HOP	ERT (minutes)	Cost (miles)
Triads	2 (8)	127	0	5 Hrs 48 min	0	127
Best Node	2 (20)	127	0	5 Hrs 48 min	0	127
NN	2	84	474	5 Hrs 57 min	255	480
GA	60 (500)	127	0	5 Hrs 48 min	0	127

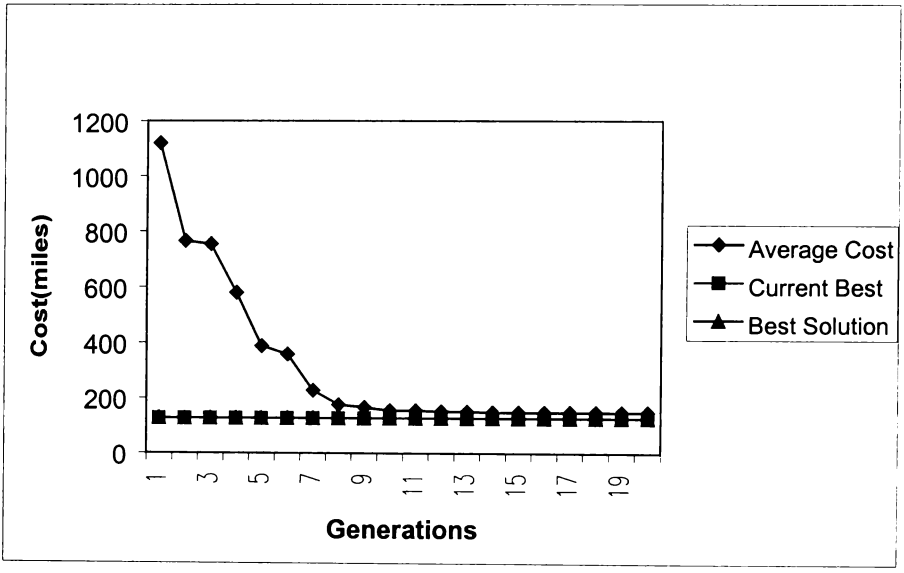


Figure 4.2: Convergence of GA for Five Customer Problem

Figure 4.2 shows a graph of average cost of solutions in a population, cost of best solution in the current generation, and cost of overall best solution against the number of generations. As seen from Figure 4.2, the solution to five-customer problem is obtained fairly quickly even with a lower population size and a fewer number of generations. We are concerned about the quality of solution along with the number of computation and GA results are much better than other heuristics as seen from the Table 4.2. Note that the quality is stressed as in case of NP-problems, simple heuristics may produce very bad results even though the computation time is very less.

Eight Customer Problem: This problem covers all the three types of requests–pick-up request, drop-off request and appointment request. These requests are shown in Table 4.4. The results are shown in Table 4.5.

Table 4.4: Eight Customer Data

Customer	Origin	Destination	Trip Type and Time
Depot	(0,0)	(0,0)	Start/End
Customer1	(5,5)	(10,10)	Pick-up at 9:00
Customer2	(5,5)	(10,10)	Pick-up at 10:00
Customer3	(5,5)	(10,10)	Pick-up at 11:00
Customer4	(5,5)	(10,10)	Pick-up at 12:00
Customer5	(5,5)	(10,10)	Pick-up at 13:00
Customer6	(3.5,3.5)	(2.5,2.5)	Drop-off at 10:00
Customer7	(3.5,3.5)	(5,5)	Pick-up at 10:15
Customer8	(5,5)	(4.5, 4.5)	Appointment at 10:15

As seen from Table 4.5, GA performs much better than other heuristics. Since GA is a stochastic process, the final solution (best organism) may depend on the random seed that GA starts with, hence it is advisable to run GA five or six times to get the best solution. Taking a cue from this, we experimented with Feedback GA as described in Chapter 3. GA starts all over again after a few generations. In this case it was observed that the best solution does not change after twenty-five generations. Hence GA was started again with the best solution from previous run or 'evolution'. This indeed improved the performance of the GA and the quality of the subsequent best solution. It is quite evident from the results shown in Table 4.5. Feedback GAs even though computationally 'expensive' can be used for benchmarking or high accuracy. The parameters for GA used in obtaining the results in Table 4.5 are given as follows.

Population size : 50
Number of Generations : 200
Selection : Tournament selection

Crossover : One point modified crossover, crossover percentage 75

Mutation : Mutation percentage 3

Parameters used in Feedback GA are given below.

Number of Evolutions : 8

Population size : 50

Number of Generations : 25

Selection : Tournament selection

Crossover : One point modified crossover, crossover percentage 75

Mutation : Mutation percentage 3

Figures 4.3 and 4.4 show the variations of average cost, cost of best solution in the current generation, and overall best solution with generations. As it is seen from the above graph, the average cost of the population decreases indicating that solutions are getting 'fitter' and fitter. But only convergence is not sufficient. It is necessary to see if GA gives results that are better than other heuristics or not. As seen from Table 4.5 show that GA gives better results as compared to other heuristics. It also shows that with proper tuning of various parameters, feedback GA can achieve much better results than other methods. For the problem at hand, Feedback GA (Figure 4.5 and Figure 4.6) gave better results than all the other methods.

Table 4.5: Result for Eight Customer Problem

Method	Runtime(secs) & (computation)	Distance (miles)	TWV (mins)	HOP	ERT (mins)	Cost (miles)
Triads	2 (14)	125	5665	5 Hrs 56 min	182	4846
Best Node	2 (38)	123	162	5 Hrs 51 min	126	258
NN	2	97	1198	6 Hrs 16 min	349	1096
GA	55 (10000)	95	87	5 Hrs 1 min	202	168
Feedback GA	143(10000)	118	0	4 Hrs 57 min	45	123

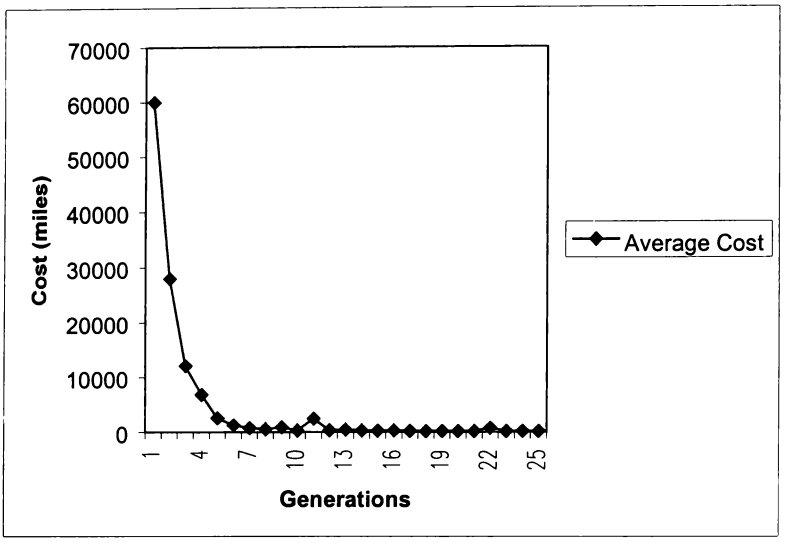


Figure 4.3: Plot of Average Cost versus Generations (eight customer problem)

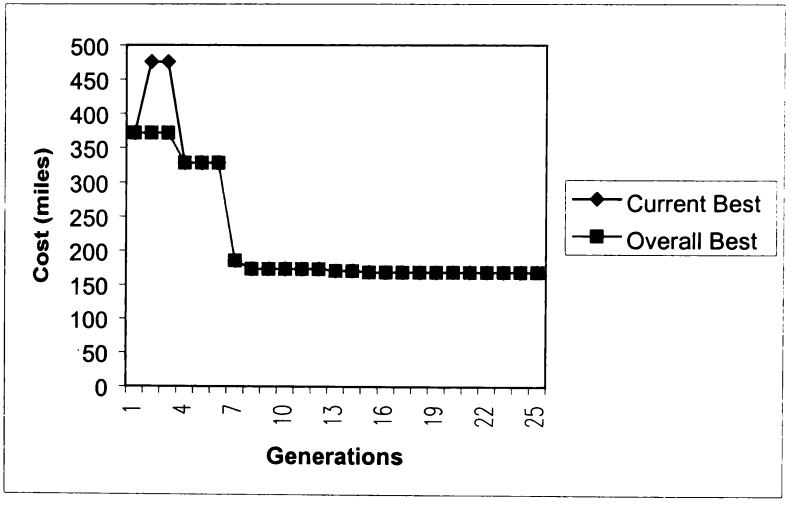


Figure 4.4: Plot of Cost of Current Best Solution and Overall Best Solution versus Generations (eight customer problem)

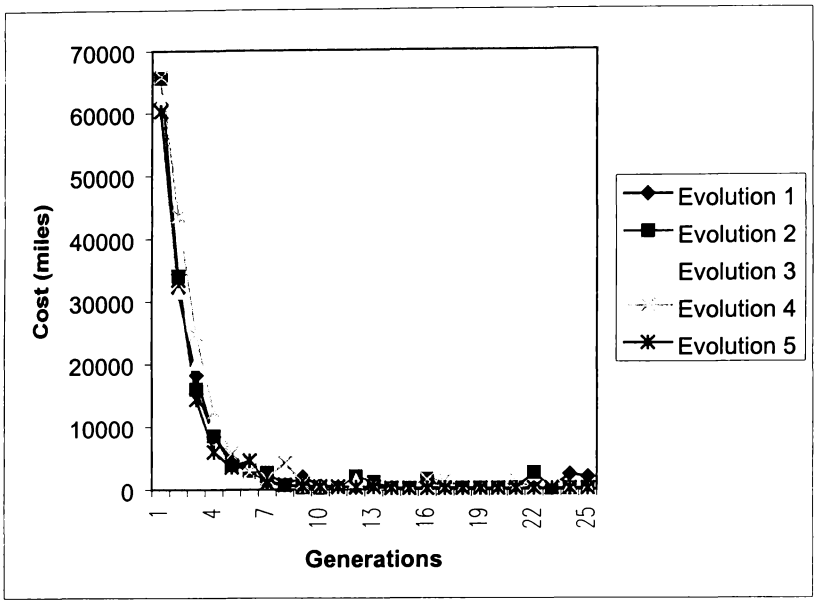


Figure 4.5: Plot of Average Cost versus Generations (eight customer problem using feedback GA)

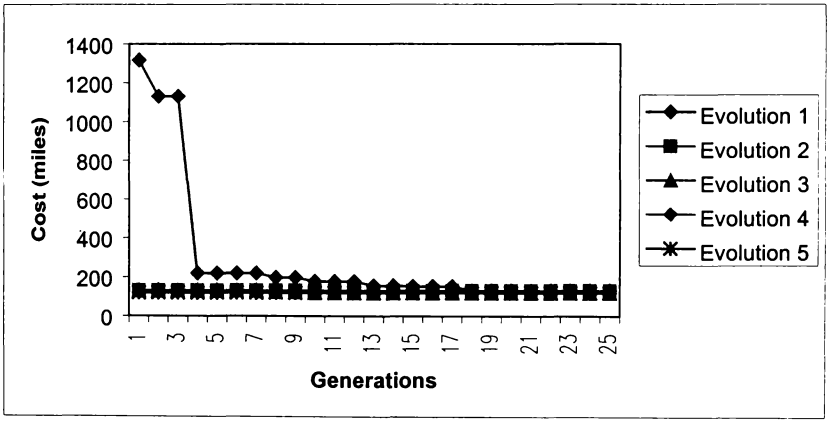


Figure 4.6: Plot of Best Solution Cost versus Generations (eight customer problem using feedback GA)

Twenty-five Customer Problem: As the number of nodes increase, the number of computations required for brute force calculations increase rapidly because the problem at hand is a multi-objective, combinatorial optimization problem. We obtained results for five and eight customer problems using the triads method, best node method and nearest neighbor heuristic. Experiments were also conducted to get parameters for GA and feedback GA. In Section 4.4.2, we saw that feedback GA improved the quality of the end solution. Feedback GA maybe used more as a benchmarking tool than as an optimizing algorithm as it may become computationally expensive with the increase in the size of the problem to have large population and more number of generations and evolutions. Note that a twenty-five customer problem has fifty nodes to be served. Some of the nodes shown in Figure 4.7 (refer Table 4.6) have same X and Y-coordinates, hence it may not be evident from the figure that there are fifty nodes in the figure. The co-ordinate system can be adjusted so that all the points lie in only one quadrant.

