11-21-2006

Optimization of transit transfer times: a system-wide approach

Fabian Cevallos

Florida International University

DOI: 10.25148/etd.FI14060131
Follow this and additional works at: https://digitalcommons.fiu.edu/etd

Part of the Civil Engineering Commons

Recommended Citation
https://digitalcommons.fiu.edu/etd/2099

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact dcc@fiu.edu.
OPTIMIZATION OF TRANSIT TRANSFER TIMES:
A SYSTEM-WIDE APPROACH

A dissertation submitted in partial fulfillment of the requirements for the degree of
DOCTOR OF PHILOSOPHY
in
CIVIL ENGINEERING
by
Fabian Cevallos

2006
To: Dean Vish Prasad  
College of Engineering and Computing

This dissertation, written by Fabian Cevallos, and entitled Optimization of Transit Transfer Times: A System-wide Approach, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

__________________________________________
Albert Gan

__________________________________________
Shih-Ming Lee

__________________________________________
L. David Shen

__________________________________________
Fang Zhao, Major Professor

Date of Defense: November 21, 2006

The dissertation of Fabian Cevallos is approved.

__________________________________________
Dean Vish Prasad  
College of Engineering and Computing

__________________________________________
Dean George Walker  
University Graduate School

Florida International University, 2006
DEDICATION

I dedicate this dissertation to my parents Isabel and Gonzalo; my wife Kimberly, and my children: Melissa and Michael. Without their patience, support, and most of all love, the completion of this work would not have been possible.
ACKNOWLEDGMENTS

I wish to thank the members of my committee for their guidance, patience, and encouragement. Their gentle but firm direction has been most appreciated. In particular, thanks to Dr. Zhao, my dissertation advisor, for her valuable assistance with mathematical modeling, for taking the time to review the numerous drafts, and for providing very detailed comments. Thanks to Dr. Shen for his constant encouragement to complete this work. Thanks to Dr. Gan for listening and for providing constructive feedback on the content and format of the dissertation. Thanks to Dr. Lee for graciously accepting to be member of my committee and for his assistance during this process.

There are many others, whose names are not mentioned here, that in a way or another provided me support, reminded me that I needed to complete this work, or were part of my struggle during this very long path I decided to take. They will always be remembered.

A special thanks goes to Broward County Transit and their staff for providing the data and information needed to conduct the case study in this dissertation.

I have found my coursework and training throughout the Curriculum at FIU to be stimulating and thoughtful, providing me with the tools needed to succeed in my Transportation Career.
ABSTRACT OF THE DISSERTATION

OPTIMIZATION OF TRANSIT TRANSFER TIMES:
A SYSTEM-WIDE APPROACH

by

Fabian Cevallos

Florida International University, 2006

Miami, Florida

Professor Fang Zhao, Major Professor

This dissertation presents a system-wide approach, based on genetic algorithms, for the optimization of transfer times for an entire bus transit system. Optimization of transfer times in a transit system is a complicated problem because of the large set of binary and discrete values involved. The combinatorial nature of the problem imposes a computational burden and makes it difficult to solve by classical mathematical programming methods.

The genetic algorithm proposed in this research attempts to find an optimal solution for the transfer time optimization problem by searching for a combination of adjustments to the timetable for all the routes in the system. It makes use of existing scheduled timetables, ridership demand at all transfer locations, and takes into consideration the randomness of bus arrivals.

Data from Broward County Transit are used to compute total transfer times. The proposed genetic algorithm-based approach proves to be capable of producing substantial time savings compared to the existing transfer times in a reasonable amount of time.
The dissertation also addresses the issues related to spatial and temporal modeling, variability in bus arrival and departure times, walking time, as well as the integration of scheduling and ridership data.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTRODUCTION ............................................................... 1</td>
</tr>
<tr>
<td>1.1</td>
<td>Background ............................................................... 1</td>
</tr>
<tr>
<td>1.2</td>
<td>Problem Statement ...................................................... 2</td>
</tr>
<tr>
<td>1.3</td>
<td>Research Objectives ..................................................... 6</td>
</tr>
<tr>
<td>2</td>
<td>LITERATURE REVIEW .......................................................... 9</td>
</tr>
<tr>
<td>2.1</td>
<td>Transit Network Optimization .......................................... 9</td>
</tr>
<tr>
<td>2.2</td>
<td>Scheduling Optimization ................................................ 13</td>
</tr>
<tr>
<td>2.3</td>
<td>Other Relevant Approaches ............................................. 16</td>
</tr>
<tr>
<td>3</td>
<td>GENETIC ALGORITHMS ....................................................... 22</td>
</tr>
<tr>
<td>3.1</td>
<td>Overview ................................................................. 22</td>
</tr>
<tr>
<td>3.2</td>
<td>Considerations in GAs .................................................. 27</td>
</tr>
<tr>
<td>4</td>
<td>METHODOLOGY ............................................................... 30</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction ............................................................... 30</td>
</tr>
<tr>
<td>4.2</td>
<td>Mathematical Formulation of Transfer Time Minimization .... 33</td>
</tr>
<tr>
<td>4.3</td>
<td>GA Implementation ....................................................... 37</td>
</tr>
<tr>
<td>4.4</td>
<td>Incorporation of Random Bus Arrivals and Departures ........ 41</td>
</tr>
<tr>
<td>4.5</td>
<td>Data Requirements ....................................................... 47</td>
</tr>
<tr>
<td>4.6</td>
<td>Flowchart for Transfer Time Optimization .......................... 49</td>
</tr>
<tr>
<td>4.7</td>
<td>Computer Application .................................................... 49</td>
</tr>
<tr>
<td>4.7.1</td>
<td>Random Numbers ......................................................... 53</td>
</tr>
<tr>
<td>4.7.2</td>
<td>Calculation of Transfer Times ......................................... 54</td>
</tr>
<tr>
<td>4.7.3</td>
<td>Computational Procedures ............................................. 55</td>
</tr>
<tr>
<td>5</td>
<td>NUMERICAL EXPERIMENTS ............................................... 57</td>
</tr>
<tr>
<td>5.1</td>
<td>Study Area and Data Collection ....................................... 57</td>
</tr>
<tr>
<td>5.2</td>
<td>Modeling Schedule Deviations ......................................... 60</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Probability Distribution of Schedule Deviations ............... 61</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Simulation of Arrival and Departure Times ....................... 65</td>
</tr>
<tr>
<td>5.3</td>
<td>Random Brute Force Search ............................................ 66</td>
</tr>
<tr>
<td>5.4</td>
<td>Results from the Genetic Algorithm .................................. 72</td>
</tr>
<tr>
<td>5.5</td>
<td>Global Optimality of Solutions ....................................... 78</td>
</tr>
<tr>
<td>5.6</td>
<td>Monte Carlo Simulation ................................................ 80</td>
</tr>
<tr>
<td>5.7</td>
<td>Partial Optimization ..................................................... 82</td>
</tr>
<tr>
<td>5.8</td>
<td>Transfer Times of Best Performing Routes .......................... 90</td>
</tr>
<tr>
<td>6</td>
<td>SUMMARY AND CONCLUSIONS .......................................... 93</td>
</tr>
<tr>
<td>7</td>
<td>FUTURE WORK ............................................................ 96</td>
</tr>
<tr>
<td>TABLE</td>
<td>PAGE</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Probability Distributions (SAS 2003)</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Cross Tabulations of Transfers among Bus Routes</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Sample Locations for Modeling Schedule Deviations</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Correspondence between GA and Actual Route Numbers</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Algorithm Parameters</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Implementation Plan of Timetable Shifting</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>GA Implementation Plan of Timetable Shifting</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Monte Carlo Results</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Optimization of 10 Routes</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>GA Sample Results</td>
</tr>
<tr>
<td>Table 5.10</td>
<td>Top 10 Optimized Route Pairs</td>
</tr>
<tr>
<td>Table 5.11</td>
<td>Operating Characteristics of Top 13 Routes</td>
</tr>
<tr>
<td>Table 5.12</td>
<td>Ranking of Routes by Their Involvement in Transfer Time Savings</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>Figure 1.1</td>
<td>5</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>7</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>38</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>40</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>44</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>50</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>51</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>56</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>59</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>62</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>64</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>64</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>66</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>67</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>69</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>70</td>
</tr>
<tr>
<td>Figure 5.9</td>
<td>71</td>
</tr>
<tr>
<td>Figure 5.10</td>
<td>73</td>
</tr>
</tbody>
</table>
Figure 5.11 Number of Chromosomes and Generations ........................................... 74
Figure 5.12 Crossover Rates ..................................................................................... 76
Figure 5.13 Mutation Rates ....................................................................................... 76
Figure 5.14 Impacts of Population Size ..................................................................... 77
Figure 5.15 Variations of Gene Values ..................................................................... 79
Figure 5.16 Convergence of GA with Monte Carlo Simulation ............................... 81
Figure 5.17 Possible Combinations of Selected Routes ............................................ 85
Figure 5.18 Transfer Times Savings of Top 13 Routes ............................................ 91
Figure 5.19 Results of Using Top 13 Routes as Initial Population ............................. 92
CHAPTER 1
INTRODUCTION

1.1 Background

A main goal of a transit agency is to provide fast, reliable, and convenient services to customers. As transit agencies attempt to provide better transportation solutions to the public to deal with the ever worsening congestion problems, their ability to achieve these goals continues to be limited by resources. Under financial constraints, efficient and low cost solutions that may be implemented quickly are needed to assist them in their effort of improving services. Short-term solutions must be feasible to be incorporated into the operations of a transit agency without demanding significant resources or changes in organizational structures. Long-term approaches with a comprehensive scope also need to be considered to provide additional alternatives for improvements in both effectiveness and efficiency. The following are some of the areas that transit agencies may focus on to improve services:

- optimizing transit network,
- improving headways by increasing service frequency,
- reducing travel times,
- increasing service span to cover more hours during a day,
- improving on-time performance,
- reducing times that passengers spend on transferring from one vehicle to another,
- providing faster and more convenient fare transaction mechanisms, and
- improving access and egress to make it easier to reach transit stops or stations.
One way to improve transit services that reduces travel time and improves comfort is by reducing the number of transfers or minimizing transfer times. While reducing the number of transfers will require the modifications of an existing transit network, minimization of transfer times for a given network configuration may be accomplished by adjusting transit schedules to achieve better schedule coordination among different transit lines such that the waiting time during transfers is minimized.

A coordinated system not only will provide better services to existing transit customers, but will also help improve the image of public transit agencies. An optimized transit service will be more attractive to transit users and will make transit a more attractive alternative to transit choice riders, who own cars thus having other alternatives to travel. Furthermore, being able to encourage these choice riders to switch from automobiles to public transit will have a positive impact in reducing traffic congestion. Therefore, a coordinated transit system will be beneficial to both the general public, as well as, transit agencies.

1.2 Problem Statement

As many urban areas, including those in South Florida, continue to grow in population, and as congestion continues to worsen, it has become increasingly recognized by the public and elected officials that construction of new highways to solve congestion problems is becoming unrealistic because of the high costs and significant negative impacts on the urban environment.
Future demands for transportation can only be met by mass transportation and by efficiently managing the entire transportation system. Therefore, the design of an optimal mass transportation system is of utmost importance.

A transit system may be optimized in many ways, including network layout, headway design, schedule design, etc. Minimizing transfer time through schedule coordination is one means to reduce the total travel time for those passengers who need to transfer from one route to another to complete their trips and to minimize the anxiety associated with transfers. However, providing coordinated transit services, in which transfer times for the entire transit system are minimized, is not easily achieved. This is mainly due to the complexity and combinatorial nature of this particular scheduling problem. Due to a lack of efficient techniques to create timetables that result in optimal coordinated services, coordination is in most cases only provided at selected transfer locations, usually at major terminals. However, minimizing transfer times for all possible connections will provide greater benefits to the users of the transit system. To provide better services, transit agencies need improved analysis techniques, such as for the creation of timetables that reflect a coordinated transit system.