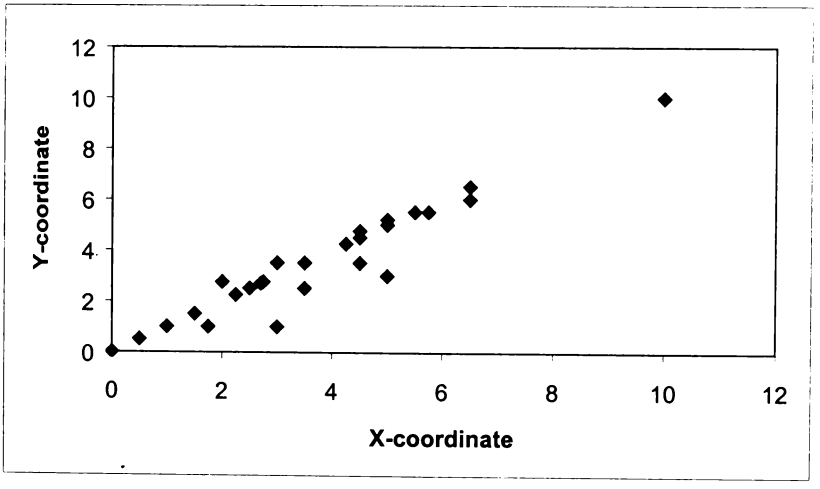


Figure 4.7: Spatial Distribution of the Points (twenty-five customer problem)

Table 4.6: Twenty-five Customer Data

Customer	Origin	Destination	Trip Type and Time
Depot	(0,0)	(0,0)	Start/End
Customer1	(5,5)	(10,10)	Pick-up at 9:00
Customer2	(5,5)	(10,10)	Pick-up at 10:00
Customer3	(5,5)	(10,10)	Pick-up at 11:00
Customer4	(5,5)	(10,10)	Pick-up at 12:00
Customer5	(5,5)	(10,10)	Pick-up at 13:00
Customer6	(3.5,3.5)	(2.5,2.5)	Drop-off at 10:00
Customer7	(3.5,3.5)	(5,5)	Pick-up at 10:15
Customer8	(5,5)	(4.5, 4.5)	Appointment at 10:15
Customer9	(1.5,1.5)	(5.5,5.5)	Pick-up at 12:00
Customer10	(2.5,2.5)	(4.5,4.5)	Pick-up at 12:30
Customer11	(3.5,3.5)	(4,4)	Pick-up at 15:00
Customer12	(4.5,4.5)	(4.25,4.25)	Pick-up at 14:00
Customer13	(5.5,5.5)	(1,1)	Pick-up at 13:30
Customer14	(0.5,0.5)	(6.5,6.5)	Drop-off at 15:30
Customer15	(4.5,4.5)	(2.75, 2.75)	Appointment at 14:15
Customer16	(5,5)	(1.5,1.5)	Appointment at 11:30
Customer17	(3,3.5)	(5.5,5.5)	Pick-up at 15:00
Customer18	(2.5, 2.7)	(3.5,3.5)	Pick-up at 15:30
Customer19	(5.75, 5.5)	(2.25,2.25)	Pick-up at 16:00
Customer20	(4.5, 4.75)	(4.25,4.25)	Pick-up at 13:00
Customer21	(1,1)	(5,5.2)	Drop-off at 13:30
Customer22	(3.5,2.5)	(6.5,6)	Drop-off at 14:30
Customer23	(4.5,3.5)	(2,2.75)	Appointment at 14:45
Customer24	(5,3)	(3,3.5)	Appointment at 11:15
Customer25	(3,1)	(1.75,1)	Appointment at 10:40

As seen from Table 4.7, the advantage of using GA is very clear. In this case GA performs much better than the heuristic methods. 'Cost' associated with a day's schedule depends on the cost of individual routes. Hence a robust routing algorithm is important for the success of a scheduling algorithm. A few experiments performed using GA and feedback GA are reported in Table 4.8. It can be seen that the feedback GA experiments prove computationally expensive and do not improve the quality of the solution. They may be used initially for benchmarking. As the search space increases, the best solution varies according to the random seed. Hence it is advisable to run the programs five to six times before deciding a best solution. For this problem size, the following parameters give fairly consistent results. Parameters used in GA are given below. Also, the runtime varies depending on the system load, for Windows NT operating system. Figures 4.8 and 4.9 show the plots of costs versus generations.

Population size	: 300
Number of Generations	: 100
Selection	: Tournament selection
Crossover	: One point modified crossover, crossover percentage 75
Mutation	: Mutation percentage 3

Table 4.7: Result for Twenty-five Customer Problem

Method	Runtime(secs)& (computations)	Distance (miles)	TWV (mins)	HOP	ERT (mins)	Cost (miles)
Triads	6(50)	276	234037	13 Hrs 57 min	1249	195307
Best Node	8(140)	251	78625	12 Hrs 59 min	1251	65773
NN	7	154	231562	14 Hrs 49 min	3142	192983
GA	345(30,000)	151	3541	9 Hrs 22 min	2098	3102

Table 4.8: Various GA Parameters and Their Effect on Quality of Solution

Population Size	Gener-ations	Evolutions (Feedback)	Runtime(secs) & (computations)	Crossover Percentage	Mutation Rate per 1000	Cost (miles)
200	100	75	7156 (1,500,000)	75	5	1914
200	100	40	2494 (800,000)	75	5	1994
100	100	1	195 (10,000)	75	30	4831
200	100	1	223 (20,000)	75	5	3236
200	50	2	254 (20,000)	75	5	4223
300	100	1	345 (30,000)	75	5	2568
300	100	1	360 (30,000)	75	5	2967
300	100	1	355 (30,000)	75	30	3102

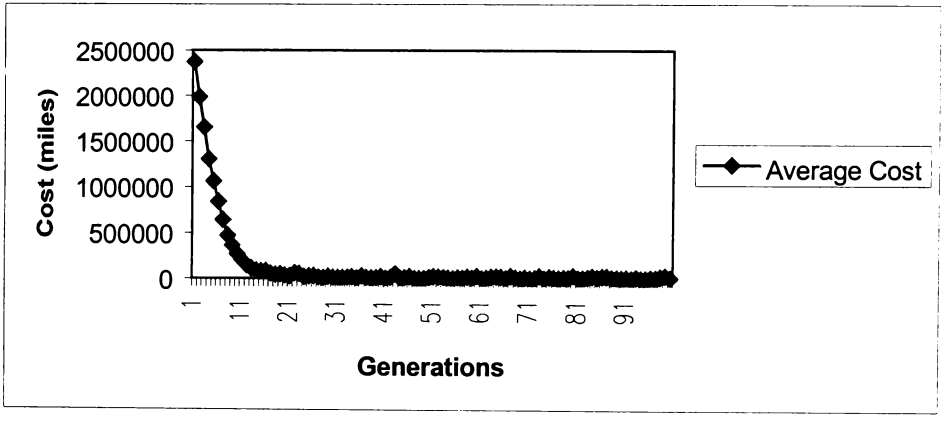


Figure 4.8: Plot of Average Cost versus Generations (twenty-five customer problem)

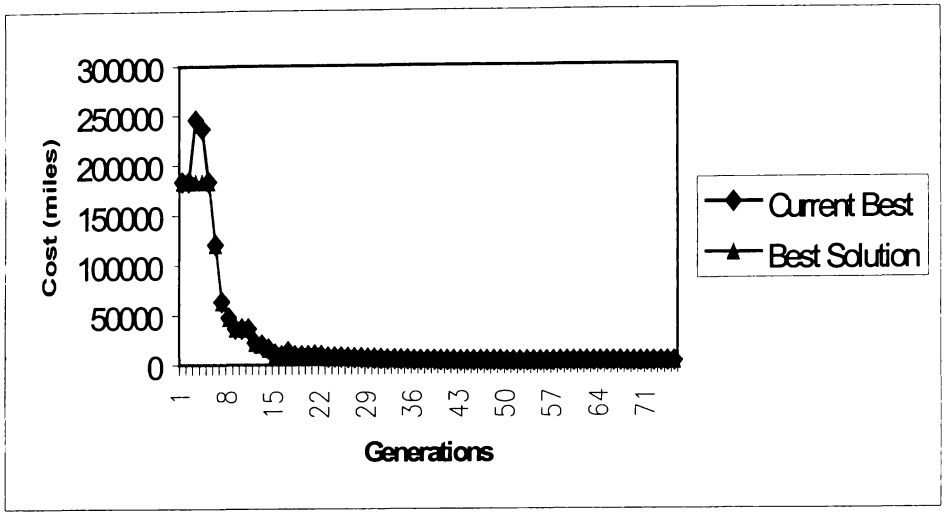


Figure 4.9: Plot of Best Solution Cost and Current Best Cost versus Generations (twenty -five customer problem)

4.2 Programming in Java

Java is used as a programming language for the purpose of encoding algorithms described in Chapter 3. Java is an object oriented (OO) programming language. OO languages and techniques are based on the concept of an "object" which is a data structure (abstract data type) encapsulated with a set of routines, called "methods" which operate on the data. Operations on the data can only be performed via these methods, which are common to all objects, which are instances of a particular "class". Methods or code in one class can be passed down the hierarchy to a subclass or inherited from a superclass (inheritance).

Java is not only an Internet programming language. It has lots of other potentials such as creating distributed environment. There are lots of other advantages of using Java as programming language. Some of which are listed as follows.

- Java is portable across platforms. It is based on the paradigm of "write once, run everywhere"

- Code is reusable. For example, repetition of code can avoided through proper use of inheritance
- Graphical User Interface is simple to create. With the advent of Java's 'Swing' GUI component, look and feel across different platforms can be maintained.
- Documentation is facilitated using 'javadoc'
- With the ease of coding and excellent documentation, it is easy to communicate code to different users which in turn enhance the reusability of the code
- With the automatic garbage collection, programmer does not need to worry about memory allocation and deallocation.

Java's Vector class is used to represent a route. Vector class implements linked list functionality. In GA, vectors represent organisms in the population. Vector 'stores' an object by maintaining a 'reference' to that particular object, instead of 'cloning' the object's structure again. This means a lot of saving in terms of memory usage. When a vector is cloned, references are cloned keeping the basic objects in place. Even though it facilitates memory management, one should be careful about modifying an object's state and using it inadvertently.

Packages and Documentation

Different 'classes' or 'objects' encoded are documented using Java's 'javadoc' and can be found at the URL given in Appendix E. The classes are put in different directories according to their functionality. This also facilitates usage of Java's 'package' command. A package mostly contains objects that perform related tasks. For example, classes used for performing routing are put in a package 'ncsu.transportation.ga.routing'. This improves the readability and understanding of the code considerably. Appendix H shows sample code for a file named 'Distance.java'. This is a library class that provides methods for calculating distance. Table 4.9 presents an overview of different classes. Table 4.10 through Table 4.14 list some of the classes written to implement routing heuristics, GA for routing, GUI and file I/O operations. Screenshots of the documentation are shown in Appendix F.

Non-Interactive Graphical User Interface Using Java™'s Swing Components

We developed non-interactive Graphical User Interface (GUI) using Java's platform independent swing components. The GUI monitors the progress of GA and plots graphs of average cost, cost of overall best solution and cost of best solution in current generation against the progress of generations. This is achieved using concept of threads in Java. Screenshots of GUI are included in Appendix G.

Table 4.9: Overview of the Packages

Package	Summary
ncsu.transportation.ga	This package contains all the useful and common classes
ncsu.transportation.ga.gui	This package contains all the useful classes used to create Graphical User Interface and plots
ncsu.transportation.ga.io	This package contains useful classes for file I/O operations.
ncsu.transportation.ga.routing	This package contains all the useful classes for implementing routing algorithms
ncsu.transportation.ga.scheduling	This package contains classes necessary to carry our scheduling

Table 4.10: Objects in Package 'ncsu.tranasportation.ga' and Their Functionality in Brief

Object (File: *.java)	Functionality
Parameters (*Interface*)	This interface provides different parameters
GA (*Interface*)	All implementing classes provide implementation of the method signatures defined in this interface
QuickSort	Provides methods for implementing quick-sort algorithm
RandomIntGenerator	An improved random number generator

Table 4.11: Objects in Package 'ncsu.tranasportation.ga.routing' and Their Functionality in Brief

Object (File: *.java)	Functionality
CostObserver (*Interface*)	Provides interface for updating the graph during the GA run
Distance	Provides methods for calculating distance between two points
GArouting	Implementation of GA for routing. It is a subclass of GA, Parameters and Routing
GeoToCartesian	Provides methods for converting geo-coordinates to Cartesian coordinates
MyStringTokenizer	Provides methods for separating substrings in a delimited string
NearestNeighbor	Provides implementation of Nearest Neighbor Algorithm
Routing	Provides methods for carrying out routing using Triads and Best Node methods described earlier
SpaceTime	A basic Provides representation of a node and methods to calculate ET, LT etc
TWVfunc	Provides methods for calculating time window violations

Table 4.12: Objects in Package 'ncsu.tranasportation.ga.gui' and Their Functionality in Brief

Object (File: *.java)	Functionality
GAroutepanel	Panel for showing the plot of GA run
GARoutingThread	Class that runs GA as a separate thread
MyFrame	Frame which acts as a 'container' for GUI components
MyTabbedPane	Provides tabbed pane
Plot, PlotBox, PlotPoint	Provide methods for plotting

Table 4.13: Objects in Package 'ncsu.tranasportation.ga.io' and Their Functionality in Brief

Object (File: *.java)	Functionality
RideFileReader	Reads and separates a tab delimited rider file. A ride file has whole day's schedule in it
TwoIntegers, TwoDoubles	Helpful classes that provide storage of two related numbers

Table 4.14: Objects in Package 'ncsu.tranasportation.ga.scheduling' and Their Functionality in Brief

Object (File: *.java)	Functionality
Schedule	Provides methods and variable for handling a day's schedule (not fully implemented)
GAschedule	Implements GA for scheduling (not fully implemented)

4.3 Summary

In the first section of this chapter, sample problems are solved to show the merit of genetic algorithm for vehicle routing problem. Various graphs and tables shown indicate that GA with modified crossover operator performs much better that other heuristics. This section also describes a few experiments with feedback GA.

In the second section of this chapter we saw advantages of Java as a programming language. Java being a popular internet programming language, distributed environment can be easily used to speed up the computations. With different developments such as platform-independence, lightweight graphical components, and easy documentation – Java as a programming language has a vast potential and can be used in various areas including paratransit.

Chapter 5
Summary, Conclusions and Recommendations

5.1 Problem Summary

As described earlier, the vehicle routing problem is a combinatorial optimization problem. Moreover the paratransit vehicle routing problem is a multi-objective optimization problem meaning that the objectives compete against each other. The search space for these kinds of problems varies according to the variation in the passenger service policy. For example objectives may be minimized or maximized based on the weights assigned to each service objective such as time window violations, distance traveled, operational cost, etc.

In general the objectives of the combined vehicle routing and scheduling problem must be optimized subject to a variety of constraints on vehicle resources, number and location of customers, service standards, etc.

In our research however, we limited our scope to routing a single vehicle. A route consists of nodes with time windows, precedence condition and capacity constraint. Given a cluster of passenger origins and destinations the aim is to find an optimal route to serve all the passenger requests. The single vehicle routing problem is similar to the travelling salesman problem which is also computationally complex..

Our approach developed a genetic algorithm for this vehicle routing problem. In the research one of the main difficulties was overcoming the infeasibilites that were generated after the crossover operation was performed. Other difficulties overcome included programming and implementation details such as representation of a route, memory management issues, etc.

5.2 Conclusions

Our conclusions address both theoretical and implementation issues. In solving the single vehicle routing problem we successfully applied GA. This application is the first documented use of GA for paratransit vehicle routing. A few conclusions about theoretical issues are presented below.

- We developed new modified crossover operator to handle the infeasibilities. The new crossover operator incorporates a repair mechanism that is based on adjacency relationship and minimum cost substitution. GA with modified crossover operator shows promise for paratransit vehicle routing. The idea of modified crossover operator can be used in other GA implementations. General delivery problem or school bus routing may also use a similar modified crossover operator.

- We also experimented with Feedback GA, in which GA reinitializes itself with the knowledge of the best solution from the previous run to search a better solution. This kind of approach may be useful in benchmarking a solution to a particular problem while experimenting with genetic parameters such as population size, probability of crossover, probability of mutation, and termination criteria etc.

- GA performs much better than simple heuristics. Simple heuristics provide a quick, deterministic way of arriving at a solution, but may result in very bad solution. We developed and experimented with two simple heuristic approaches namely,

In accomplishing the GA we demonstrated very practical, state-of-the-art programming technique. A few conclusions about implementation issues are presented as follows.

- The most natural way to represent a tour is through path representation. List or vector classes in Java make this representation easy to deal with.

- Java being an object oriented programming language, all the programs are objects that have attributes and methods related to that object. Also Java's documentation facility is helpful in understating and code building. It also facilitated reuse.

- Each program is a part of an overall package. This approach provides a very logical and structured way of storing files.

- Java's platform independence makes it a unique language and facilitates convenient code development. For example programs written for the purpose of research were partly written on a Unix system and partly on a Windows NT. Java documentation of the code can be found in the research locker: http://www2.ncsu.edu/eos/service/ce /research/stone_res/vnkarle_res/www/programs/java/packages/doc/index.html

- Java's Swing GUI components are platform independent and preserve their look and feel across different platforms. We developed a non-interactive GUI in this research.

- Today, the Internet provides a very effective and efficient way of communication. Hence we constructed a research web-site which describes the work we are doing as a part of this research effort.

5.3 Recommendations

GA with a modified crossover operator gives good results in a reasonable amount of time (dependent on the problem size). More experiments can be performed to find the appropriate parameters such as population size, number of generations, probability of crossover, probability of mutation, degree of elitism etc. for different problem size. This work can further be extended to the combined vehicle routing and scheduling problem. Conceptual work for representing a schedule is presented in this thesis (Section 3.7) and maybe extended in future research.

A different clustering strategy (for example simulated annealing) for scheduling may use genetic algorithm for routing. The idea of the modified crossover operator can be extended to problems where a similar repair strategy is needed.

In this research we used GA for the vehicle routing problem, which is a combinatorial optimization problem with competing objectives. Genetic algorithm is in general a good optimization tool to solve complex optimization problems and it's implementation is problem dependent. GA may also be used in conjunction with other heuristics for getting the desired quality of the solution. GA developed in this thesis uses simple stopping criteria. Further experimentation based on statistical properties of the solution can be done to devise better stopping criteria.

The GA developed in this research can be further fine tuned to different problem sizes. This indicated some more lab work. Once the GA parameters are finalized, it needs to be field-tested using real data. WSTA data can provide an excellent test-bed, as pre-processed reliable WSTA data can be made available. The results obtained with the field data using GA can be compared to the PASS routes that are produced.

There is also an opportunity for applying parallelism in the algorithm. Using Java's distributed environment, different Java Virtual Machines (JVM – Java interpreter) can be used to speed up the execution of the program. This concept can be further extended by

exploiting Java's internet compatibility to distribute computing across a huge network. The GUI can be enhanced to make it more user friendly and tailored to the needs of customers.

References

Barker, B. M. Further improvements to vehicle routing heuristics, *Journal of the Operational Research Society,* 43(10): 1009-1012, 1992.

Baugh, J. W., Kakivaya, G. R., and Stone, J. R. Multi-objective optimization of the dial-a-ride problem using simulated annealing, in J.P. Mohsen, editor, *Computing in Civil Engineering: Proceedings of the Second Congress.* American Society of Civil Engineers, 1995.

Bodin, L., Golden, B., Assad, A., and Ball, M. The state of the art in the routing and scheduling of vehicles and crews. *Computers and Operations Research,* 10:63-211, 1983.

Bott, K., and Ballou, R. H. Research perspectives in vehicle routing and scheduling, *Management Science,* 20A(3): 239-243, 1986.

Brill, E. D., Flach, J. M., Hopkins, L. D., and Ranjithan S. MGA: A decision support system for complex incompletely defined problems, *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20(4): 745-757, 1990.

Evans, S. R., and Norback, J. P. A heuristic method for solving time-sensitive routing problems, *Journal of the Operational Research Society,* 35(5): 407-414,1984.

Gendreau, M., Hertz, A., and Laporte, G. A tabu search heuristic for the vehicle routing problem. *Management Science,* 40(10): 1276-1290, October 1994.

Holland, J. H. Genetic algorithms: Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand, *Scientific American,* pages 66-72, July 1992.

Intelligent Vehicle-Highway Systems (IVHS) Primer, US Standards Catalog, Circular 412 (http://www.itsa.org/usstandcat.nsf/), 1998.

Kakivaya, G. R. *Computational Approaches to Intelligent Transportation System,* Ph.D. dissertation, North Carolina State University, Department of Civil Engineering, Box 7908, Raleigh, NC 27695-7908, April 1996.

Linden, P V. D. *just JAVA*$^{tm\ 1.1}$ *and Beyond*, Sun Microsystems Press – A Prentice Hall Title, 1998

Osman, I. H., and Kelly J. P. *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers, 1996.