Optimization of transfer times is a subject that has been studied to some extent. However, the classical optimization models such as the Mixed-Integer Programming (MIP) and Mathematical Programming (MP) are not well-suited to this type of problems, which often include a large number of binary and discrete variables with large data sets, which results in combinatorial conditions.
To illustrate the complexity and computational intensity of the problem, assume a relatively small transit system with 40 directional routes, 10 trips per route, and 30-minute headways for which the transfer times are optimized by shifting the times in the timetables. For each route, there will be 30 possible time shifts if one-minute shifts are allowed. Therefore, for 40 routes, the number of combinations of all possible timetable shifts will be $40^{30} \approx 1.15E+48$. This is a number of astronomical magnitude, and it is impractical to find a global optimal solution based on exhaustive search or mathematical programming techniques, the latter is due to non-convexity of the problem. Additionally, the computational complexity increases when other variables such as all the possible transfer locations, ridership demand, random arrivals, etc., are considered.

The total number of possible transfer connections at one location depends on the number of routes that meet at that location. Mathematically, this is represented as $4N_R(N_R - 1)$, where $N_R$ is the number of routes that meet at that transfer point. For instance, Figure 1.1 illustrates the possible transfer combinations from route 1 to route 7. In other words, when two routes meet, there are eight possible transfers $4 \times 2 \times (2-1)$. However, when more than two routes cross at a transfer point, which is the case for transit terminals, the numbers of possible transfers increase significantly. For instance, if five routes pass through a single transfer point, the total number of connections will be 80 (i.e., $4 \times 5 \times (5-1)$).
Recently, there have been interests in solving this type of scheduling problem using evolutionary algorithms since they present a great opportunity in solving nonlinear, nonconvex, and combinatorial search problems. Among the evolutionary algorithms are the genetic algorithms (GAs). GAs are well known for their global search capabilities and their potentials for implementation on parallel computing platforms, which may significantly improve the speed of search (Haupt and Haupt 1998). However, previous research has been limited to using simplified networks or theoretical frameworks, or has not considered operational issues. In particular, there is a need to address how to handle large data sets that include the whole transit network efficiently, use operational data like
timetables and ridership, and incorporate the optimization results into the scheduling process.

1.3 Research Objectives

This research attempts to solve the transfer time optimization problem by developing a methodology based on genetic algorithms. The research concentrates on the coordination of schedules to reduce the time spent during transfers by searching for optimal combinations of bus departures and arrivals in a fixed-route transit system. It considers all trip combinations for all the timetables without modifying existing headways. The intent is to use this methodology as part of the timetable development in which transfers are optimized without having a major impact on the amount of service provided by the agency. While approaches that involve headway modifications (e.g., Deb and Chakroborty 1998) have practical applications, they may not be suitable without sensitivity to other design, operational, and local considerations. Furthermore, transit agencies in the U.S. cannot easily modify their headways without impacting contract labor rules, analyzing resource reallocation, and addressing political jurisdiction and equity issues. Because of these factors, there is the risk of the headway modifications designed to reduce transfer times not being accepted by the public or elected officials, which will defeat the purpose of the plan to improve transit services.

The approach proposed in this dissertation is to be used at the operational level to provide an analytical tool that may be incorporated in the scheduling process, without having to address funding, labor, and political influences. Figure 1.2 below depicts the basic steps
of the scheduling process on how transfer optimization may be incorporated into it. It includes the following: 1) schedule writing, which is the process of developing the service schedule; 2) block building, which is the process of creating vehicle assignments; and 3) run-cutting, which is the process of developing driver assignments. Transfer Optimization may be performed as part of the development of service schedules and the creation of public timetables.

*The National Transportation Communications for ITS Protocol (NTCIP) 1404

Figure 1.2 Scheduling Process

The optimization algorithms developed in this dissertation attempts to determine the best solution based on available computer resources. The effect of irregularity of bus arrivals is addressed. Special attention is given to finding optimum search techniques and
practical procedures that may be successfully implemented. GIS is used to display the transit network configuration and to display transfer locations.

The research attempts to achieve the following objectives:

- Explore search techniques and computational methods that may be used as part of the genetic algorithm.

- Address issues regarding data requirement, especially timetables, transfer locations, ridership demand, and schedule deviations.

- Develop and evaluate the transfer time optimization methodology in terms of time saving improvements.
CHAPTER 2

LITERATURE REVIEW

Transit agencies may benefit significantly from using optimization methods to improve their services without incurring significant demand for additional resources. Transfers are one of those areas that present great opportunities for optimization.

This literature review investigates research in the area of transit transfers. It includes research that deals with the optimization of transit networks to minimize the numbers of transfers (Section 2.1), the optimization of transit schedules to reduce transfer times (Section 2.2), and other relevant research (Section 2.3).

2.1 Transit Network Optimization

A key element for building transit ridership is the optimal design of bus route network. This section deals with minimizing the number of transfers by modifying an existing transit network configuration based on a set of operational and financial constraints. Such constraints impact the overall travel time as well as service coverage. Within this framework, it is important to point out that transfer optimization may be limited to minimizing the number of transfers, or it may be part of a more comprehensive optimization analysis of the entire transit system that also includes schedule optimization, user cost minimization, operator cost minimization, or a combination of them. Yet, if minimization of transfers occurs within a larger framework that includes the entire transit system, the computational complexity is significantly increased, making it difficult to be solved efficiently using the existing methodologies and computing resources.
In general, analytical and numerical solutions to route optimization have been investigated at some extent. Ceder and Wilson (1986) presented a two-level approach methodology that considered the passengers viewpoint as well as the passengers and operator impacts. An algorithm was used to design the feasible bus routes. The authors mentioned that to establish a planning process that is accepted, the methodology needs to be simple, flexible, and practical with considerations to operational elements. Accordingly, they presented an approach that was relatively easy to implement, practical, and sensitive to risks involved with route changes.

Van Nes et al. (1988) considered providing good transit services to be maximizing the number of passengers with a given budget. The problem of network design was formulated as finding “which routes and which frequencies should be offered to fulfill the demand for public transport as well as possible, given a certain available budget”. They recognized that providing a service “as well as possible” is a political decision, and the main objective was to maximize the number of passengers who can travel without transfers.

Constraints of this model included: the available budget, number of vehicles per vehicle type, a restriction on the frequency values to make it easy for passenger to remember, and a limit on the number of vehicles than can be assigned to a route. The optimization model was tested with a fictitious network consisting of 75 routes, considering demand patterns during mid-day and peak-hour periods. They also tested an actual network, consisting of 182 nodes and 115 zones, using three different scenarios and demand
patterns for the morning peak. An increase of 300 direct trips was obtained. The authors concluded that the methodology provides good results and that future research would be focused on creating a model that includes a set of all possible routes.

Even though the Transit Network Design Problem (TNDP) has been studied to some extent, formulations of the TNDP problem vary substantially depending on the variables as well as the number of constraints considered. Different solutions to the transit network design problem have been attempted. Baaj and Mahmassani (1992) stated that the goal of the TNDP is to determine a configuration that consists of a set of transit routes and associated frequencies that achieves a desired objective, based on certain constraints. The existing formulation may be viewed as variants of a mathematical problem, the main goal of which is to minimize the overall cost for both the user and the agency. They discussed the limitation of using the TNDP and presented potential search techniques using artificial intelligence to generate better solutions.

The complexity of transit networks is mainly due to the huge number of possible solutions. Conflicting objectives create additional complexity. For instance, if the goal is to minimize the network travel time, it cannot be expected to maximize coverage and accessibility as well. In their paper, two methods for designing the alignment of a transit route were investigated using TransCAD as the GIS software and data from Logan, Utah. The first method involved the use of a demand model that considered socioeconomic and demographic variables to create a generalized impedance function, which was later used in determining the best transit route alignment.
The second method used origin-destination data to improve transit coverage for large employment centers (Ramirez and Seneviratne 1996).

Most of the solutions to the transit network problem have been solved using mathematical formulations, of which the objective function is to minimize either user or operator costs or a combination of both (Shih et al. 1997). User costs include access costs, waiting costs, transfer cost, and travel time cost. Operator costs include service miles and hours of operation. Constraints include minimum frequencies, maximum loads, and maximum fleet size. The complexity and the combinatorial nature of the TNDP prevent these problems to be solved by exact mathematical optimization methods. Shih et al. (1997) pointed out that other approaches that use heuristic algorithms could not guarantee global solutions and that most of the heuristic approaches dealt with the TNDP separately as route design problem and frequency determination problem.

Zhao et al. (2004) developed a methodology to minimize the number of transfers while optimizing route directness and maximizing service coverage. A comparison of the different approaches used in the design of transit networks was presented as part of the literature review. The methodology combined a local greedy type search with simulated annealing and tabu search to improve the effectiveness and efficiency of the search for the global optimum solution. Simulated Annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space. The methodology was tested with published benchmark problems and was applied to a large-scale transit
network design problem based on the Miami-Dade County Transit System, which consisted of 81 routes, over 4,300 street segments, and 2,804 street nodes. The results show the methodology was able to improve the existing network in a reasonable amount of computing time.

### 2.2 Scheduling Optimization

Like network optimization, transfer time optimization is another way to improve the efficiency of a transit system. Rapp and Gehner (1976) presented a graphical approach to transit planning. The researchers dealt mainly with transfer delay optimization as one of the steps in a four-stage interactive model. By using an iterative procedure, transfer optimization was obtained by modifying the terminal offsets to reduce system-wide transfer delays. A relatively small case study during off-peak hours was conducted and showed promising results. However, the approach was only able to find local minima and the solutions were dependent on the starting point as well as the order in which the routes were considered in the iterative process. Further, due to modification in arrival and departure times, issues such as variation of layovers and vehicle requirements arose, which added complexity to the model due to labor requirements and operation costs issues.

Bookbinder and Desilets (1992) presented a good foundation on this subject from the operational aspect. The paper addressed several issues such as holding and no-holding policies, stochastic bus travel times, and the combination of simulation and optimization. They proposed two ways to reduce transfer times: 1) decreasing the headways on
particular routes and 2) providing transfer coordination. Additionally, timed transfers
was described as a scheduling strategy with which buses are to meet at certain locations
within a time window and layover is usually added at these locations to make certain that
connectivity occurred. However, in transfer optimization, buses do not necessarily have
to meet at these locations. Instead, buses are scheduled in such a way to minimize the
inconvenience of all passengers in the transit network. It was suggested that more
research was needed in this area, particularly to see which transfer policy would offer
greater benefits and whether deterministic travel times could have negative impacts in the
optimization of transfer times.

The model proposed by Bookbinder and Desilets (1992) assumed constant headway,
which allowed the use of offset times as the only controlled variable, and departure times
were derived based on this variable. The authors used a procedure that changed the
offsets for the first trip of the line one route at the time until no further improvement
could be obtained. It was stated that the benefits of transfer optimization would diminish
when the randomness in bus arrivals increased. Therefore, a key element in the success
of transfer time optimization is to ensure that buses arrive according to schedule, or as
close as possible. The authors recommended that future research should consider
changes in departure times not only for the first trip on each line, but for all the trips.
They also recommended that the model be improved by considering variable headways.

Chakroborty et al. (1995) also studied the optimal schedule problem. Their study was
aimed at finding a schedule that provided the best level of service with the available
resources. The authors pointed out that one way to achieve this was to minimize the waiting time for both transferring passengers and for first time boarding passengers. They reported that although there had been attempts by Rapp and Gehner (1976) using simulation and by Bookbinder and Desilets (1992) using an optimization model and simulation, only optimization of transfer times had been considered. They stated that because of the nature of the problem, conventional methods such as mathematical programming were not suitable because even for a small transit network it was an extremely difficult task (Kikuchi and Parameswaran 1993).

Due to the nature of the problem, mixed integer programming have been attempted, using a combination of binary, real, and nonlinear terms in both the objective and the constraints. However, researchers such as Ceder and Tal (1997) have mentioned that when dealing with large networks, using MIP to solve this scheduling problem was impractical. Researchers have concluded that using the classical methodology, the number of variables and constraints become very large even for small problems (Kikuchi and Parameswaran 1993, Deb and Chakroarty 1998, Chakroarty et al. 1995, and Ceder et al. 2001). To solve this problem, Chakroarty et al. (1995) used the genetic algorithm to find optimal schedules within existing computational constraints. It was stated that the success of GAs is mainly due to their computational efficiency.

Shih et al. (1997), Dessouky et al. (1999), and Kikuchi and Parameswaran (1993) investigated the design of coordinated timed-transfer for transit systems at transit centers, which were to operate as main hubs. Routes at transit centers may be synchronized to
allow buses to arrive within a time window of each other and thus minimizing transfer times. Shih et al. (1997) presented three main elements in coordination of transit networks: the selection of appropriate transit centers, the design of the transit network, and determination of headways under coordinated and uncoordinated conditions. This research expanded on the work of Baaj and Mahmassani (1992).

Ceder et al. (2001) attempted to create bus timetables with maximum synchronization by maximizing the number of simultaneous bus arrivals. The synchronization problem was examined using mixed integer programming as well as a heuristic approach. Even when the problem could be formulated using an operations research technique such as MIP, the time taking to run small transit networks proved to be excessive for practical use. A heuristic approach proved to be more efficient.

2.3 Other Relevant Approaches

Han (1987) assessed transfer penalties for bus riders in Taipei, Taiwan. A disaggregate binary choice model was used to describe the behaviors of bus riders. The study pointed out that the current practice might have underestimated transfer penalties. For instance, it was reported that the estimated monetary values for walk time and wait time were six and three times the value of the in-vehicle time, respectively. However, it is important to note that transfer penalties to bus riders in Taipei may be different to those in the U.S.

Kuah and Perl (1989) reported that the costs to transit agencies were mainly attributed to labor and fuel costs, service in low-density areas, and poor configuration of the transit network.
system due to inadequate planning tools, unrealistic agency goals, and inadequate fare structure to offset the agency costs. To remove these deficiencies, transit planners, operators, and researchers proposed the following measures: identifying adequate funding sources, developing a more realistic fare structure, improving services to attract ridership, providing a feasible peak-hour service, reducing or discontinuing service in low ridership areas, effective use of technology, privatization of transit services, and creating an integrated transit system, Kuah and Perl (1989).

Newell (1973) presented an array of solutions to common operations research problems. The author attempted to show that most of these problems could be solved by elementary calculus, for instance, determining the set of times at which buses can be optimally dispatched so that the total passenger waiting time is minimized or finding optimal locations of stations.

More innovative approaches such as artificial intelligence (AI) have been studied to solve similar problems. AI search algorithms are supposed to have some advantages over more conventional models. Baaj and Mahmassani (1992) used a fifth-generation computer language called Lisp to carry out the search. Their approach is heuristic and is used to find the optimal solution. The three main components in their approach include: 1) a route design algorithm to create different sets of routes based on the main objectives, 2) an analysis procedure that keeps load factors under the stipulated maximum, and 3) a route improvement algorithm to determine a new set of routes.
Bookbinder and Desilets (1992) suggested to find a more efficient methodology for solving the transfer time problem using newer algorithms such as simulated annealing, which is a technique based on the annealing process to obtain a faster optimum solution. The annealing process consists of using temperature to heat up solids into a liquid state and the slowly cooling it off to obtain a crystallized structure state in which the energy is minimal.

To solve the transit network problem, researchers have used mathematical programming, combinations of artificial intelligence and operations research approaches, fuzzy logic, artificial neural networks, and genetic algorithms. For instance, Kikuchi and Parameswaran (1993) used a fuzzy control method to coordinate schedules at airport hubs and transit terminals. They used fuzzy rules such as high, low, little, and more to perform time shifting in a more efficient manner. The authors claimed that this method produced better results than the enumeration method, which was a heuristic trial-and-error method. To test the methodology, two hypothetical examples were presented to compare the enumeration and the fuzzy control methods. Despite the limitations of this research to one transfer hub, the fuzzy rule-based approach present good opportunities in the solution of the transfer time optimization problem.

Koffman and Rousseau (1993) dealt with the issue of trip shifting and attempted to reduce operating costs. They believed that with minor tweaking of the departure times and headway in the timetable, particularly during peak periods, a reduction in the number of vehicles could be achieved. Although cost was a very important issue to consider, the
paper concentrated on a global optimization with the main goal of minimizing the waiting time for the majority of users.

A heuristic approach that maximized bus synchronization was described in (Ceder and Tal 1997 and Ceder et al. 2001). The optimal solution was based on maximizing the number of simultaneous arrivals at a transfer node. This heuristic approach was mainly concerned with time synchronization and did not deal with random arrivals or passenger demand.

Sutanto and Koshi (1999) developed a heuristic system to create, analyze, and optimize routes in a transit network. Their approach was based mostly on mathematical programming, which differed from others because it used elastic demand to allow for the interaction between modes due to different level of service (LOS).

Another relevant and innovative research was conducted by Pattanaik et al. (1998), who studied the design of an optimal bus route network using a genetic algorithm, with the objective function of reducing the total cost of users and operators. The model dealt with the development of the best route configuration by creating a set of routes and their frequencies. Using strings of fixed length to represent routes, an iterative process optimized route layout and number of routes. To be able to reduce the number of iterations, a variable string length was used to simultaneously consider both the route layout and the number of routes.
Deb and Chakroborty (1998) also presented a GA based approach as a theoretical framework on the formulation of transfer optimization. The main goal was to find optimal bus arrival and departure times. They used four relatively small case studies to test the genetic algorithm under different scenarios, which included different network characteristics, schedule adherence, and bus capacity.

Shrivastava and Dhingra (2002) studied the coordination between trains and buses and tested this type of intermodal schedule coordination at two train stations in Mumbai, India. The coordination of buses was obtained using a Schedule Optimization Model (SOM), with the objective function being minimizing the transfer time between trains and buses and minimizing the operating costs for the bus system. Like other previous research (Kikuchi and Parameswaran 1993, Deb and Chakroborty 1998, Chakroborty et al. 1995, and Ceder et al. 2001), the authors mentioned that using classical approaches were not suitable since the objective function and constraints made the problem nonlinear and non-convex with a large number of variables. Therefore, they recommended using genetic algorithms for the optimization of this particular type of transfer coordination.

Fleurent et al. (2005) presented a heuristic approach for the synchronization problem, taking into account concerns that schedulers face in practice. The model includes parameters such as minimum, maximum, and ideal waiting times. Weighting factors such as time of day, places, route, and trip directions are also included in the model. The authors used concepts like Trip Meets and Meet Builders as well as a Quality Index in the synchronization process. The algorithm was tested to improve the connection between
buses and rail. Different scenarios, to be used in the selection of the most appropriate solution, were created showing the feasible meets and their associated unproductive times.
CHAPTER 3
GENETIC ALGORITHMS

3.1 Overview

Genetic algorithms (GAs) are an optimization technique inspired by the theory of natural evolution. These algorithms fall into the category of evolutionary algorithms (Whitley 1994, Sipper 2000). They are modeled after biological principles, and have been a subject of extensive research over the last several decades, in fields including both science and engineering (Deb 2001). One of the main reasons for this interest is the ability of GAs to deal with non-linear problems and avoid being trapped into local optima and minimum requirement on the mathematical properties of the objective functions and constraints.

Originally, GAs were developed by John Holland of the University of Michigan, Ann Arbor in 1975 (Holland 1975). Since then, there have been significant developments in the theory and applications of GAs. Much of the work has been referred to as “evolutionary” methods (Sipper 2000). According to Haupt and Haupt (1998), some of the advantages of GAs are:

- They deal with both continuous and discrete parameters;
- They do not require derivatives;
- Search is carried out in various places of the search space simultaneously;
- Large number of parameters may be handled;
- They are well suited for parallel computing;
• They do not get “stuck” in local optima;
• They can provide a list of possible solutions; and
• They work well with numerical, experimental, and analytical data.

Genetic algorithms use a chromosome-like data structure to encode possible solutions to specific problems (Whitley 1994). The most common genetic algorithms are binary GAs and continuous GAs. Both algorithms follow the same principles of a GA, which are borrowed from natural genetics (Haupt and Haupt 1998, Deb 2001). In binary GAs, parameters are encoded as binary strings, whereas the continuous GAs works with continuous parameters. They differ from classic optimization procedures because they can 1) work with coding of the set parameters, rather than the parameters themselves, 2) search from a population of points, not particular points, 3) use objective functions, rather than derivatives, and 4) use probability functions, rather than deterministic functions, Golberg (1989).

Overall, a genetic algorithm is an iterative procedure that searches for a global optimal solution. Figure 3.1 shows a flowchart of a basic genetic algorithm process. The algorithm starts with data from parameters, functions, costs, etc. To begin, the population size is pre-determined and an initial population is selected from the search space, which is made up by individuals, or potential solutions, encoded as strings, known as the “genome” or “chromosome”. The search space is the space of all the feasible solutions to a particular problem. Chromosomes are made up of genes, and a gene is the basic element of a chromosome.
The Genetic Algorithm

Figure 3.1 Flowchart of a Simple GA
A complete loop of the algorithm is called a generation during which a new generation of population is evolved from the previous one. The new generation is expected to have improved overall fitness, determined by their objective function values. Therefore, it may take many generations before an optimal (global or local) solution is reached.

As part of the iteration process, the “fitness” of each individual or genome in the current population is evaluated according to an objective function. The fitness values determine which chromosomes are to be selected for reproduction when generating a new population. There are many methods to select genomes. The most common methods are roulette wheel selection and the rank selection. There are also others methods such as the tournament selection, which is a variation of the ranking method.

As previously mentioned, chromosomes are selected according to their fitness values. This ensures that only the most fit will survive and reproduce to generate the next generation of population. The probability of a chromosome or “individual” with a better fitness value to be selected for reproduction is greater than individuals with a low fitness value; the genes of the latter thus tend to disappear (Sipper 2000).

The reproduction process typically involves the application of two operators: crossover and mutation. In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next, and mutation is another operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. Crossover involves combining genes from
different parents to form new chromosomes. The selected chromosomes are the “parents” and the resulting chromosomes are called “offsprings”. Crossover enables chromosomes to improve subsequent generations of population thus moving toward the optimal solutions. The probability that chromosomes are selected is called crossover rate. Mutation involves randomly changing genes within a chromosome and is used as a means to prevent the GA from getting stuck in local optima. Mutation improves the likelihood that chromosomes jump to a different area within the search space. The probability of mutation is called the mutation rate.

The process of selection, crossover, and mutation is iterative and will continue until the results converge or a predetermined number of generations have been reached.

A pseudo code of a simple GA is given below (Sipper 2000).

begin GA
    g:=0 { generation counter }
    Initialize population P(g)
    Evaluate population P(g) { i.e., compute fitness values }
    while not done do
        g:=g+1
        Select P(g) from P(g-1)
        Crossover P(g)
        Mutate P(g)
Evaluate $P(g)$

end while

end GA

3.2 Considerations in GAs

Genetic algorithms may belong to either the category of the $(\mu + \lambda)$ evolutionary strategy (ES) or the $(\mu, \lambda)$ ES (Whitley 1994). The $(\mu + \lambda)$ ES selects a new population based on the mostly fit chromosomes from a set of $\mu$ parents and $\lambda$ offsprings. With the $(\mu, \lambda)$ ES, the $\lambda$ offsprings replace the $\mu$ parents in every generation.

Other flavors of GAs include the CHC algorithm. CHC stands for Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation. The CHC algorithm behaves like a $(\mu, \lambda)$ ES model, uses small groups, and selects the most fit members of the population based on a random selection. In this model, aggressive search is performed, allowing only the best chromosomes to survive. To balance this aggressiveness, it uses uniform crossover, in which half of the genes are swapped between chromosomes. Heavy mutation is applied when parents and offsprings begin to look alike, but the best chromosomes are kept intact.

Some researchers believe that in order to produce better results from genetic algorithms, hybrid models are necessary (Whitley 1994). For instance, GAs are known to be good performers at the global level, but are poor performers at the local level. As a result, it may take longer for a genetic algorithm to find an optimal solution than other algorithms.
Therefore, it is suggested that the performance of GAs at the local level may be improved with the use of hill-climbing algorithms. Others suggest that only a combination of mutation and hill-climbing is necessary to solve many problems. This issue has been controversial (Whitley 1994).

There has also been much debate on mutation. Some believe that mutation alone is enough to produce good results. Others argue that mutation is only a means to prevent a GA from converging too quickly and getting trapped in a local optimum and that mutation rate should be low. Sengoku and Yoshihara (1998) suggested an optional mutation rate of 20% for their 2-Opt Mutation operators, while Obitko (1998) recommended a very low mutation rate of 0.5% to 1%.

It is worthwhile to mention that parallel computing is an important element of genetic algorithms, since evaluation of individuals in a population may be accomplished independently thus in parallel. For instance, the population may be divided into smaller subsets to be handled by multiple processors or networked computers simultaneously.