Russell, R. A. Hybrid heuristics for the vehicle routing problem with time-windows, Transportation Science, Vol. 29, No. 2: 156-166, May 1995.

Solomon, M. M. On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time windows, *Networks,* 16: 6-17, 1986.

Stone, J. R., Winston-Salem transit authority mobility management: Enhancing service delivery through advanced public transportation technology systems. Draft Final Report Submitted to The Transit Administration by the Winston-Salem Transit Authority and the North Carolina State University Civil Engineering Department and Institute of Transportation Research and Education, October 1995.

Thangiah, N. R., and Nygard, K. E. School bus routing using genetic algorithms, *SPIE*, Applications of Artificial Intelligence: Knowledge-Based Systems, 1707: 387-398, 1992.

Uchimura, K., and Sakaguchi, H. Vehicle routing problem using genetic algorithms based on adjacency relations, *IEEE*, 214-217, 1995.

Wyner, P. Genetic Algorithms: Programming takes a valuable tip from nature, *Byte,* 16: 361-368, January 1991.

Zbigniew, M. Genetic *Algorithms + Data Structures = Evolution Programs*, Springer Publication, 1995.

APPENDIX A

Manhattan Distance v/s Crow-flight distance

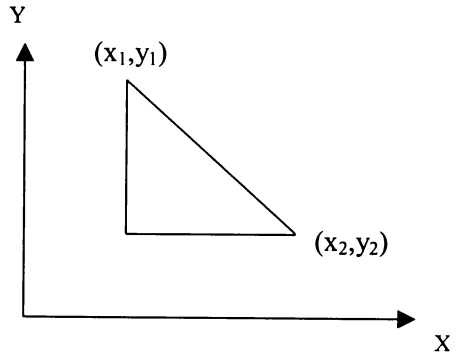
In our calculations, crow flight distance is multiplied by a factor of 1.5 as a rough estimate of distance between two points (including uncertainties such as traffic condition). We could also have used Manhattan distance as a means to estimate distance between two points. If (x_1,y_1) and (x_2,y_2) are two points then :

Crow flight distance between two points = $[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}$

Manhattan distance between two points = $|x_2 - x_1| + |y_2 - y_1|$

where $|x_2 - x_1|$ represents the absolute value of the difference between x_2 and x_1

For mathematical interest, we want to prove that 1.5*(Crow-flight distance) is more than the Manhattan distance.



Assume that, 1.5(Crow-flight distance) > Manhattan Distance

$1.5[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2} \geq |x_2 - x_1| + |y_2 - y_1|$

Squaring both sides,

$2.25[(x_2 - x_1)^2 + (y_2 - y_1)^2] \geq |x_2 - x_1|^2 + |y_2 - y_1|^2 + 2\,|x_2 - x_1|\,|y_2 - y_1|$

which is equivalent to,

$2.25[|x_2 - x_1|^2 + |y_2 - y_1|^2] \geq |x_2 - x_1|^2 + |y_2 - y_1|^2 + 2\,|x_2 - x_1|\,|y_2 - y_1|$

Rearranging terms,

$1.25\,[|x_2 - x_1|^2 + |y_2 - y_1|^2] - 2\,|x_2 - x_1|\,|y_2 - y_1| \geq 0$

$1.25[|x_2 - x_1|^2 + |y_2 - y_1|^2 - 1.6\,|x_2 - x_1|\,|y_2 - y_1|\,] \geq 0$

$|x_2 - x_1|^2 + |y_2 - y_1|^2 - 1.6\,|x_2 - x_1|\,|y_2 - y_1| \geq 0$

$|x_2 - x_1|^2 + |y_2 - y_1|^2 - 2\,|x_2 - x_1|\,|y_2 - y_1| \geq 0 - 0.4\,|x_2 - x_1|\,|y_2 - y_1|$

$[\,|x_2 - x_1| - |y_2 - y_1|\,]^2 \geq -0.4\,|x_2 - x_1|\,|y_2 - y_1|$

Above equation is always true for all real values of x and y.

Hence 1.5(Crow-flight distance) > Manhattan distance.

Now, let us assume 'k' such that $\{k[(x_2 - x_1)^2+(y_2 - y_1)^2]^{1/2}\}$ is greater then the Manhattan distance.

$k\,[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2} \geq |x_2 - x_1| + |y_2 - y_1|$

Squaring both sides,

$k^2\,[(x_2 - x_1)^2 + (y_2 - y_1)^2] \geq |x_2 - x_1|^2 + |y_2 - y_1|^2 + 2\,|x_2 - x_1|\,|y_2 - y_1|$

which is equivalent to,

$k^2\,[|x_2 - x_1|^2 + |y_2 - y_1|^2] \geq |x_2 - x_1|^2 + |y_2 - y_1|^2 + 2\,|x_2 - x_1|\,|y_2 - y_1|$

Rearranging terms,

$(k^2 - 1)\,[|x_2 - x_1|^2 + |y_2 - y_1|^2] - 2\,|x_2 - x_1|\,|y_2 - y_1| \geq 0$

$(k^2 - 1)\,[|x_2 - x_1|^2 + |y_2 - y_1|^2 - \frac{2}{k^2 - 1}\,|x_2 - x_1|\,|y_2 - y_1|\,] \geq 0 \qquad (\ k>0,\ k \neq 1)$

$|x_2 - x_1|^2 + |y_2 - y_1|^2 \geq \frac{2}{k^2 - 1}\,|x_2 - x_1|\,|y_2 - y_1|$

Adding $[-2\,|x_2 - x_1|\,|y_2 - y_1|]$ on both sides,

$|x_2 - x_1|^2 + |y_2 - y_1|^2 - 2\,|x_2 - x_1|\,|y_2 - y_1| \geq \frac{2}{k^2 - 1}\,|x_2 - x_1|\,|y_2 - y_1| - 2\,|x_2 - x_1|\,|y_2 - y_1|$

$[\,|x_2 - x_1| - |y_2 - y_1|\,]^2 \geq (\frac{2}{k^2 - 1} - 2)\,|x_2 - x_1|\,|y_2 - y_1|$

LHS ≥ 0 $(\forall x,y)$. Therefore, for this inequality to be satisfied $\forall x,y$,

$0 \geq (\frac{2}{k^2 - 1} - 2)\,|x_2 - x_1|\,|y_2 - y_1| \Rightarrow 0 \geq \frac{2}{k^2 - 1} - 2 \Rightarrow 1 \geq \frac{1}{k^2 - 1}$

Hence $k \geq \sqrt{2}$.

This implies that $\sqrt{2}$ (Crow-flight distance) $\geq$ Manhattan distance for all x and y.

APPENDIX B

Routing of a Vehicle – Complexity of the Computations Involved

Suppose there are 'n' customers that need to be served by a service vehicle. This implies that there will be '2n' stops on the route as each customer will have a pick-up and a drop-off associated with him. These '2n' stops can be arranged in *(2n)!* ways for routing. But there is a precedence condition associated with each customer, that is, a drop-off cannot be before pick-up.

If P_i and D_i represent the pick-up and drop-off for the i^{th} passenger then D_i cannot be before P_i . This means that the number of permutations will be reduced by two for each customer. Therefore, the total number of permutations possible are $[(2n)!/2^n]$. This can be proved by the principle of mathematical induction.

This formula is true for n = 1 and n = 2. Assume that it is true for any 'n'. We have to prove that it should be true for 'n+1' that is total number of permutations for 'n+1' customers are represented by $[2(n+1)]!/2^{(n+1)}$.

We add $(n+1)^{th}$ customer to already existing permutation of 'n' customers with the condition that P_{n+1} should be before D_{n+1}. There are '2n+1' slots where P_{n+1} can be added. If P_{n+1} is added at the first spot then Dn+1 can be added in 2n+1 spots. Similarly if P_{n+1} is added between 1^{st} and 2^{nd} spot then Dn+1 can be added in 2n spots and so on. Therefore there are 1+2+3+.....+(2n+1) = [(2n+1)(2n+2)/2] new permutations possible.

Therefore total number of permutations possible $= [(2n+1)(2n+2)/2]*[(2n)!/2^n]$

$= [2(n+1)]!/2^{(n+1)}$

Hence by the principle of mathematical induction, our formula is valid. For five customers, n = 5. Number of feasible permutations = $10!/2^5 = 113400$. This gives the idea of the computational burden of a single vehicle routing.

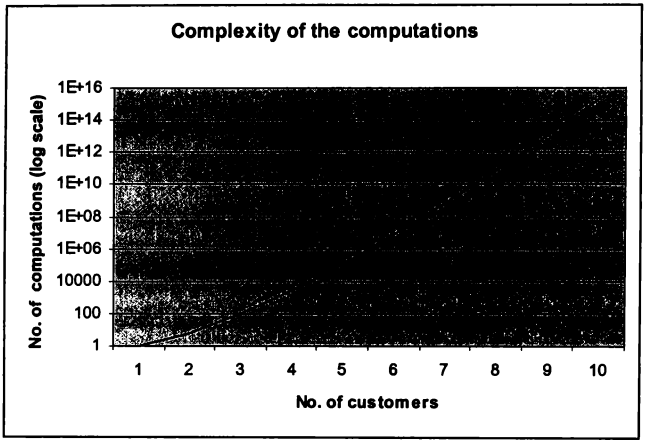


Figure B.1: Complexity of the Computations – Graph of Number of Computation (log scale) against Number of Customers

APPENDIX C

PASS Schedule Data Fields

The data fields in a PASS output file are listed below. Each data field is briefly explained depending on the information available. For more information please refer the PASS manual.

1. BUS: Route number (abbreviated) to which the trip was assigned. "0" indicates the trip is not assigned to a route. 999 indicate trips assigned to taxi.
2. LAB: Once a trip is routed, unique identification numbers are assigned by PASS for each individual stop. They display in ascending order for each route. The first three numbers identify the route.
3. RIDERS: Number of aides.
4. EQUIP: Equipment. If customer verification is "Y" in the system parameter file, PASS will insert the value from the CALLER file. If it is "N", enter the appropriate code (as defined in the System Parameter file) in this field. PASS may add to the pick - up and drop - off time, as recorded in the System Parameter file, depending on the type of special equipment needed (i.e. W - wheelchair, L - lift, O - oxygen, R - Ramp). PASS will also verify the vehicle performing the trip has the equipment and capacity necessary to fulfill this request. Enter single digit (codes) for each client's requirements. If two clients are travelling together and both require wheelchair tie downs, enter WW (or the equivalent code). Time will be multiplied for both chairs. If a code of WW is entered because one client requires both tie downs for a wide wheelchair, then you may want to minus additional time in the client record so that the time is not multiplied also.
5. ID: Trip ID (identification). No two trips have the same ID number on a given day unless more than 9999 trips were scheduled. Both the pick - up and drop - off have the same ID number. A return trip would have its own unique number. These are generally

used as confirmation numbers for the customers and for radio use by drivers and dispatchers.

6. CALL: The time trip was originally entered into the PASS system.

7. ETA: The time given to the client. It is the estimated arrival time of the vehicle at the location.

8. ETACALC: Estimated time of arrival of the vehicle as calculated by PASS as changes occur in the route (calculated for a p/u and d/o).