Using a migration operator may enable the algorithm to gain some knowledge from other populations to help preventing the GA from converging prematurely. However, migration needs to be controlled, as a large number of chromosomes might change the "direction" of the entire population due to migration. This may cause chaos, which might put at risk the valuable learning previously obtained.
Genetic Algorithms are particularly useful with problems with large numbers of variables and large search spaces that are non-linear and discrete in nature. Nevertheless, genetic algorithms do not guarantee optimal solutions. Although they usually do not get trapped into local optima, they are likely to provide a solution that is close to the global optimum. Therefore, genetic algorithms use, as the stopping criteria, a maximum number of generations until a satisfactory solution is reached. Lastly, GAs are usually designed and implemented differently. Therefore, results from similar optimization models may vary due to the use of different random number generator, parameters, operators, etc. (Mardle and Pascoe 1999).
CHAPTER 4

METHODOLOGY

This chapter first presents a mathematical definition of the problem, and then describes an approach based on the GAs developed in this dissertation research.

4.1 Introduction

The problem of minimizing transfer time may be formulated as a mixed-integer nonlinear programming problem. The main objective is to obtain optimal arrival and departure times to minimize total transfer times (Chakroborty et al. 1995, Deb and Chakroborty 1998). While concisely defined, such a formulation does not provide a practical method for solving the problem. According to the findings from the literature review, analytical solutions for such a problem are infeasible when dealing with large data sets.

The proposed approach is to minimize the total transfer time for all transfer connections in an existing fixed-route transit system using a genetic algorithm (GA). Considerations are given to the most critical elements involved in the process, such as existing time tables, walking time, transfer demand, random arrivals, connectivity, etc. The proposed methodology is designed to assist transit agencies in their efforts to provide a more efficient and coordinated service. The research concentrates on the coordination of schedules by finding the optimal combination of departures and arrival times for all routes of a fixed-route transit system. This is accomplished by shifting the departure times in the timetables. It considers all trip combinations for all timetables without
modifying existing headways. This method provides enough flexibility to allow scheduled times in individual routes to remain constant or vary, as desired.

This approach differs from other research in that it is conceived from an operation perspective and practical applications in mind. Most of the studies in this area have either presented a theoretical formulation alone, tested fictitious or small case studies, or present solutions difficult to implement in practice. For instance, Deb and Chakroborty (1998) presented an approach to optimize bus arrival and departures and used three transfer stations to test their methodology. A binary string coding was used within the genetic algorithm and a continuous density function was assumed to accommodate stochastic arrivals. Bookbinder and Desilets (1992) presented a theoretical framework in which the only two variables were headways, which were assumed constant, and offsets. The term offset times refers to the departure times on the first bus of a particular line.

This dissertation attempts to fill the gap by developing an optimization methodology that may be incorporated into the scheduling process at a transit agency. The intent is to use this methodology as part of the timetable development in which transfers are optimized without having a major impact on the amount of service provided by the agency. While other GA approaches involving headway modifications (Chakroborty et al. 1995, Deb and Chakroborty 1998, Shrivastava and Dhingra 2002, and Deb 2001) have practical applications, they may not be suitable without careful consideration of issues related to design, operation, and local variables.
Below is a list of issues considered and the differences between this research and previous approaches:

1. This dissertation addresses and tests the methodology on a large network considering an entire transit system.

2. The proposed methodology makes use of the Genetic Algorithm concepts to solve this scheduling problem. It makes direct use of timetable data from scheduling software to compute all possible trip combinations.

3. The algorithm uses chromosomes with discrete values to represent time shifts, rather than binary coding to represent a series of variables.

4. The dissertation also incorporates ridership demand to account for passengers transferring between buses. The randomness of bus arrivals is addressed. Considerations are given to using average bus arrivals as well as simulated random bus arrivals from data collected by an Automatic Vehicle Location (AVL) Systems or Automatic Passenger Counters (APCs).

5. A walking time factor from arriving to departing buses is included.

6. Finally, a computer program with an user-friendly GUI was developed to test the methodology.

Time shifts can be applied to each timetable to generate a new set of optimized timetables and easily incorporated into the regular scheduling process. This approach will have a minimum impact on the existing operation, as there are no headway changes or a need for more buses or additional/less services. This is important because transit agencies cannot
easily modify their headways without getting involved into a political process or impacting its operation.

4.2 Mathematical Formulation of Transfer Time Minimization

The mathematical formulation that deals with transfer time optimization, as presented by Chakroborty et al. (1995), is given as follows:

Min. \[ \sum_{i} \sum_{j} \sum_{k} \sum_{l} \sum_{m} (d_{ik}^m - a_{ij}^l) \delta_{ijk}^m \omega_{ijk} \]  

Subject to:

\[ d_{ij}^l - a_{ij}^l \leq s_{ij}^{\text{max}} \quad \forall i,j,l \]
\[ d_{ij}^l - a_{ij}^l \geq s_{ij}^{\text{min}} \quad \forall i,j,l \]
\[ a_{ij}^l - a_{ij}^{l-1} \leq h_{ij} \quad \forall i,j,l \]
\[ (d_{ik}^m - a_{ij}^l) \delta_{ijk}^m \leq T \quad \forall i,j,k,l,m \]
\[ d_{ik}^m - a_{ij}^l + M(1 - \delta_{ijk}^m) \geq 0 \quad \forall i,j,k,l,m \]
\[ \sum_{m} \delta_{ijk}^m = 1 \quad \forall i,j,k,l \]

where

\[ a_{ij}^l = \text{arrival time of bus } l \text{ on route } j \text{ at station } i \]
\[ d_{ik}^m = \text{departure time of bus } m \text{ on route } k \text{ at station } i \]
\[ \delta_{ijk}^m = \text{binary variable (0, 1). It assumes a value of 0 if there is no transfer from bus } \]
\[ \text{ of route } j \text{ to bus } m \text{ of route } k. \text{ A value of 1 means transfer may occur.} \]
\[ h_{ij} = \text{headway of route } j \text{ at station } i \]
\[ T = \text{maximum transfer time} \]
\[ s_{ij}^{\text{max}} = \text{maximum stopping time for bus } j \text{ at station } i \]
\[ s_{ij}^{\text{min}} = \text{minimum stopping time for bus } j \text{ at station } i \]
\[ M = \text{a large number to ensure that the constraint is } \geq 0 \]
\[ \omega_{ijk}^l = \text{transfer volume from bus } l \text{ of route } j \text{ to route } k \text{ at station } i \]

Considering ridership in the model is very important to ensure that schedules are optimized in such a way to benefit the majority of transit riders. Therefore, locations with higher ridership will carry a heavier weight than locations with a lower ridership.

This formulation is based on a classical nonlinear mixed integer programming problem. Setting up the model is complex, cumbersome, computationally inefficient, and in some cases it may fail to converge or to provide the optimal solution. Even for relatively small transit systems, setting up a model including all the variables and constraints with all the possibilities is just unwieldy, as mentioned by Kikuchi and Parameswaran (1993), Deb and Chakroborty (1998), Chakroborty et al. (1995), and Ceder et al. (2001). Therefore, improved techniques are necessary to efficiently solve the transfer time minimization problem.

The following is a description of the formulation of the minimization problem developed in this research. Assume there are \( N \) bus routes. Now consider the time a passenger spends transferring from bus \( p \) on route \( i \) to bus \( q \) on route \( j \). The bus stop for bus \( p \) is \( k \) and that for bus \( q \) is \( l \). Denote the arrival time of bus \( p \) at stop \( k \) as \( a_{ikp} \). \( \Delta_{ikp} \) is bus \( p \)'s
deviation from its scheduled arrival time at bus stop $k$ due to the randomness in bus arrivals and $v \sigma_{kp}$ is a measure of variability of the schedule deviations. The variability measure is included to incorporate the variance in the distribution of bus arrivals. $S_i$ is the time shift in the timetable for route $i$. Time shifts are the times that will be added to the existing timetables in order to create optimized timetables. Therefore, $(a_{ikp} + \Delta_{ikp} + v \sigma_{kp} + S_i)$ is the time when the passenger from bus $p$ transferring to bus $q$ arrives at stop $k$.

Now consider the departure time of bus $q$. Let $d_{jql}$ be the scheduled departure time of bus $q$ at stop $l$. $\Delta_{jql}$, $v \sigma_{jql}$, and $S_j$ have similar meanings as $\Delta_{ikp}$, $v \sigma_{kp}$, and $S_i$, respectively. Therefore, $(d_{jql} + \Delta_{jql} + v \sigma_{jql} + S_j)$ represents the new departure time of bus $q$ considering the schedule deviation and the change made to the timetable for route $j$. The time spent by the passenger waiting for bus $q$ to depart is, therefore, $(d_{jql} + \Delta_{jql} + v \sigma_{jql} + S_j) - (a_{ikp} + \Delta_{ikp} + v \sigma_{kp} + S_i + w_{ijkl})$. $w_{ijkl}$ is the time the transfer passenger spent to walk from stop $k$ to stop $l$. Summation over all passengers transferring between all possible buses on all routes will give the total transfer time in the system. The problem may thus be formulated as:

Minimize

$$\sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{D_i} \sum_{l=1}^{D_j} R_k \sum_{p=1}^{B_i} \sum_{q=1}^{B_j} \{[d_{jql} + (\Delta_{jql} + v \sigma_{jql}) + S_j] - [a_{ikp} + (\Delta_{ikp} + v \sigma_{kp}) + S_i + w_{ijkl}]\}$$  \hspace{1cm} (2)

Subject to:
\[-b_i \leq S_i \leq b_i \quad b_i \leq \frac{\sqrt{2}}{2} h_{min}^{(i)} \]
\[-b_j \leq S_j \leq b_j \quad b_j \leq \frac{\sqrt{2}}{2} h_{min}^{(j)} \]
\[d_{jkl} \geq (a_{ikp} + w_{ijkl}) \]

\[
C_{ijkl} = \begin{cases} 
0, & \text{if there is no connectivity between routes } i \text{ and } j \text{ at bus stops } k \text{ and } l \\
1, & \text{if there is connectivity between routes } i \text{ and } j \text{ at bus stops } k \text{ and } l
\end{cases}
\]

where

- $a_{ikp}$ = arrival time of bus $p$ for Route $i$ at transfer point(s) $k \forall r_n$
- $d_{jkl}$ = departure time of bus $q$ for Route $j$ at transfer point(s) $l \forall r_n$
- $\Delta_{ikp}, \Delta_{jkl}$ = deviation from schedule due to random bus arrivals
- $\nu$ = schedule variability coefficient
- $\sigma_{jkl}$ = standard deviation in departure time distribution for bus $q$ on route $j$ at stop $l$
- $\sigma_{ikp}$ = standard deviation in arrival time distribution for bus $p$ on route $i$ at stop $k$
- $r_n$ = set of timetables for all $N$ routes
- $R_k$ = ridership demand at transfer point $k$
- $S_i, S_j$ = time shifting for Routes $i$ and $j$, respectively
- $b_i, b_j$ = minimum and maximum time shifts
- $h_{min}^{(i)}$ = minimum headway on Route $i$
- $h_{min}^{(j)}$ = minimum headway on Route $j$
- $w_{ijkl}$ = walk time between bus stops at transfer points $k$ and $l$ from Route $i$ to Route $j$
\[ C_{ykl} = \text{connectivity matrix from Route } i \text{ to Route } j \text{ and from transfer points } k \text{ to } l \]

\[ N = \text{number of bus routes in the system} \]

\[ D_i = \text{number of transfer points on route } i \ (i = 1, 2, \ldots, N) \]

\[ D_j = \text{number of transfer points on route } j \ (j = 1, 2, \ldots, N) \]

\[ B_i = \text{number of buses in service on route } i \ (i = 1, 2, \ldots, N) \]

\[ B_j = \text{number of buses in service on route } j \ (j = 1, 2, \ldots, N) \]

In the formulation, \( S_i \) and \( S_j \) are the decision variables, which assume time shifts that vary from \(-\frac{h}{2}\) to \(\frac{h}{2}\) at a minimum of one minute increments. A solution is represented by a vector \( S \), which stores all the \( S_i \), that is \( S = \{S_i\} \).