9. APPT: Appointment time. Time by which the customer is required to be at the destination. Keep in mind, entering an appointment time several minutes prior to the requested time will ensure the customer arrives in time to check in or to punch a time clock, etc. e.g. If a customer has a 09:00 appointment, enter the time as 08:55. An appointment time will display in the drop off record only when the early time feature is being used. It will display in red if the trip is running late. Should only be used for doctor appointments, school or work. If any entry has been made in the "Early" field (cannot leave earlier than this time) of the trip ticket or in the STANDING file, the early time will display in green at the pick-up (+) and the appointment time will display at the drop-off (-).

10. PD: Pick up (indicated by a '+' sign) or a drop off (indicated by a '-' sign).

11. ADDRESS

12. APARTMENT

13. CITY: The city in which pick up or drop off address is located.

14. BUSCOLOR: Bus number.

15. DATES: Date on which the trip is scheduled. If the effective dates in the CALLER record are outside of the date of the trip, a warning message will display. The same holds true if the client is suspended. A warning message will display indicating such and prompts the user to continue or terminate the trip request. If a day has not been yet opened, a warning message will come indicating so.

16. ADDLOADTIME: Time to load the passengers on bus. If customer verification is "Y" in the System Parameter file, PASS will insert the value from the CALLER file in the Load time field of the trip ticket. If set to "N" and there is knowledge that a certain client requires more or less than the average amount of time for loading or unloading,

enter a value to increase or decrease the average load time. Enter the estimated additional time needed at either the pick up (+) or drop-off (-) locations. This time is in addition to the time PASS automatically makes addition for apartments, special equipment, etc.

17. COLLECT: Actual time of pick-up or drop-off as recorded by driver.

18. XCOORD: X-Coordinate of pick-up or drop-off point.

19. YCOORD: Y-Coordinate of pick-up or drop-off point.

20. ROUTER: Initials of the router. AR and PRE indicate the trip came from the Standing Ride Database.

21. VEHICLE: The number of the vehicle servicing the trip. Identification number of the actual piece equipment that served the stop.

22. DRIVER: The individual driving the vehicle.

23. COMMENTS: Special information entered by the reservationist. Prints on the driver manifest.

24. DI: Tells you the trip status C-Canceled, P-Performed, Fare.

25. CAPCALC: Total number of persons on the vehicle at the completion of stop.

26. CAP1CALC: Total number requiring special equipment category 1 (defined in parameter file).

27. CAP2CALC: Total number requiring special equipment category 2.

28. CAP3CALC: Total number requiring special equipment category 3.

29. CUSTNUM: Identification number of the customer. For agencies not requiring registration, this field would be blank.

30. CUSTNAME

31. SLACK1: Time available to vehicle between consecutive scheduled trips. This time is commonly referred to as "slack" or "stand by" time and is truly the number of extra minutes available since drive time is already calculated.

32. PX: X-Coordinate of the previous stop of route.

33. PY: Y-Coordinate of the previous stop of route.

34. TRIPTYPE: W-Will call (entered with an invalid time such as 29:00) when the client does not know what time they will need the trip. These display in fuchsia, D-Demand responsive call (the client requests a particular day and time). These display in black, S - Standing (permanent) or Subscription trip. These display in light gray, B-Stand-by call

(trip is entered as unassigned in hopes that time will be made available through a cancellation or other means). These display in light blue, N-Voucher call/non-trip calls (trips being performed by another agency/company e.g. cab company), these display in dark blue, O-Out of service call, F-Fixed stops. Various denial types: These display in brown. C - denial based upon lack of capacity, E - trip that could be taken by the user on a fixed route, A - for adversarial.

35. FARETYPE: Code for the type of fare for this trip (e.g. ticket, token, cash). If Customer verification is "Y" in the system parameter file, PASS will insert the value from the CALLER file. If it is "N", enter a valid fare type as established by your agency in the Fare Type file (database). Entry of multiple codes is allowed for each client. PASS will calculate the total fee based upon this entry e.g. Three passengers may be entered as "FR", indicating one free and two regular passenger fares; "RF", indicating one regular and two free passenger fares; "RFC" indicating one regular, one free and one courtesy passenger). There is a limit of 5 codes within this field. PASS considers the last entry as the default for the remaining passengers in calculating the total fee.

36. ODOMETER: Direction of the vehicle.

37. TRIED

38. ETAEARLY: Takes precedence over all previously requested times, optional entry on user request. Enter a time not earlier than the specified appointment time. This time if entered supercedes any "windows" set in the System Parameter file and will affect the pickup time only for that particular trip. This time will eliminate any suggestions prior to the time indicated in the "Early Time" field. Using these suggestions may create a new "hostage" situation, causing the originally scheduled person to dwell onboard the vehicle long enough to wait for the pick up of an added passenger. If this hostage situation is more than 10 minutes, a new hostage message will appear as an Error Level 2.

39. NEXTLAB

40. BDY: The service area boundary in which the trip occurred. The field will be blank if there is only one service area.

41. CALLDATE: The date the trip was originally entered into the PASS system.

42. FS: The funding source for this trip, as defined in the Funding Source table.

43. NETNODE

44. AIDES: Number of companions (assistants) as taking this trip.

45. REASON: User - defined field. An entry may be recorded in this field to indicate the reason for a denial or a cancellation. Entry in this field is usually an "after-the-fact" function when something has required a change to the trip after entry and an attempt to route the trip is made.

46. PROVIDER: Identification of the agency providing the service for the trip (e.g. cab company). If customer verification is "Y" in the System Parameter file, PASS will insert the value from the CALLER file. If it is "N", enter a valid user-defined funding source as established by your agency in the Funding Source file (database). If utilizing the ADA Fixed Route Overlay (see Mapping Section for further information), an ADA funding source must begin with letters "ADA".

47. TRACKER: Identifies up to the five last individuals who have changed the trip (cancel, assign, edit, uncancel, move and unassign) by transaction type, initials and date.

48. SPEEDFCTR

49. ORIGETA: Original ETA if negotiations are made. PASS stores the original pickup time requested (+) by the client in this field, regardless of the eventual ETA that is agreed to the customer. If the negotiated ETA is greater than a user defined number of minutes in the ADA window FIELD in the System Parameter file, a prompt will display.

50. SATISFIED: This field prompts for a yes or no if the original requested ETA and the negotiated ETA is outside the defined ADA window setting from the Parameter file. A 'Yes' would indicate that the customer is satisfied even though it is outside the defined window and a 'No' would indicate that even though the customer has agreed to this trip they are really not satisfied with the service.

51. NEGETA: The actual time negotiated for a pick-up regardless of the time requested. The difference between the Original ETA and the Negotiated ETA is how close your agency is accommodating customer's requests. Negotiated times outside one hour (Federal maximum) or the amount indicated in the parameter file as ADA window will indicate a possible violation of your ADA requirements. A message will appear asking whether the customer is satisfied with negotiated time, since it is outside the defined ADA window.

52. PURPOSE: Purpose of the trip. It is a user-defined field.

53. FAREAMT: Amount of fare. Indicates the normal dollar amount to be collected for this trip, based upon the Fare type code entered or defaulted. The client may actually owe more or less depending on upon previously collected fares. A warning message displays at trip entry to indicate a previously collected fare balance over or under.

54. AREAS: Displays if a vehicle has been restricted to an area(s). This is the area of pick-up (+) or drop-off (-) location.

55. MAPGRD: If the map book overlay has been created, the map indicator (page number and grid ID) for the location will automatically fill in this field. This will print on the manifest (run sheet, log) as a reference to the composite map book being utilized by the drivers.

56. VERIFIED

57. DOORCURB

58. FARECOLL: Amount in dollars actually collected for trip.

59. ARRIVE: The time that driver arrives at a location but yet is not ready to leave. This is commonly referred to as dwell time. This is an optional entry by the dispatcher pressing the Z key when a driver indicates they have arrived at a location. The driver would then call again when they are ready to proceed. This data may also be entered by the driver through mobile data terminals.

60. PHONE

61. EARLYTIME

62. FAREBDY

63. AVLUPD

64. INADA

65. DATEFIELD

66. ZIPCODE

APPENDIX D

Different Types of Calls Handled at WSTA

WSTA handles different types of requests for providing service. These requests can be one of the following types:

1. Skeleton trip
2. Will call
3. Demand Responsive call
4. Requests by new customers – 14 day demand responsive policy

Each of the above request types are described below.

1. Skeleton Trips

Skeleton trips are the special type, which occur at regular interval of time. These may include visit to the doctor or going to work on a regular basis. About 90% of the trips scheduled by PASS are skeleton trips or subscription trips. These are indicated by 'S' on PASS display screen under the triptype column. No shows and cancellations occur frequently in these trips which upset schedules and require recalculations, causing reduction in operating efficiency as well as placing a burden on the scheduling/dispatching function.

Handling Skeleton Trips

Skeleton trips are dealt in advance. Suppose a client calls up WSTA requesting service for trips he/she has to make three times a week to the dialysis center. For each day the customer has an appointment time. The reservationist takes down the necessary information on the PASS screen. These may include address, name, destination, expected

pick-up time, expected drop-off and appointment time. As these information are entered, PASS gives several schedule options keeping in mind the appointment time. The dispatcher chooses one the best option. The estimated time of arrival (ETACALC) is calculated by PASS and this information is conveyed to the client. A twenty-minute time window frame on either side of the pick-up and drop-off is considered. Many operations are performed automatically by PASS, for example, fare calculation, time of arrival calculation. But PASS operates in a taxi like manner, it is possible that two persons starting at a common origin and going to the same destination end up in different buses. These types of situations are usually taken care of manually by dispatchers.

2. Will Calls

Will call is one of the valid trip types as given in PASS manual. In will call (open return), the client is not sure of the time for the return. They are displayed in fuchsia in PASS. They are indicated by 'W'' on PASS display screen under the column 'trip-type'. A will call can be skeleton (subscription call) or demand - responsive will call.

Handling the Will Calls

A will call is handled in a real time fashion by the dispatcher. When a client requests a return trip, the dispatcher looks through the schedules and finds out the appropriate bus to put the client on. In PASS will call trips are assigned route zero and the 'ETACALC' column is left blank. Will calls, as stated earlier, are displayed in fuchsia.

PASS has a way to find out if the will call trip was on time or not. On the On-Time Report, a call is considered on time in the column 'Will Calls on Time' if the collect time is less than CALL TIME + parameter. Currently the parameter is set to 3 hrs by WSTA, but they never go beyond 1 hr (as per the interview with staff). Call time is the time when client calls and says he/she is ready to go (e.g. if a client calls at 10:00 am to indicate that he/she is ready to go and parameter = 45 min, a pick-up time of 10:30 is considered on time). Note the parameter values are different for peak and off-peak hours. If the

dispatcher is unable to assign the passenger, taxicab service is called and route number for these trips is 999.

3. Demand-Responsive Calls

Besides regular passengers, whose trip information (i.e. origin, destination, pick-up and drop-off time, etc) is known to the PASS database, in case of Demand Responsive calls, a passenger with no subscription trip record calls up with a trip request. The trip may be accompanied by a return trip. The requested times are recorded in PASS as ETA i.e. estimated time of arrival. The system requires that all such requests be made at least one day in advance of the trip date. For each demand responsive call, PASS checks whether the caller is a valid passenger (registered with WSTA and travel being paid by authorized donors). Depending on the desired time of pick-up/drop-off, PASS suggests alternatives for accommodation of the new passenger trip. These alternatives are based on available slack time after the skeleton trip has been scheduled. The alternatives are ranked and usually the best one is selected.

4. New Customer -14 day Demand-Responsive Policy

A customer may be new to WSTA. He may even want to be a 'subscription' customer. The trip request for a particular day can be made before 14 days (till the last but one day). This means that the skeleton trips for that day already exist. So the subscription customer for first 14 days would not be on the skeleton schedule. For that period, he will be put on route 0.

APPENDIX E

Internet Resources Related to the Project

Site	URL
Project Programming Documentation	http://www2.ncsu.edu/eos/service/ce/research/stone_res/vnkarle_res/www/programs/java/packages/doc/index.html
Project Web Page	http://www4.ncsu.edu/~vnkarle/research.html
WSTA Information	http://www2.ncsu.edu/eos/service/ce/research/stone_res/mnsurask_res/www/homepage1.html
Genetic Algorithm	http://www.cs.purdue.edu/coast/archive/clife/FAQ/www/
Operations Research and Management Sciences	http://www.informs.com http://mail.informs.com
Operations Research related Java Objects	http://OpsResearch.com/OR-Objects/index.html
Java™ Platform 1.2 API Specification	http://java.sun.com/products/jdk/1.2/docs/api/index.html
Learning Java language	Http://www.javasoft.com:80/docs/books/tutorial/java/index.html
Java Soft Web-page	http://www.javasoft.com:81/
Transit Communications Interface Protocols (TCIP)	http://www.tcip.org/paper4.html
Computing Dictionary	http://wombat.doc.ic.ac.uk/foldoc

Note: The web page addresses listed above may change in the course of time. For more information about the project please contact Dr. John R. Stone, Professor, Civil Engineering, NC State University, USA (stone@eos.ncsu.edu)

APPENDIX F

Screenshots of the Documentation of Java™ Code

Documentation is produced using Java's 'javadoc'.