### 4.3 GA Implementation

The model described in the previous subsection is implemented using a GA-based algorithm. The GA in this research is based on the evolutionary strategy \((\mu + \lambda)\), where the population is selected from the mostly fit chromosomes of a set of \( \mu \) parents and \( \lambda \) offsprings. The data flow is depicted in Figure 4.1. It starts with the collection of the timetables from scheduling software, from which scheduled arrival and departure times are obtained. Real-time arrivals and departures may be obtained from AVL or APC systems. Transfer ridership data may be obtained from an electronic fare collection system with swipe cards or smart card capabilities.
Figure 4.1 Optimization Model
Next, the data go through a formatting and quality assurance process to be prepared for the optimization algorithm. This step includes the creation of a connectivity matrix, as well as a ridership transfer matrix. Figures 4.2 and 4.3 depict, respectively, a snapshot of both the ridership and transfer tables that were used in the case study of this research.

![Ridership Table](image)

**Figure 4.2  Ridership Table**

The rows and columns of each table correspond to the route numbers, and the cell values in Figure 4.2 are the average transfer ridership from one route to another route per bus trip. As an example, the value 0.46 in row 1 and column 5 means that on average 0.41 rider transfers per bus trip between Route 1 and Route 5.

Similarly, the cell values in Figure 4.3 represent the transfer connectivity between routes and the locations where transfers occur. For instance, the cell with value of 0201 in row 1 and column 5 indicates that transfers occur from time point 2 of route 1 to time point 1 of route 5.
In the GA, a chromosome that represents a solution contains $N$ integers, where $N$ is the number of routes and the $i$th integer represents the time shift to be applied to the $i$th route.

The reproduction operator ranks solutions based on their fitness and the most fitting chromosomes are retained after each generation (Elitism) to produce the next generation of solutions. The crossover operator used in the algorithm is the standard uniform crossover as in classical GAs. The operator swaps genes from two chromosomes based on the crossover rate. The mutation operator replaces the existing gene value with another value from a list of all possible time shifts ($-h_{\text{min}}/2$, ..., 0, ..., $+h_{\text{min}}/2$), where $h_{\text{min}}/2$ is one half of the minimum headway for the whole system, based on the mutation rate.

Figure 4.4 depicts an example of the crossover and mutation operators. Uniform crossover is used in the GA computations. That is, for each new generation, genes from
all parent chromosomes are swapped, based on random gene selection and a predetermined crossover rate. In the case of the mutation operator, the genes are randomly selected based on the mutation rate and are mutated within a predetermined range of values.

<table>
<thead>
<tr>
<th>Parent 1</th>
<th>Parent 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-12</td>
<td>-4</td>
</tr>
<tr>
<td>11</td>
<td>-13</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>-6</td>
</tr>
<tr>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-12</td>
<td>-6</td>
</tr>
<tr>
<td>-12</td>
<td>-5</td>
</tr>
<tr>
<td>-12</td>
<td>-12</td>
</tr>
</tbody>
</table>

**Generation n**

<table>
<thead>
<tr>
<th>Child 1</th>
<th>Child 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-4</td>
<td>-12</td>
</tr>
<tr>
<td>-13</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>-6</td>
</tr>
<tr>
<td>-9</td>
<td>-9</td>
</tr>
<tr>
<td>-12</td>
<td>-6</td>
</tr>
<tr>
<td>-12</td>
<td>-12</td>
</tr>
</tbody>
</table>

**Generation n + 1**

(a) Cross-Over Operator

(b) Mutation Operator

Figure 4.4  Crossover and Mutation Operators

The algorithm terminates when a pre-determined condition is met, which may be either when a specified computing time has been exceeded or when no significant improvements are made within a number of generations. The final time shifts are saved and may be used to generate new timetables.

4.4. **Incorporation of Random Bus Arrivals and Departures**

Buses do not usually arrive exactly according to their scheduled times; they are often early or late due to unpredictable traffic conditions. Even though bus arrival and
departure times may be considered a random function, this function is highly dependant on the scheduled times. Additionally, arrival and departure times may vary considerably from one day to the next.

When dealing with random bus arrivals, one solution might be to directly use the archived data from an AVL or APC system to obtain the actual bus arrival and departure time, which replace terms of \((d_{jlq} + \Delta_{jlq} + v\sigma_{jlq})\) and \((a_{kp} + \Delta_{kp} + v\sigma_{kp})\) in expression (2). The caveat of this approach is that the transit agency needs to have the ability to extract actual departure and arrival time information from the AVL or APC data and match it with the ridership information. However, there is a trend among transit agencies in the U.S. towards using data from AVL, APC, and similar technologies to improve efficiencies and performance. This includes collecting and using AVL or APC data to help improve transit schedules.

Another approach is to use average schedule deviation values obtained based on data collected from an AVL or APC system at each transfer location. Such averages do not reflect the distribution or variability of the schedule deviations. Moreover, median values are preferred to averages, particularly when data are likely to contain outliers.

In the proposed approach, a theoretical framework is presented to not only include the average or the median, but also the variability of schedule deviations. This is incorporated into the model by using a schedule variability coefficient and a standard deviation, as presented in Formula 2. With these parameters, a safety value (e.g., 10%)
of the standard deviation may be added to the average value of the schedule deviation to ensure that transfers are not missed due to the variability of arrivals and departures. Nevertheless, an important consideration with this approach is the cost. As transit agencies add safety factors to the running times to improve on-time performance, operating costs also increase because of the additional vehicle running time and labor hours. Therefore, these factors have to be carefully considered, as transit agencies may be highly sensitive to increase in operational costs.

A third approach is to use a probability distribution to describe the deviations of scheduled times from real-time bus arrivals or departures and then simulate the effects of schedule deviations using the Monte-Carlo method.

Real-time arrivals or departures may be obtained from manual observations, data collected by Automatic Vehicle Location Systems, or data from Automatic Passenger Counters. To illustrate this further, Figure 4.5 depicts a histogram, which gives the frequencies of deviations of bus arrivals at the Broward Terminal in Fort Lauderdale, Florida, superimposed by four probability distributions of Lognormal, Normal, Gamma, and Weibull.

Table 4.1 gives the probability density function of these distributions. These theoretical distributions are tested to determine which one best fits the observed distribution of schedule deviations.
The following goodness-of-fit tests are used: Kolmogorov-Smirnov, Cramer-von-Mises, and Anderson-Darling based on the empirical distribution function (EDF). The Goodness-of-Fit tests are conducted to examine the null hypothesis that the observations are a random sample of the selected probability distribution. According to (SAS 2003), the Kolmogorov-Smirnov statistic (D) is defined as:

\[ D = \sup_x | F_n(x) - F(x) | \]

This statistic is computed as the maximum of D+ and D-, where D+ is the largest vertical distance between the EDF and the distribution function when the EDF is greater than the
distribution function, and $D^{-}$ is the largest vertical distance when the EDF is less than the distribution function.

$$D^+ = \max_i \left( \frac{i}{n} - U_{(i)} \right)$$

$$D^- = \max_i \left( U_{(i)} - \frac{i-1}{n} \right)$$

$$D = \max (D^+, D^-)$$

### Table 4.1 Probability Distributions (SAS 2003)

<table>
<thead>
<tr>
<th>Distribution</th>
<th>PDF</th>
</tr>
</thead>
</table>
| **Lognormal** | \[ p(x) = \begin{cases} 
\frac{100h\%}{\sigma \sqrt{2\pi}} \exp \left( - \frac{(\log(x - \theta) - \zeta)^2}{2\sigma^2} \right) & \text{for } x > \theta \\
0 & \text{for } x \leq \theta 
\end{cases} \] |
| Where | \( \theta = \text{threshold parameter} \)  \( \zeta = \text{scale parameter} \)  \( \sigma = \text{shape parameter} \)  \( \sigma > 0 \) |

| **Normal** | \[ p(x) = \frac{100h\%}{\sigma \sqrt{2\pi}} \exp \left( - \frac{1}{2} \left( \frac{x - \mu}{\sigma} \right)^2 \right) \] for \(-\infty < \zeta < \infty \) |
| Where | \( \mu = \text{mean} \)  \( \sigma = \text{standard deviation} \)  \( \sigma > 0 \) |

| **Gamma** | \[ p(x) = \begin{cases} 
\frac{100h\%}{\Gamma(\alpha)\sigma} \left( \frac{x - \theta}{\sigma} \right)^{\alpha-1} \exp \left( - \left( \frac{x - \theta}{\sigma} \right) \right) & \text{for } x > \theta \\
0 & \text{for } x \leq \theta 
\end{cases} \] |
| Where | \( \theta = \text{threshold parameter} \)  \( \sigma = \text{scale parameter} \)  \( \sigma > 0 \)  \( \alpha = \text{shape parameter} \)  \( \alpha > 0 \) |
The Anderson-Darling statistic and the Cramér-von Mises statistic belong to the quadratic class of EDF statistics, and are based on the squared difference \((F_n(x) - F(x))^2\).

Quadratic statistics have the following general form:

\[
Q = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 \psi(x) dF(x)
\]

The function \(\psi(x)\) weights the squared difference \((F_n(x) - F(x))^2\).

The Anderson-Darling statistic \((A^2)\) is defined as

\[
A^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 [F(x)(1 - F(x))]^{-1} dF(x)
\]

where the weight function is \(\psi(x) = [F(x)(1 - F(x))]^{-1}\).

The Anderson-Darling statistic is computed as

\[
A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} [(2i-1) \log U_i + (2n+1-2i) \log (1-U_i)]
\]

The Cramér-von Mises statistic \((W^2)\) is defined as

\[
W^2 = n \int_{-\infty}^{+\infty} (F_n(x) - F(x))^2 dF(x)
\]

where the weight function is \(\psi(x) = 1\).

The Cramér-von Mises statistic is computed as

\[
W^2 = \sum_{i=1}^{n} \left( U_i - \frac{2i-1}{2n} \right)^2 + \frac{1}{12n}
\]

Probability plots can also be used in determining the best fit. Linearity in the probability plot is a good indicator of how well a distribution fits the data. The hypotheses of distribution are tested using actual data from the AVL system of BCT. The data, as well
as the test results, are described and presented in Chapter 5, which provides the results from numerical experiments designed to test the proposed methodology.

Once a distribution is determined for schedule deviation, a Monte Carlo simulation process may be used to create a series of randomly-generated scheduled deviation values, which can be added to the scheduled bus departure and arrival times to simulate realistic departures and arrival times. Monte Carlo methods are commonly used to simulate the behavior of physical or mathematical systems by using random numbers and probability statistics. They allow a large system to be sampled randomly and the sampled data may be used to describe the system. They are useful for modeling phenomena with a significant degree of uncertainty.

4.5 Data Requirements

To be able to test and apply the proposed methodology, the following data are needed:

- Timetables – to provide information on routes, time points, and arrival and departure times. Scheduled arrival and departure times may be obtained from the timetables, but need to be converted to a specific format such as converting the time based on a 24-hour clock, sequentially numbering the time points, removing information that is not needed, etc.

- Route maps – to provide information on individual routes and their time points. Route maps and time points can make use of GIS for display and analysis purposes.
• Transit system map – to visualize routes and transfer locations. Route maps and
time points, as well as route alignment and potential transfer locations, may be
displayed in a GIS for visualization and analysis purposes.

• Ridership data – to provide information on ridership at transfer points. The
ridership data need to come from an automated system, such as an electronic fare
collection system, that allows the determination of the path or direction of
transfers. Unfortunately, the data outputs from these systems are not yet designed
to support this type of applications. Therefore, the ridership data require
extensive formatting and manipulation before they may be used by the algorithm.
The data may be formatted in a matrix format to represent a measure of ridership
demand at time points. A connectivity matrix can be developed using discrete
values to indicate if a transfer occurs and among what routes and time points, as
shown in Figure 4.3.

• AVL and APC data – to obtain schedule deviations and information on boarding
activity at transfer points.

Lastly, the walking time, which is the time it takes to walk from an arrival bus stop to the
departure stop, is also needed. This is an important component, because it may take a
few minutes to transfer from one bus to another. Without this element, transferring
passengers may not have enough time to catch the departing bus. Initial field
observations suggest that on average it takes about two to three minutes. The value of
three (3) minutes will be used as the default in the transfer calculations.
4.6 Flowchart for Transfer Time Optimization

Figure 4.6 presents a flowchart describing the steps in the transfer time optimization. The process starts with the entry and initialization of parameters, variables, constraints, as well as the creation of the necessary arrays. Data from the timetables, transfer connectivity, and ridership matrices are then read and stored in the internal arrays.

Once the data are initialized, the GA process starts. This involves creating chromosomes (or solution vectors), generating an initial population, and loading the rates for the GA mutation and crossover operators. During each iteration or generation of the GA, time shifts will be added to the existing arrival and departure times. Transfer times are calculated and compared with the results from the previous generation. The process will stop when a predefined criterion is met. Final results may be used to optimize the timetables.