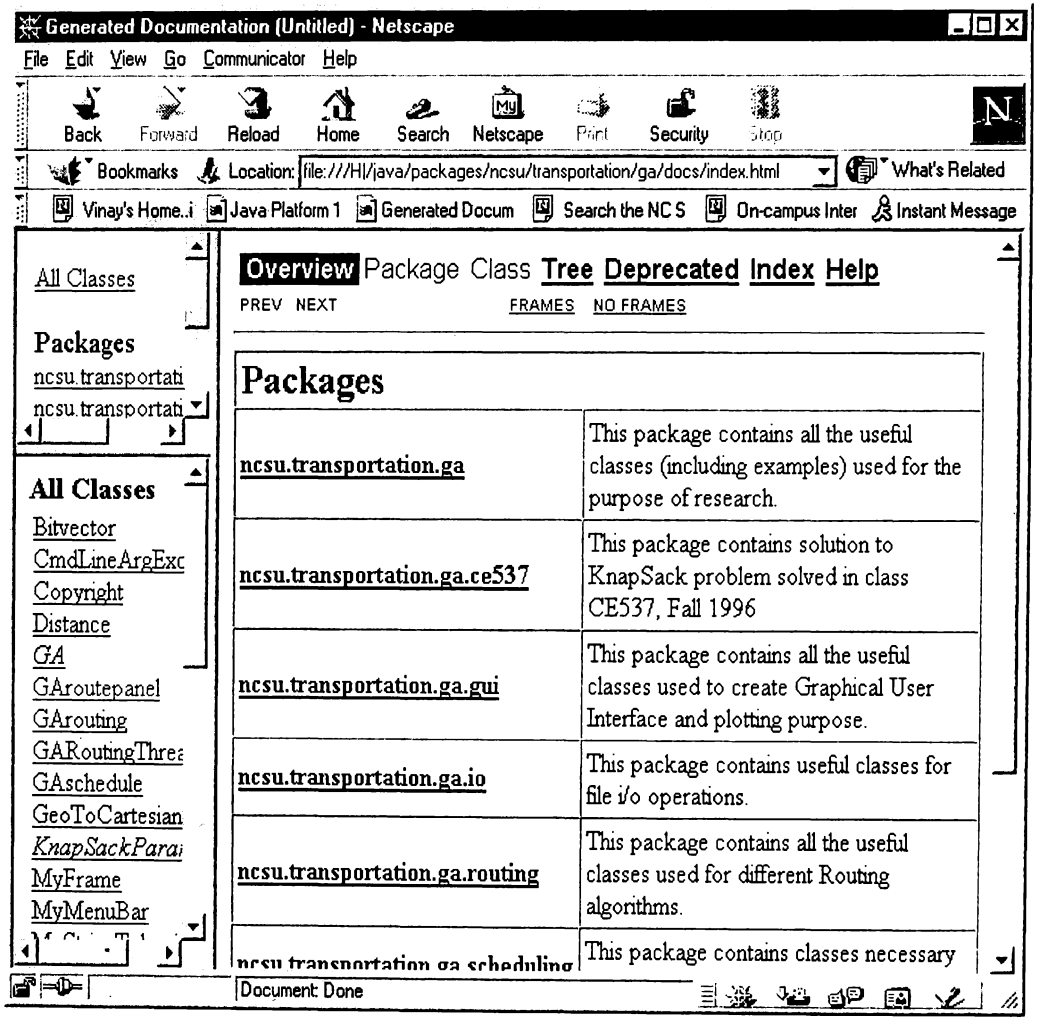


Figure F.1: Different Packages Created for Coding and Easy Understanding

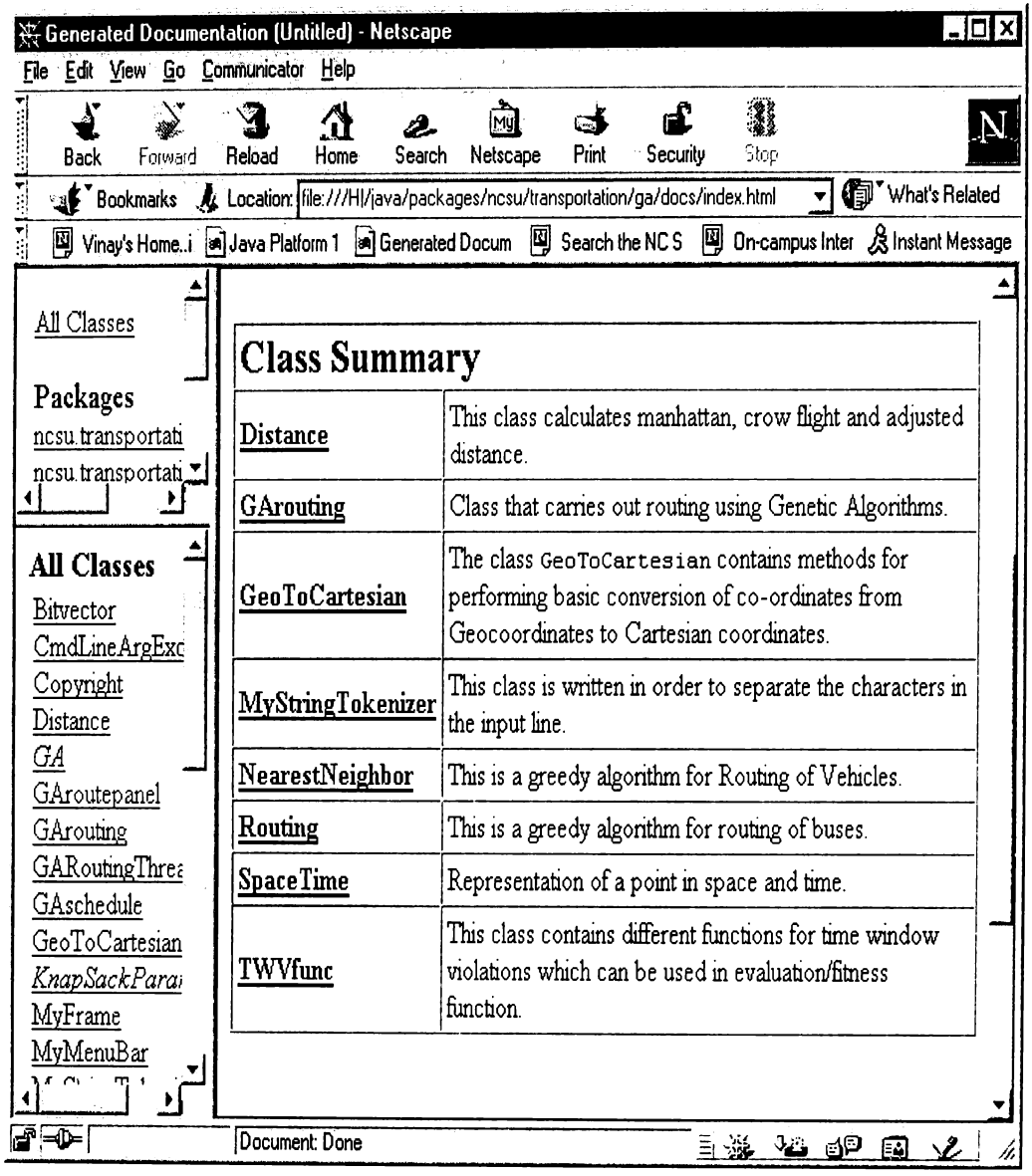


Figure F.2: Classes in Package Routing

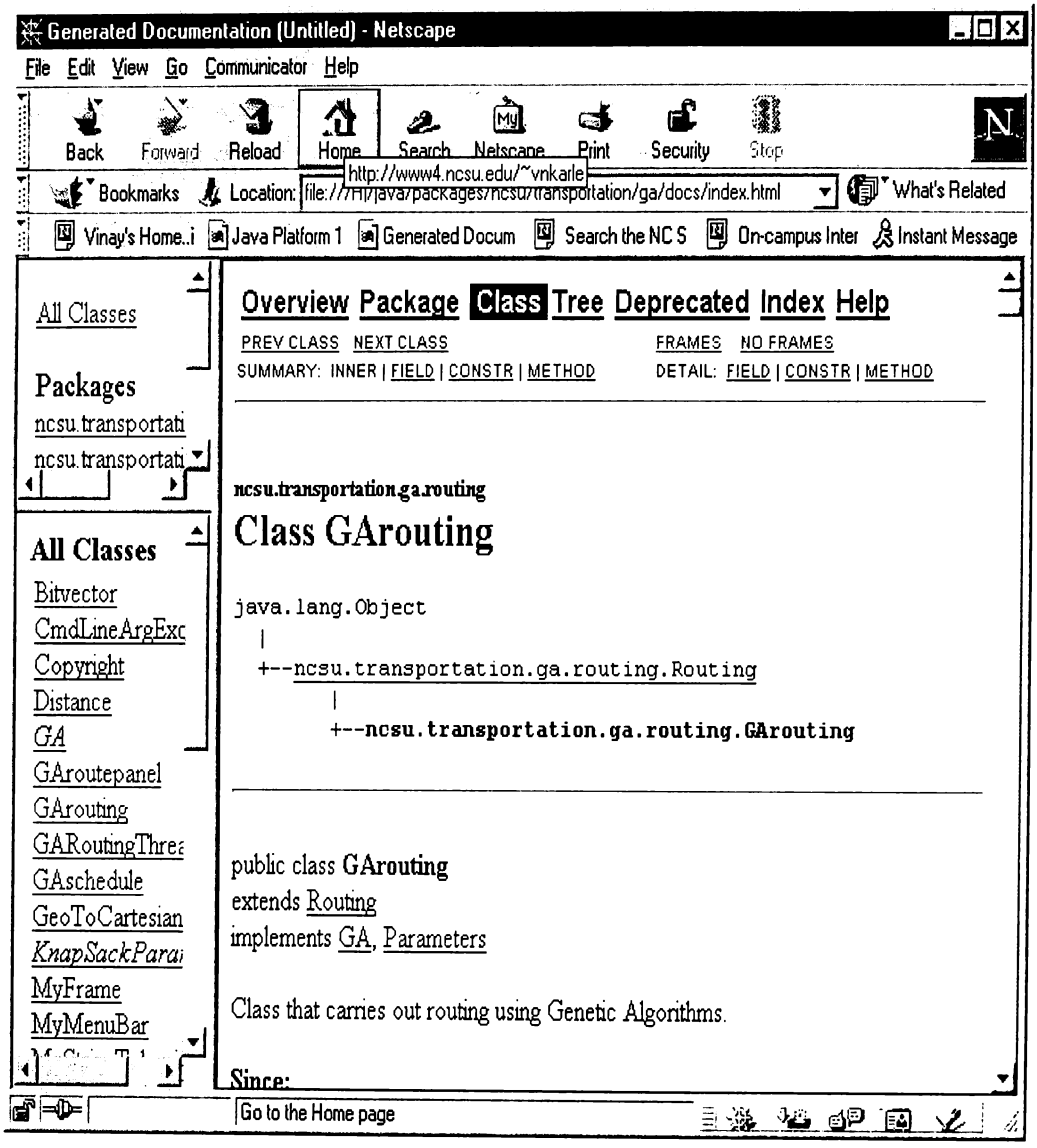


Figure F.3: Typical Documentation of a Java Program

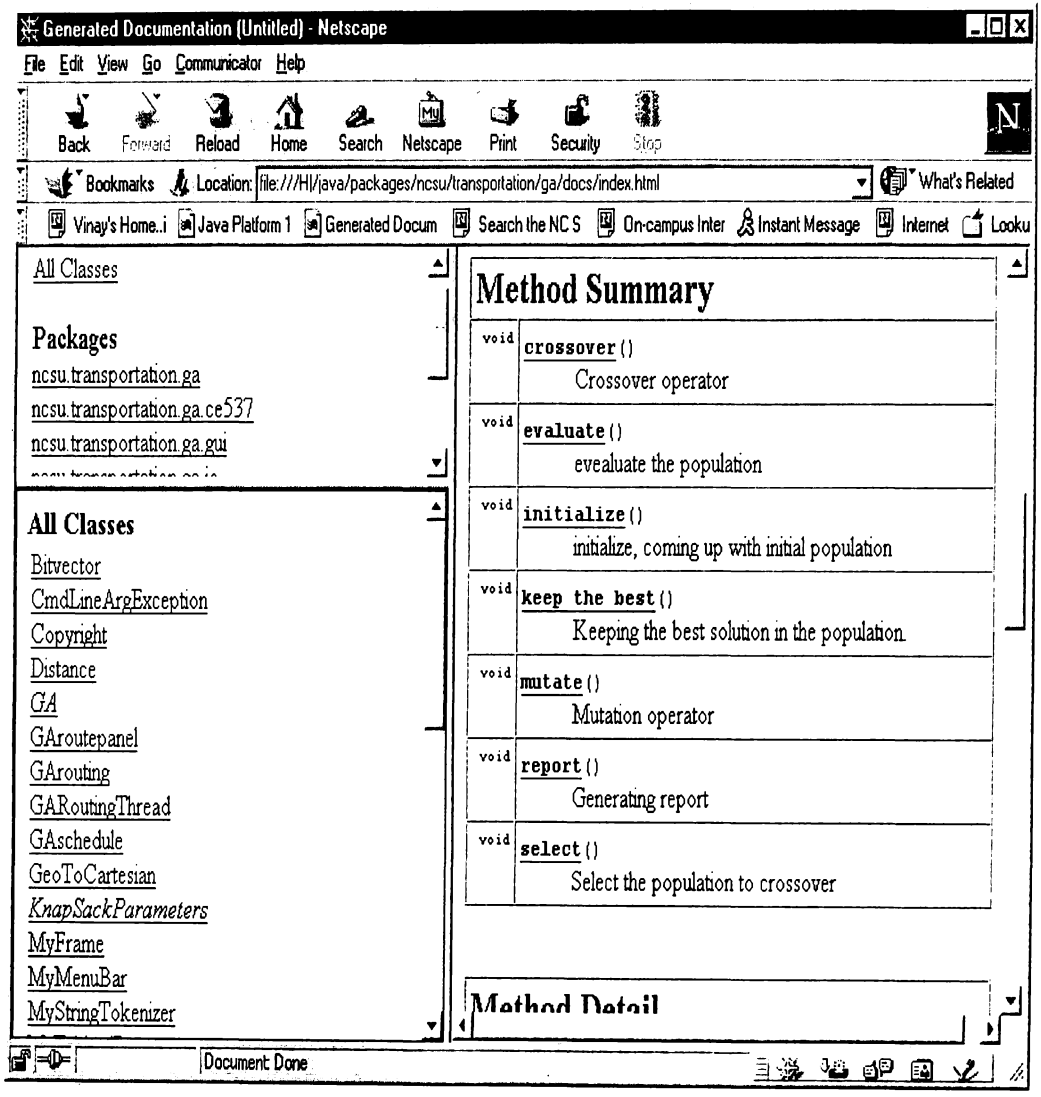


Figure F.4: Method Summary of a Typical Java Program

APPENDIX G

Screenshots of a Non-Interactive GUI Using Swing Components

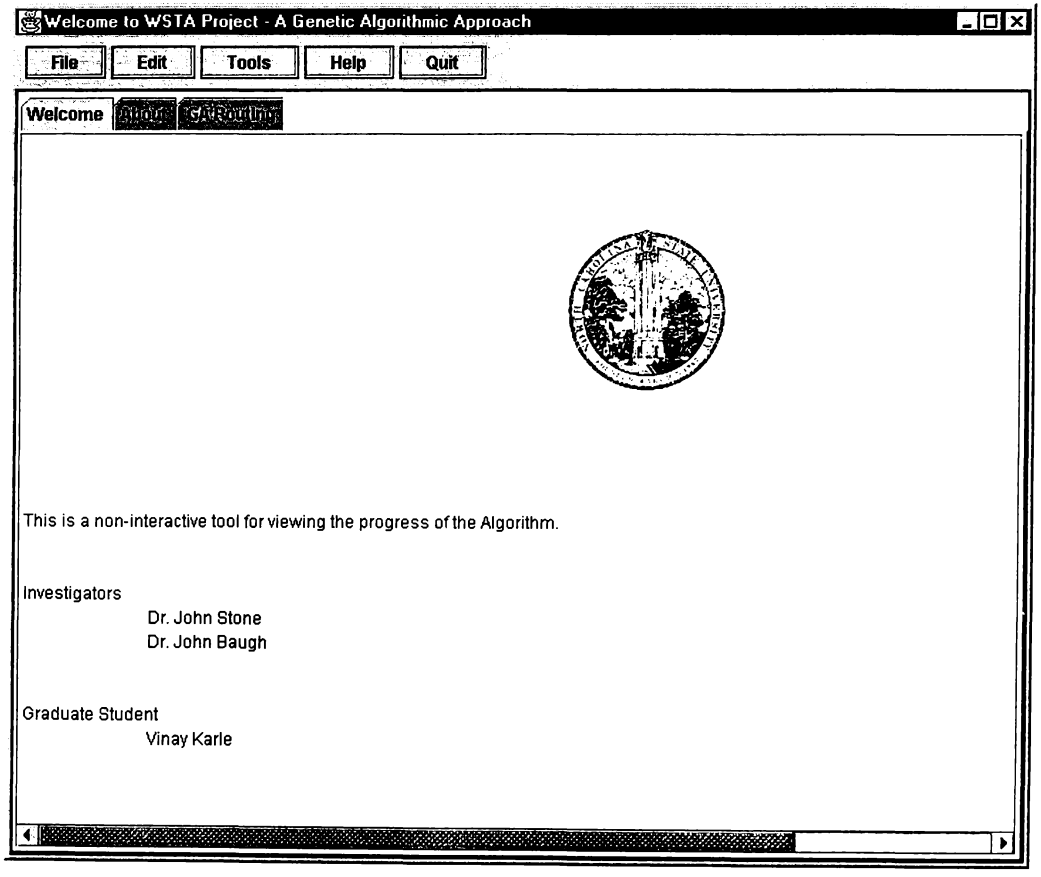


Figure G.1: GUI - Welcome Tabbed Pane

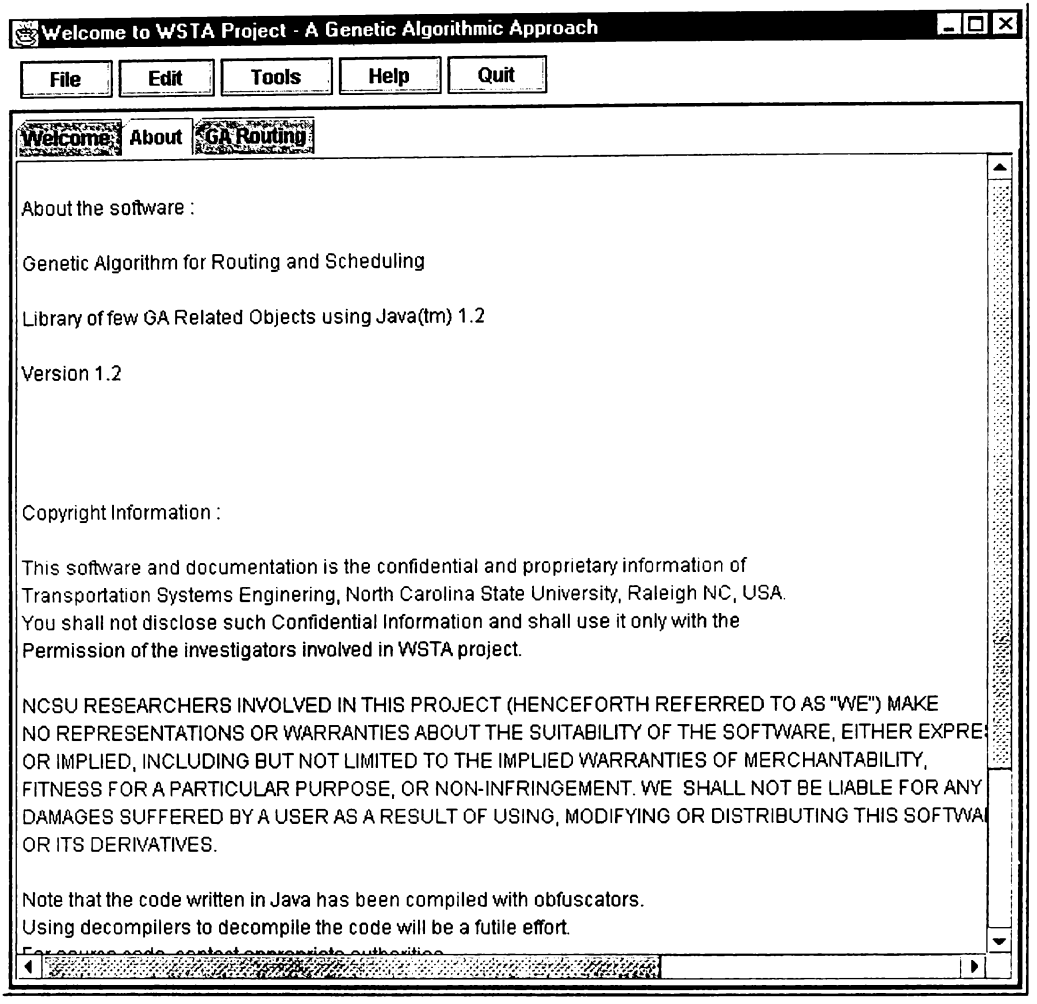


Figure G.2: GUI - Disclaimer Tabbed Pane

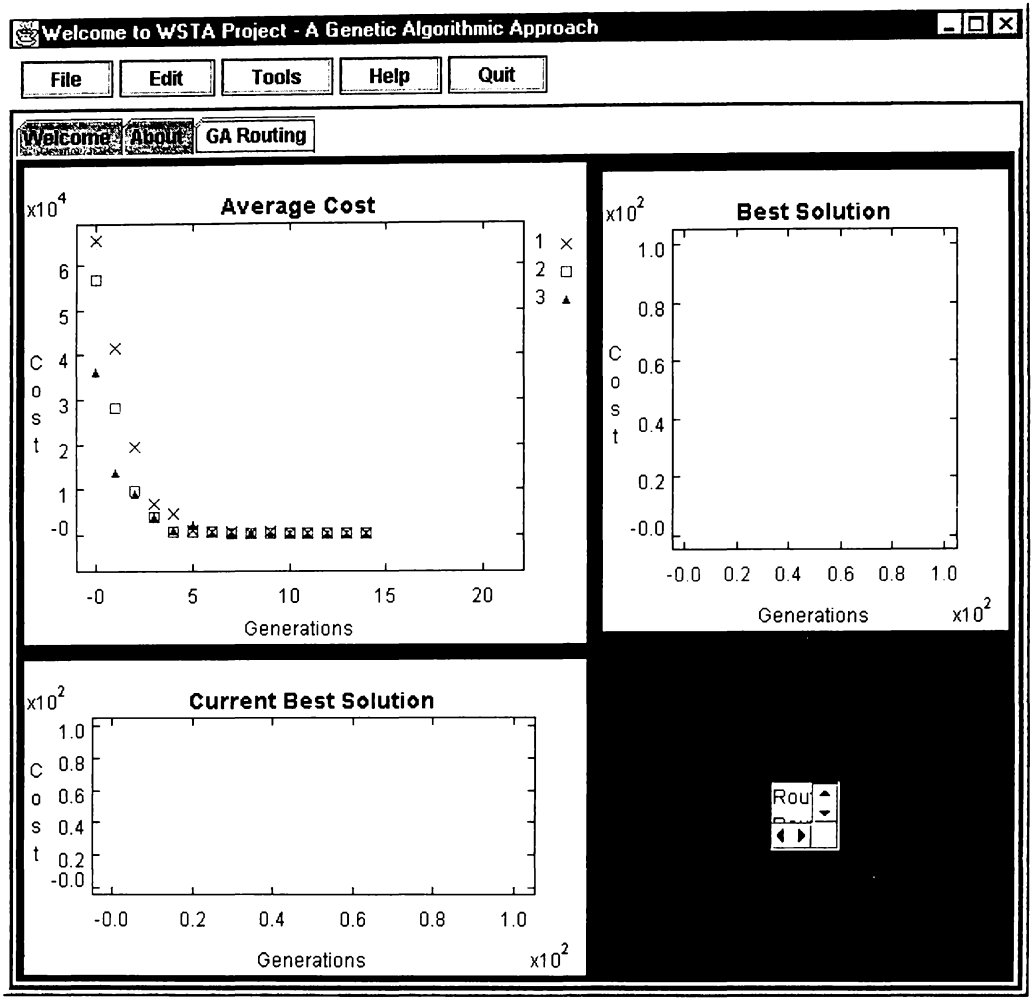


Figure G.3: GUI - Progress Monitor Tabbed Pane (1)

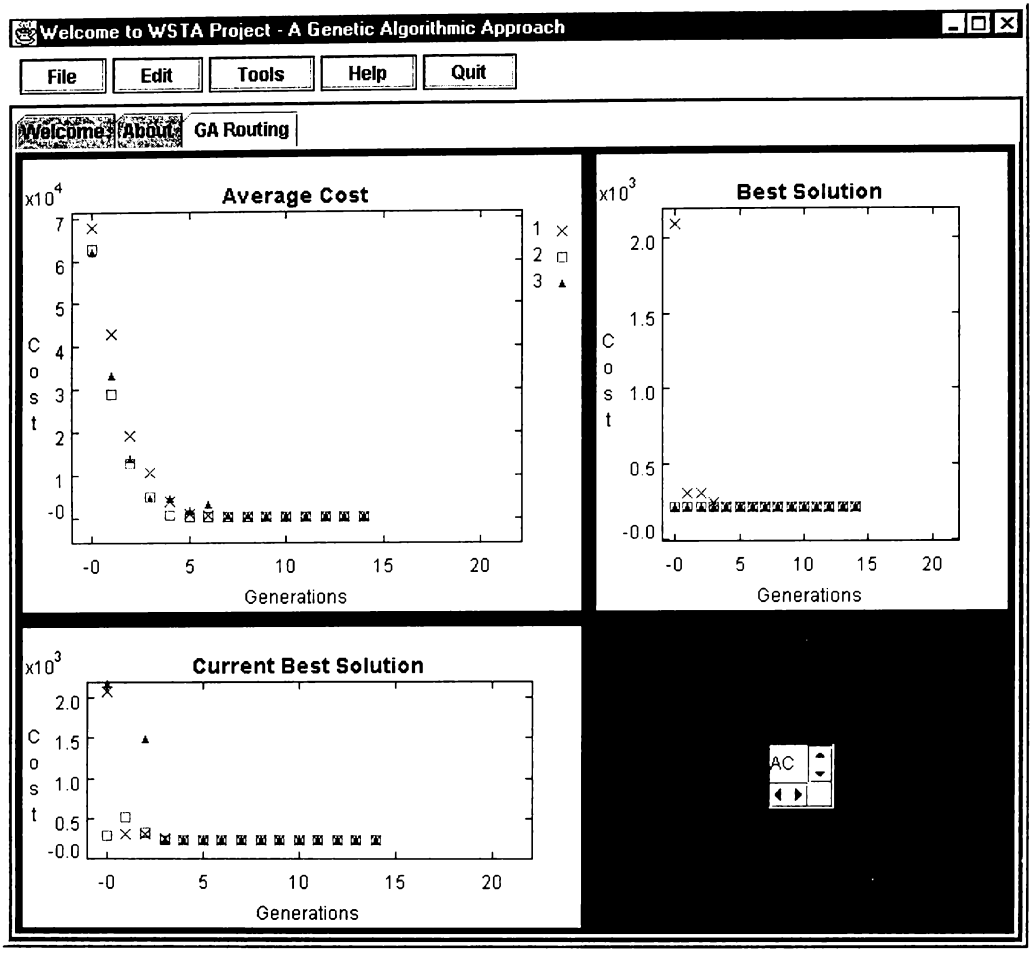


Figure G.4: GUI - Progress Monitor Tabbed Pane (2)

APPENDIX H

Sample Java Code with Comments for Java Documentation

Distance.java

```
package ncsu.transportation.ga.routing;

import java.io.*;
import java.lang.Math;
import java.util.StringTokenizer;
import ncsu.transportation.ga.Parameters;
import ncsu.transportation.ga.routing.SpaceTime;

/**
 * This class calculates manhattan, crow flight and adjusted distance.
 * This class serves as a library of distance calculation methods
 * @author  Vinay Karle
 * @version 1.2, 96-99
 * @since   JDK1.1
 */
public class Distance{

  /**
  * Distance factor. Initialized from Parameters file
  */
  public static double DISTANCE_FACTOR = Parameters.DISTANCE_FACTOR_CROWFLIGHT;

  /**
  * Returns distance as calculated by eucladian distance formula
  * @param first SpaceTime co-ordinate
  * @param second SpaceTime co-ordinate
  * @return eucladian distance (straight line)
  *
  */
  public static double CrowFlight(SpaceTime first, SpaceTime second){
    return java.lang.Math.sqrt((java.lang.Math.pow((first.x-second.x),2))+(java.lang.Math.pow((first.y-
second.y),2)));
  }

  /**
  * Returns adjusted distance (calculated, eucladian distance * distance factor
  * @param first SpaceTime co-ordinate
  * @param second SpaceTime co-ordinate
  * @return eucladian distance (adjusted)