4.7 Computer Application

To test the transfer time optimization methodology, a computer program has been developed based on the platform of dBASE Plus. The dBASE database (dbf files) is also used to store the data needed to perform the calculations in the optimization program. Figure 4.7 shows the Graphical User Interface (GUI) of the Transfer Optimization program. With this interface, the user can enter the necessary information to perform the calculations for transfer time optimization.
Figure 4.6  Flowchart for the Optimization of Transfer Times
The user interface allows for entering parameters such as the number of directional routes, desired number of generations in the genetic algorithm, the number of chromosomes, as well as the crossover and mutation rates. Selections for random arrivals/departures, average schedule deviation values, or scheduled arrival/departure values are included forming the GUI. Maximum shifting values, walking times between transfer stops, and shifting intervals may be also selected on the form. Lastly, the GUI
allows for the selection of seed values, which are used in the generation of random numbers for Monte-Carlo simulation, as discussed in Section 4.7.1.

The transfer optimization program consists of a main form and four different programs that are used for comparison purposes. They perform the following functions: 1) calculation of existing transfer times; 2) calculation of the transfer times using a random brute force approach; 3) calculation of transfer times by entering a possible solution manually, and 4) calculation the transfer times using the genetic algorithm.

In computer science, a brute-force search consists of systematically enumerating every possible solution of a problem until a solution is found, or all possible solutions have been exhausted. Each button in the Options section of the user interface is associated with its corresponding program that performs a specific calculation. The computer code is flexible enough to allow modifications or improvements to be easily made by simply modifying some lines of code. The software code for this computer application is given in Appendix C.

Outputs may be either generated directly by pressing an appropriate button in the Output Section of the GUI or obtained from the dBase tables, which store the intermediate calculations, as well as the final results. The computer application also provides some flexibility to performed sensitivity analysis (see Numerical Experiments in Chapter 5), allows the user to experiment with different time-shift values, and is able to determine the exact route and location where the transfer savings occur.
4.7.1 Random Numbers

A random number generator is used in the program to generate the initial time shifts for all directional routes. The random number generator is also used by the crossover and mutation operators in the GA. There are the three options available in the GUI to create random numbers:

*Default* – It uses the dBASE default seed value of 179757 to generate the random numbers. The random function rand( ) generates the same sequence of values. This is useful to replicate scenarios. Using this random seed, every time the computer program is run, the same series of numbers are generated by the program. Therefore, these values can not be considered truly random numbers; they are pseudo random numbers. In order to generate a series of numbers that vary every time the computer program is run, a new random value needs to be used as the seed.

*Constant* – A constant value, selected on the form, may be added to the default seed value. Even though the behavior of the random numbers is similar to the seed generated using the default option, it will produce a different sequence of values. Therefore, initial solution vectors are different. This may be used to test the impact of using different seed values.

*Random* – In this case, seed values are generated by the following function: Int(1000000*Rand()). This function generates seed values between 0 and 999,999, which are used at each iteration. Therefore, different time shift values are created every time
the Random Calculation button or the GA button is pressed. This feature is useful for comparing transfer time calculations based on different initial solutions and a different set of random numbers at each generation.

4.7.2 Calculation of Transfer Times

Transfer times at transfer points are calculated by subtracting the bus arrival time from the bus departure time plus the waking time from one bus stop to another. This calculation is performed for all the trips in the timetables for all connecting transfer points. Schedule deviations, due to random bus arrivals, can also be included in the calculation of transfer times.

The time point locations and scheduled times in the timetables are dependent on the existing trip patterns and they may vary throughout a day. Because of this, there may be empty records either at the beginning, middle, or end of the timetables. This poses a challenge in the calculation of transfer times.

To make the calculations more efficient, some code has been added in the computer program to search only for non-null values. That is, transfer calculations will not be performed if the values for either departure or arrival times are not available or if the arrival time plus the walking time is greater than the departure time. The complete computer code that is used to calculate the transfer times is given in Appendix C. The algorithm can be set to stop when there is no improvement after a given number of generations or after running for a predetermined number of hours.
4.7.3 Computational Procedures

One of the main elements of the algorithm is the timetables and the transfer connection table that the algorithm must frequently access and modify. The algorithm was implemented as part of a computer application developed in dBASE. The application serves as the front-end (GUI) and it uses dbf tables as the back-end database.

Timetables were converted from text files to dbf tables. The transfer connectivity table was manually coded into a dbf table, based on information from printed timetables and route maps. Ridership demand was also manually entered into a dbf table. Data were read from the dBASE tables and stored in memory using arrays to reduce computational time.

Figure 4.8 illustrates how the cell values of the transfer connectivity table were created. This diagram depicts a transfer from Route 7 Eastbound to Route 18 Northbound. In this example, the cell value in the transfer connectivity table for these two routes would be 0603 (sixth and third column of the corresponding timetables). This meant that the transfer occurred from time point 6 (TP6) of route 7 to time point 3 (TP3) of route 18. The transfer time was seven minutes (7:10 – 7:03), which is the time it took to transfer from the 4th trip of the route 7 to the 8th trip of route 18. A similar process was followed, using all the directional timetables in the system, in order to populate the transfer connectivity table.
Figure 4.8 Transfer from Route 7 to Route 18 (Broward County Transit)
CHAPTER 5
NUMERICAL EXPERIMENTS

Numerical experiments have been conducted as part of this research to test the proposed methodology. The main objective is to demonstrate the feasibility and benefits of the proposed approach and to evaluate its effectiveness and efficiency. Lessons learned from this case study will provide insights into this particular type of scheduling problems. In this chapter, the proposed methodology is tested. Section 5.1 describes the data used in the numerical experiments. Section 5.2 discusses the simulation of schedule deviations.

5.1 Study Area and Data Collection

Data from Broward County Transit (BCT) are used for the case study. BCT is a medium-sized transit property with 40 routes and a total fleet of 275 buses, providing bus services to Broward County, Florida. Current ridership is over 120,000 boardings per day, and it continues to increase, making BCT one of the largest transit systems in Florida. Figure 5.1 depicts the BCT transit network, which includes the bus routes and time points overlaid on the street network.

Ridership data were collected from the GFI farebox system, a product of GFI Genfare, A Unit of the SPX Corporation, at Broward County Transit for the entire May, 2005. Using the GFI system, a SQL statement was created to export the data from the Sybase database to a text file and formatted as necessary to be used with the algorithm. The data only included swipe-card transactions. The swipe-card used in BCT is a fare medium with
encoded information on a magnetic stripe that allows passengers to quickly pay fares by swiping the card in the slot of a farebox card reader. The farebox records information on fare transactions, including date and time, along with operational data such as route, run, and vehicle number. For the purpose of this case study, ridership data were collected for the AM peak (from 7:00 AM to 9:00 AM) during weekdays. The transfer averages from this time period were used in the calculations of daily transfer ridership. To obtain the total daily transfer riders among the different routes, a crosstab table (see Table 5.1) of the average percentage of transfers by route was created. The total 120,000 daily riders were distributed among routes based on these percentages.

While the algorithm and computer program have been designed to handle networks at any level of detail, the following assumptions are used in this case study:

1. The level of detail is at the time point level, as defined in the timetables;
2. Schedule deviation at time points is assumed zero, when the effectiveness of the methodology is measured, so that results may be compared with the existing transfer times to measure the time savings on the same basis.
3. Walking time from one stop to another at a transfer point is assumed to be three minutes, which includes getting off the bus and walking to the transfer stop, and may include crossing the street.
4. Buses do not wait for transfer passengers (no-holding policy).
5. Buses have enough capacity to accommodate all transfer passengers.
Broward County Transit Network

Figure 5.1  BCT Network
The daily transfers per route were obtained by multiplying the route ridership with the 22% transfer rate. To estimate the average number of transfers per trip, the total number of daily riders for each route was divided by the total number of daily trips (riders per trip). Results from these calculations were provided in the ridership table, as depicted in Figure 4.2. Each cell value represents the average number of riders transferring between two routes.

Table 5.1 Cross Tabulations of Transfers among Bus Routes

<table>
<thead>
<tr>
<th>From_Rte</th>
<th>10B</th>
<th>100B</th>
<th>11B</th>
<th>110B</th>
<th>12B</th>
<th>120B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer</td>
<td>Layer Total N %</td>
<td>Layer Total N %</td>
<td>Layer Total N %</td>
<td>Layer Total N %</td>
<td>Layer Total N %</td>
<td>Layer Total N %</td>
</tr>
<tr>
<td>10B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>100B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>11B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>110B</td>
<td>0.10%</td>
<td>0.20%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>12B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>120B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>14B</td>
<td>0.02%</td>
<td>0.03%</td>
<td>0.01%</td>
<td>0.05%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>140B</td>
<td>0.01%</td>
<td>0.02%</td>
<td>0.00%</td>
<td>0.01%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>15B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>150B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>17B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>170B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>18B</td>
<td>0.00%</td>
<td>0.01%</td>
<td>0.01%</td>
<td>0.03%</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>180B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.02%</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
<tr>
<td>19B</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.01%</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>190B</td>
<td>0.00%</td>
<td>0.05%</td>
<td>0.01%</td>
<td>0.07%</td>
<td>0.01%</td>
<td>0.01%</td>
</tr>
</tbody>
</table>

5.2 Modeling Schedule Deviations

One of the issues concerning transfer time is the variance in bus arrival and departure times. Using the mean schedule deviations by route is one possible way to deal with this issue when optimizing transit schedule. Another possibility is to make the problem definition closer to reality by considering the probability distributions of schedule deviations and produce time shifts using the Monte Carlo Simulation to generate bus arrival data.
For this purpose, 12 locations were selected to represent typical transfer locations, based on different levels of ridership. Table 5.2 gives the locations where scheduled deviations were analyzed. At each location, schedule deviation observations were made for a two-week time period during the AM peak (7:00 AM – 9:00 AM). Data at these locations were obtained from a data set of a previous AVL study conducted by Zhao and Chung (2004).

Table 5.2 Sample Locations for Modeling Schedule Deviations

<table>
<thead>
<tr>
<th>Observations</th>
<th>Route</th>
<th>Direction</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>1</td>
<td>NB</td>
<td>Aventura Mall</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>SB</td>
<td>Aventura Mall</td>
</tr>
<tr>
<td>55</td>
<td>1</td>
<td>NB</td>
<td>Broward Terminal</td>
</tr>
<tr>
<td>61</td>
<td>1</td>
<td>SB</td>
<td>Broward Terminal</td>
</tr>
<tr>
<td>54</td>
<td>2</td>
<td>NB</td>
<td>Coral Square Mall</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>SB</td>
<td>Coral Square Mall</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>NB</td>
<td>Boca Raton</td>
</tr>
<tr>
<td>33</td>
<td>10</td>
<td>SB</td>
<td>Boca Raton</td>
</tr>
<tr>
<td>46</td>
<td>22</td>
<td>EB</td>
<td>Broward Terminal</td>
</tr>
<tr>
<td>39</td>
<td>22</td>
<td>WB</td>
<td>Broward Terminal</td>
</tr>
<tr>
<td>38</td>
<td>22</td>
<td>EB</td>
<td>Broward Blvd. and 441</td>
</tr>
<tr>
<td>38</td>
<td>22</td>
<td>WB</td>
<td>Broward Blvd. and 441</td>
</tr>
</tbody>
</table>

To perform the simulation, a distribution of schedule deviations is needed, from which a probability density function may be used to produce simulated schedule deviations during the Monte-Carlo Simulation. The next two subsections describe, respectively, the determination of the probability distribution of schedule deviation and the simulation results.

5.2.1 Probability Distribution of Schedule Deviations

To determine the distribution, the schedule and schedule deviation data were analyzed using SAS (Statistics Analysis Software) Version 9.1 for Windows. Each location is
analyzed using four probability distributions: Lognormal, Gamma, Normal, and Weibull distributions, depicted in Figure 5.2. Table 4.1 gives the probability density function of these distributions. The SAS code used to perform the calculations and to generate the graph is provided in Appendix A.

![Probability Distributions](image)

**Figure 5.2  Probability Distributions**

Based on visual inspection as well as statistical tests, the distribution that best fits the data for the 12 cases is the lognormal distribution, as demonstrated from various Goodness-of-fit tests, as well as probability plots. Figure 5.3 shows the output from SAS. Figure 5.4 depicts a probability plot for the random arrivals of the north bound buses of Route 1 at
the Broward Terminal. The data points fall close to a straight line, implying that the Lognormal function fits the data well.

<table>
<thead>
<tr>
<th>Bus Schedule Deviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCT Route 1 NB at Broward Terminal</td>
</tr>
<tr>
<td>Weekdays (7:00 AM - 9:00 AM)</td>
</tr>
</tbody>
</table>

The UNIVARIATE Procedure
Fitted Distributions for Deviation

<table>
<thead>
<tr>
<th>Parameters for Lognormal Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Threshold</td>
</tr>
<tr>
<td>Scale</td>
</tr>
<tr>
<td>Shape</td>
</tr>
<tr>
<td>Mean</td>
</tr>
<tr>
<td>Std Dev</td>
</tr>
</tbody>
</table>

Goodness-of-Fit Tests for Lognormal Distribution

<table>
<thead>
<tr>
<th>Test</th>
<th>Statistic</th>
<th>p Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kolmogorov-Smirnov</td>
<td>D</td>
<td>0.09004799</td>
</tr>
<tr>
<td>Cramer-von Mises</td>
<td>W-Sq</td>
<td>0.06272924</td>
</tr>
<tr>
<td>Anderson-Darling</td>
<td>A-Sq</td>
<td>0.43126093</td>
</tr>
</tbody>
</table>

Quantiles for Lognormal Distribution

<table>
<thead>
<tr>
<th>Percent</th>
<th>Quantile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Observed</td>
</tr>
<tr>
<td>1.0</td>
<td>-9.000000</td>
</tr>
<tr>
<td>5.0</td>
<td>-8.000000</td>
</tr>
</tbody>
</table>
Quantiles for Lognormal Distribution

<table>
<thead>
<tr>
<th>Percent</th>
<th>Observed</th>
<th>Estimated</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0</td>
<td>-7.00000</td>
<td>-6.92324</td>
</tr>
<tr>
<td>25.0</td>
<td>-5.00000</td>
<td>-5.17987</td>
</tr>
<tr>
<td>50.0</td>
<td>-3.00000</td>
<td>-2.43917</td>
</tr>
<tr>
<td>75.0</td>
<td>2.00000</td>
<td>1.50578</td>
</tr>
<tr>
<td>90.0</td>
<td>8.00000</td>
<td>6.51901</td>
</tr>
<tr>
<td>95.0</td>
<td>10.00000</td>
<td>10.40669</td>
</tr>
<tr>
<td>99.0</td>
<td>10.00000</td>
<td>20.11510</td>
</tr>
</tbody>
</table>

Figure 5.3 SAS Output

Lognormal Probability Plot for Random Bus Arrivals
Route 1 NB at Broward Terminal

Figure 5.4 Probability Plot
5.2.2 Simulation of Arrival and Departure Times

To estimate the deviation values from the Lognormal distribution, it is necessary to obtain the inverse of the probability function. The LOGINV function in Excel is used for this purpose. The format of the LOGINV function in Excel is \( \text{LOGINV}(\text{Probability}, \text{Mean}, \text{Standard\_dev}) \). Where \( \text{Probability} \) is a probability associated with the lognormal distribution, \( \text{Mean} \) is the mean of the distribution, and \( \text{Standard\_dev} \) is the standard deviation of the distribution.

For this function to work properly, there is a need to transform the \( x \) values (deviation) into a log form. That is, a new variable \( y = \ln(x) \) is created and the mean and standard deviations of the normal distributions are obtained. The Loginv function also uses the uniform random generator function rand(). These values are entered as \( \text{Loginv}(\text{Rand()}, \text{Mean}, \text{Standard\_Deviation}) \) to generate new values that follow the Lognormal distributions.

The Excel macro that implements the Monte Carlo Simulation to generate the random bus arrivals and departures is provided in Appendix B. The simulation consists of a series of iterations to generate output values that follow the Lognormal probability distribution. The number of output values was the same as the number of the AVL observations shown in Table 5.2. The simulated deviation values are compared to the field values, as depicted in Figure 5.5. Figure 5.6 depicts the histogram of simulated values, which follow the Lognormal distribution.
5.3 Random Brute Force Search

In an attempt to test the feasibility of the transfer time methodology and to measure the efficiency of the proposed GA, a random brute force search was conducted. Although this methodology is unlikely to produce an optimal solution, it is a better alternative to the Exhaustive Sequential Search that is inefficient and time consuming. Below are a series of steps performed during this test.

The first step is to convert existing scheduling data (timetables) into a series of dBASE tables (dbf), one table per route per direction. Each dbf table stores all the information from the timetables, including the trip sequences, scheduled departure/arrival times, and time point locations.
All the routes are renumbered for computational purposes. Table 5.3 shows the correspondence between the actual Inbound BCT Routes to the sequentially-numbered GA routes. A similar transformation is used for outbound routes.

The algorithm starts by initializing a series of program parameters, which are provided in Table 5.4. Next, the timetables are read, and data are stored in internal arrays. A vector of time shifts for all routes, similar to the one illustrated in Figure 5.7, is then randomly generated. During the testing, the solution vector contained 40 time shifts. These time shift values are added to the exiting timetables to create a new shifted timetable.
Table 5.3  Correspondence between GA and Actual Route Numbers

<table>
<thead>
<tr>
<th>Sequential Number</th>
<th>BCT Routes</th>
<th>Route DIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>IB</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>IB</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>IB</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>IB</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>IB</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>IB</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>IB</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>IB</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>IB</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>IB</td>
</tr>
<tr>
<td>11</td>
<td>14</td>
<td>IB</td>
</tr>
<tr>
<td>12</td>
<td>15</td>
<td>IB</td>
</tr>
<tr>
<td>13</td>
<td>17</td>
<td>IB</td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td>IB</td>
</tr>
<tr>
<td>15</td>
<td>20</td>
<td>IB</td>
</tr>
</tbody>
</table>

Transfer times are calculated based on the timetables on a trip-by-trip basis, considering all time points and all routes and for all times throughout the day. The algorithm loops to produce solution vectors until a specified stopping criterion is met. For this exercise, the program stops after 11 hours of computing time. The final solution vector is stored and the time shifts implementation plan is created.

Table 5.4  Algorithm Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Inbound/Outbound Routes</td>
<td>80</td>
</tr>
<tr>
<td>Range of Shiftings</td>
<td>+/- 15 min.</td>
</tr>
<tr>
<td>Time from bus to transferring stop</td>
<td>3 min.</td>
</tr>
<tr>
<td>Shifting Intervals</td>
<td>1 min.</td>
</tr>
<tr>
<td>Number of Transfer Stops</td>
<td>255</td>
</tr>
</tbody>
</table>
Transfer connections at time points are also stored in the dbf tables. Tables are organized in such a way that the column and row numbers represent the route numbers. A pseudo route number is used instead of actual route numbers so that all the routes are listed in a sequential order. Similarly, a conversion number is used to represent the time points of each route in sequential order, as required by the computer algorithm.

Transfer times are calculated as the difference between the time when the bus arrives at the stop and the departure of the connecting bus at the transfer stop. This includes the walking time to the transferring bus stop. Calculations are based on the timetables on a trip-by-trip basis and consider all time points and all routes. The program calculates the time it takes for passengers to transfer from one route to another for all times throughout the day. The algorithm loops to create solution vectors until a specified criterion is met. For this exercise, the program stops after 11 hours of computing time. The final solution vector is stored and the time shifts implementation plan is presented.

The flowchart for the initial testing is depicted in Figure 5.8. This chart illustrates the basic steps to solve the transfer optimization problem. The flowchart includes the calculation of transfer times, which are used to measure the time savings from previous iterations.
Figure 5.8 Flowchart for the Initial Testing
Using a Dell Precision M60 with a Pentium M processor 1.7 Ghz, 1GB RAM, and a 60GB (7200 rpm) hard drive, several runs of the computer program were made and the median values of the total transfer time with their corresponding CPU times were obtained.

Figure 5.9 shows the results of these iterations by plotting transfer times against CPU time. The line labeled “Curve Fit” is used for illustration purposes to display a uniform trajectory of the optimization improvements. The original transfer time was 3,150 hours per day. Significant improvements were obtained within a few hours of CPU time, with a total transfer time savings of 231.78 hours per day after 2,000 iterations.

The time shifting implementation plan for all routes is depicted in Table 5.5, which shows in minutes the time that the timetables will need to be shifted in order to be optimized for transfers. Using this methodology, the total transfer times were reduced
from 3,150.37 to 2918.59 hours per day, representing a 7.36% improvement. Savings beyond 8.11 hours of CPU time were less significant, at the cost of additional computational time.

The initial case study consisted of 40 routes (80 directional routes) and 255 transfer stops, using scheduling data from BCT. The results show that greater transfer-time savings may be potentially obtained. Based on these results, it is expected that the efficiency of the algorithm and the time savings will be greater by using a rigorous methodology based on the genetic algorithm.

Table 5.5 Implementation Plan of Timetable Shifting

<table>
<thead>
<tr>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-5</td>
<td>11</td>
<td>4</td>
<td>21</td>
<td>6</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>-8</td>
<td>12</td>
<td>-3</td>
<td>22</td>
<td>-1</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>13</td>
<td>-9</td>
<td>23</td>
<td>-1</td>
<td>33</td>
<td>-12</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>14</td>
<td>3</td>
<td>24</td>
<td>-3</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>15</td>
<td>5</td>
<td>25</td>
<td>-11</td>
<td>35</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>-11</td>
<td>16</td>
<td>-13</td>
<td>26</td>
<td>4</td>
<td>36</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>-12</td>
<td>17</td>
<td>-5</td>
<td>27</td>
<td>6</td>
<td>37</td>
<td>-5</td>
</tr>
<tr>
<td>8</td>
<td>-8</td>
<td>18</td>
<td>-6</td>
<td>28</td>
<td>1</td>
<td>38</td>
<td>-15</td>
</tr>
<tr>
<td>9</td>
<td>-6</td>
<td>19</td>
<td>1</td>
<td>29</td>
<td>1</td>
<td>39</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>5</td>
<td>30</td>
<td>-1</td>
<td>40</td>
<td>-6</td>
</tr>
</tbody>
</table>

5.4 Results from the Genetic Algorithm

The genetic algorithm was tested using the same datasets of ridership and schedule deviations. The schedule deviation table used values of zero to allow the comparison of results from the GA with the existing transfer times as well as the results from the random brute force approach on the same basis, since one of the goals of this research was to measure the efficiency and effectiveness of the genetic algorithm.
The computer program was run on the same computer. The population size is 80 and the GA ran 20 generations. Different crossover and mutation rates were tested. The average values of the total transfer time with their corresponding CPU times were obtained. Figure 5.10 shows the results from these iterations by plotting the transfer times against the CPU time. The existing transfer time was 3,150 hours per day. Recall that significant improvements were obtained within a few hours of CPU time after 2,000 iterations using a random brute force search, with a total transfer time savings of 231.78 hours per day. The GA was applied, the time savings were increased to 413.09 hours. This demonstrates that the GA is both more effective and more efficient.

Figure 5.10  Computational Results

Figure 5.11 plots the number of chromosomes and generations used in the calculations of transfer times against the transfer times obtained. It may be seen that the solution
continues to improve when either the number of generations, chromosomes, or both are increased.