  */
  public static double adjustedDist(SpaceTime first, SpaceTime second){
    return ((Distance.CrowFlight(first, second))*DISTANCE_FACTOR);
  }

  /**
  * Returns Manhattan distance (calculated, eucladian distance * distance factor
  * @param first SpaceTime co-ordinate
```

```
* @param second SpaceTime co-ordinate
* @return Manhattan distance (straight line)
*/
public static double Manhattan(SpaceTime first, SpaceTime second){
 return (java.lang.Math.abs(first.x-second.x)+java.lang.Math.abs(first.y-second.y));
}

/**
* Returns distance as calculated by eucladian distance formula
* @param x
* @param y
* @param x1
* @param y1
* @return eucladian distance (straight line)
*/
public static double CrowFlight(double x,double y,double x1,double y1){
 return java.lang.Math.sqrt((java.lang.Math.pow((x-x1),2))+(java.lang.Math.pow((y-y1),2)));
}

/**
* Returns adjusted distance (calculated, eucladian distance * distance factor
* @param x
* @param y
* @param x1
* @param y1
* @return eucladian distance (straight line)
*/
public static double adjustedDist(double x,double y,double x1,double y1){
 return ((Distance.CrowFlight(x,y,x1,y1))*DISTANCE_FACTOR);
}

/**
* Returns Manhattan distance (calculated, eucladian distance * distance factor
* @param x
* @param y
* @param x1
* @param y1
* @return eucladian distance (straight line)
*/
public static double Manhattan(double x,double y,double x1,double y1){
 return (java.lang.Math.abs(x-x1)+java.lang.Math.abs(y-y1));
}

public static void main(String args[]){
  SpaceTime x = new SpaceTime(3,3,"999983116","BCD","9:00",2,1,6,6);
  SpaceTime y = x.create_request();
  System.out.println("x "+x);
  System.out.println(" Crow Flight = " + Distance.CrowFlight(x,y));
  System.out.println(" Adjusted Distance = " + Distance.adjustedDist(x,y));
  System.out.println("Manhattan Distance = " + Distance.Manhattan(x,y));

  System.out.println("\n Crow Flight = " + Distance.CrowFlight(1,1,2,2));
  System.out.println(" Adjusted Distance = " + Distance.adjustedDist(1,1,2,2));
  System.out.println("Manhattan Distance = " + Distance.Manhattan(1,1,2,2));
 }
}
```