Figure 5.11  Number of Chromosomes and Generations

A time shifting implementation plan for all routes based on the output from the GA is shown in Table 5.6, which gives the time in minutes that the timetables will need to be shifted in order to achieve the desired results obtained from the GA. Using this methodology, the total transfer times were reduced from 3,150.37 to 2,737.28 hours per day, representing a significant 13.1% improvement. Savings beyond the 8.24 hours of CPU time were less significant, at the cost of additional computational time.
Table 5.6  GA Implementation Plan of Timetable Shifting

<table>
<thead>
<tr>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
<th>Routes</th>
<th>Time Shifting (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>21</td>
<td>0</td>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>-9</td>
<td>12</td>
<td>-13</td>
<td>22</td>
<td>0</td>
<td>32</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-12</td>
<td>13</td>
<td>0</td>
<td>23</td>
<td>-10</td>
<td>33</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>14</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>25</td>
<td>-13</td>
<td>35</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>-9</td>
<td>16</td>
<td>-13</td>
<td>26</td>
<td>15</td>
<td>36</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>-6</td>
<td>17</td>
<td>-9</td>
<td>27</td>
<td>-13</td>
<td>37</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>-9</td>
<td>18</td>
<td>-7</td>
<td>28</td>
<td>-2</td>
<td>38</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>19</td>
<td>12</td>
<td>29</td>
<td>-1</td>
<td>39</td>
<td>-10</td>
</tr>
<tr>
<td>10</td>
<td>-13</td>
<td>20</td>
<td>15</td>
<td>30</td>
<td>11</td>
<td>40</td>
<td>7</td>
</tr>
</tbody>
</table>

The efficiency of the genetic algorithm is affected by its parameters. To evaluate the effects of the crossover rate and mutation rate on the results, different scenarios were tested. This involved holding one parameter constant and varying the other. The crossover rate was varied from 10% to 90%, and the mutation rate was also varied from 10% to 90%. A total of ten (10) scenarios of their combinations are tested. The results are shown in Figure 5.12 and 5.13. Figure 5.12 plots the transfer times against the crossover rate. It appears that when the mutation rate is held at a constant of 10%, the crossover rate of 50% produced best results. Figure 5.13 presents the relationship between and the mutation rate and the transfer time. It may be seen when the crossover rate was 50%, a mutation rate of 10% produced the best results.

A series of additional tests were performed to assess the genetic algorithm capabilities, its performance, and its sensitivity to the variation of selected elements. The main purpose of these tests was to ensure that the proposed methodology was capable of finding an optimal solution. Another purpose is to assess the capabilities of the computer
application and its ability of performing sensitivity analysis. These tests and the results are described in the subsequent sections.

Figure 5.12 Crossover Rates

Figure 5.13 Mutation Rates
Another parameter of the GA is population size. To evaluate the impact of the number of chromosomes (population size) on the optimization result and on the algorithm efficiency, the same data set consisting of 40 routes was run using 50% crossover rate and 10% mutation rate. To determine the effect of population size, the number of generations was kept a constant of 20. Computer runs were performed for various numbers of chromosomes from 10 to 140 at an interval of 10. Figure 5.14 shows the results. In the legend, the G stands for generation and the C for chromosomes. For instance, 20G_10C means a 20-generation run with 10 chromosomes.

![Impact of Chromosomes on Transfer Times](image)

Figure 5.14 Impacts of Population Size

As expected, the larger the population size, that is the number of chromosomes, the better the results. Although not clearly shown in Figure 5.14, the transfer time savings based on population size larger than 40 are relatively close. The computing time difference,
however, between a population of 40 chromosomes and that of 140 was close to 12 hours, using a Dell Precision M90 with a dual processor. One interesting observation was that the best result was obtained with 90 chromosomes, rather than with 140 chromosomes. This means that although there is a greater probability of getting better results with a larger population and more generations, the GA does not give full assurance that this will be the case. This is due to the randomness inherent in genetic algorithms, particularly in the crossover and mutation processes.

To visualize the variation of gene values (time shifts), Figure 5.15 provides the results from the GA based on a population of 90 chromosomes. In this figure, each row represents one chromosome. Column TT gives the total transfer times in hours, sorted in ascendant order. The rest of the columns represent the gene values for each chromosome. It may be observed that the gene values look increasingly alike as they converge to the optimal solution.

5.5 Global Optimality of Solutions

To determine if the algorithm is capable of reaching an optimal solution, a small test case was constructed and an exhaustive sequential search was performed. The result was compared with the genetic algorithm. The small test case consisted of six (6) directional routes and ten (10) trips, with five (5) minute shifts and three possible values (-5, 0, 5). The exhaustive sequential search was set up to cover all possible transfers in the network. The sequential search program evaluated all the possible solutions in approximately
3,300 seconds of CPU time and an optimal solution was found. The total transfer time in this test improved approximately 5%, from 104.22 hours to 99.24 hours.

Figure 5.15 Variations of Gene Values
The same data set was applied to the GA to see if it was capable of finding the same solution in the same or less time as the sequential search did. The GA found the solution in approximately 30.03 seconds when all the 100 solutions converged to the global optimum. Nevertheless, the GA found the global optimal solution in six (6) of the 100 chromosomes after only one iteration, in 1.61 seconds of CPU time. This demonstrates that GA is capable of finding global optimal solutions, although finding global optima is not guaranteed.

5.6 Monte Carlo Simulation

To test the ability of the methodology to simulate the randomness in bus arrivals and departures, using the existing data from the BCT Automatic Vehicle Location system, an experiment was conducted using the small data set of 10 routes, 20 generations, a crossover rate of 50%, and a mutation rate of 10%. The simulated arrival and departure deviations, as described in Section 5.2.2, are calculated for each arriving and departing bus. These simulated values are added to the schedule times in the timetables to compute new arriving and departing times that would represent actual arrival and departing times. The GA process uses the new arriving and departing times in the calculations of transfer times.

The simulation process was successful. However, it took, on average, about 40 times more computer time than the GA without the Monte Carlo Simulation incorporated. Table 5.7 provides the results of using Monte Carlo Simulation, and Figure 5.16 shows the convergence of the GA for this small problem.
### Table 5.7 Monte Carlo Results

<table>
<thead>
<tr>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exist. Total TT = 104.22 Hours</td>
</tr>
<tr>
<td>Final Total TT = 95.67 Hours</td>
</tr>
<tr>
<td>Number of Generations = 20</td>
</tr>
<tr>
<td>Number of Chromosomes = 10</td>
</tr>
<tr>
<td>Savings = 8.55 Hours (8.20%)</td>
</tr>
</tbody>
</table>

#### Time Shifts

<table>
<thead>
<tr>
<th>Genes (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genes (1) = 0</td>
</tr>
<tr>
<td>Genes (2) = -5</td>
</tr>
<tr>
<td>Genes (3) = -5</td>
</tr>
</tbody>
</table>

Computing Time = 841.47 sec.

---

**Figure 5.16 Convergence of GA with Monte Carlo Simulation**
5.7 Partial Optimization

In practice, it may be infeasible, or at least reluctant on the part of a transit agency, to modify the timetables of all routes all at once. This is due to the sensitivity in modifying certain routes, either to avoid user confusion or because of riders with special needs. The program therefore is equipped with the ability to allow a selected route or group of routes in the system to remain fixed. In other words, only the time tables of selected routes are optimized. This feature is useful in practice, as it gives transit agencies the ability to determine which routes are too sensitive to change their schedules. In the algorithm, the selected timetables can be kept constant by not adding time shifts to the calculations. The parameters used in this test are given below:

- 40 Routes (80 directional Routes)
- 4 Generations
- 20 Chromosomes
- 50% Cross Over Rate
- 10% Mutation Rate
- 15-minute range at 1-minute intervals

For the purpose of this experiment, only the first ten (10) out of 40 routes were optimized; the rest of the routes were kept constant. Table 5.8 shows the output from the computer program. Note that the time shifts for route numbers greater than 10 are zeroes, which means that the timetables of those routes are not modified.
Table 5.8  Optimization of 10 Routes

<table>
<thead>
<tr>
<th>Gene</th>
<th>Time Shifts (Minutes)</th>
<th>Gene</th>
<th>Time Shifts (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Computing Time = 1667.95 seconds

Results from the 10-route optimization are compared with the results from a system-wide optimization using the same data and parameters. It is found, as expected, that optimizing the schedules for 10 routes only results in 122 hours of transfer time savings.
which is about 59% of the 207 hours of savings from system-wide optimizations. This is quite significant given that only 25% of the routes have their schedules optimized while the savings obtained is 59%. However, this result is not guaranteed, since there is no a priori knowledge as which routes might produce the greatest benefits. Therefore, an interesting question is whether it is possible to find the best $r$ routes out of $n$ routes that would produce the greatest transfer time savings. This is a combinatorial problem in which all the possible $n$ elements (e.g., 40 routes) may be combined into groups of $r$ items (e.g., 10 routes). The number of possible combination is

$$nC_r = \frac{n!}{r!(n-r)!}$$

For a 40 route system with the timetables of 10 routes to be optimized, the total number of combinations needed to be evaluated to determine the best 10 is $8.48E+08$ ($40C_{10}$). Figure 5.17 plots the total number of combinations needed to select the best group of routes out a total of 40 routes if every possible solution is to be evaluated. For instance, if 15 out the 40 routes were to remain unmodified, i.e., 25 may be changed, there will be $4.02E+10$ possible combinations that will need to be compared to determine which case will produce the best solution.

The computer application can also generate reports that provide insightful and detailed information on the transfer time calculations. This is useful for determining the time and locations where optimizations occur.
For instance, Table 5.9 gives the transfer times for all trips at the transfer location between Route 1 and Route 5. The first column shows the arrival times in hours for the buses on Route 1 and the second column the departure times for the buses on Route 5. The third column describes the transfer trips, the routes involved, and the timepoints on the two routes. The fourth and fifth columns are the time shifts for the two routes involved at the transfer location. Column six lists the average number of transfer passengers for each bus trip at that location. The last column gives the transfer times if the shifts are implemented. As an example, in the first row, a bus (first trip – r1) on Route 1 (R1) arrives at 5:20 am (5.33 h) at timepoint 2 (c2), and a second bus (fifth trip – r5) on Route 5 (R5) departs at 5:45 am (5.75 h) from timepoint 1 (c1). The total transfer
time at this location, labeled as R1-R5 r1, c2, r5, c1 in the third column, is 0.31 hours. This is computed as \([(5.75 + 15/60) - (5.33 + 2/60)] \times 0.46\). That is, scheduled departure times plus its time shift minus scheduled arrival times plus its corresponding time shift, times the ridership between the two routes.

In addition to detailed transfer time calculation, a summary of transfer times among routes can also be obtained. Table 5.10 depicts 10 route pairs that produced the greatest time savings after 20 generations of a run with 90 chromosomes. The first column lists a pair of routes identified by their sequentially assigned sequence numbers. The second column gives the actual route numbers. The third column describes the locations of transfer points. The fourth and fifth columns show the average transfer ridership per trip and per day. The values in columns sixth and seventh represent the total and average time savings for all the transfer trips between the route pairs. The last column in Table 5.10, percent improve, represents the improvement in transfer times compared with the existing transfer times.

The routes presented in Table 5.10 represented some of the busiest routes in the system and at locations with high ridership. For instance, the transfer from Route 18 Outbound to Route 36 Outbound occurs at Lauderhill Mall, which is a bus transfer facility with high activity.
## Table 5.9 GA Sample Results

<table>
<thead>
<tr>
<th>Arrivals (hour)</th>
<th>Departures (hour)</th>
<th>Transfer Description</th>
<th>Time Shifts (Minutes)</th>
<th>Average Riders</th>
<th>Transfer Times (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.33</td>
<td>5.75</td>
<td>[R1-R5 r1,c2,r5,c1]</td>
<td></td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>5.67</td>
<td>5.75</td>
<td>[R1-R5 r2,c2,r5,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>6.00</td>
<td>6.25</td>
<td>[R1-R5 r3,c2,r6,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>6.33</td>
<td>6.75</td>
<td>[R1-R5 r4,c2,r7,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>6.67</td>
<td>6.75</td>
<td>[R1-R5 r5,c2,r7,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>7.00</td>
<td>7.25</td>
<td>[R1-R5 r6,c2,r8,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>7.33</td>
<td>7.75</td>
<td>[R1-R5 r7,c2,r9,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>7.67</td>
<td>7.75</td>
<td>[R1-R5 r8,c2,r9,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>8.00</td>
<td>8.25</td>
<td>[R1-R5 r9,c2,r10,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>8.33</td>
<td>8.75</td>
<td>[R1-R5 r10,c2,r11,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>8.67</td>
<td>8.75</td>
<td>[R1-R5 r11,c2,r11,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>9.00</td>
<td>9.25</td>
<td>[R1-R5 r12,c2,r12,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>9.33</td>
<td>9.75</td>
<td>[R1-R5 r13,c2,r13,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>9.67</td>
<td>9.75</td>
<td>[R1-R5 r14,c2,r13,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>10.00</td>
<td>10.25</td>
<td>[R1-R5 r15,c2,r14,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>10.33</td>
<td>10.75</td>
<td>[R1-R5 r16,c2,r15,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>10.67</td>
<td>10.75</td>
<td>[R1-R5 r17,c2,r15,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>11.00</td>
<td>11.25</td>
<td>[R1-R5 r18,c2,r16,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>11.33</td>
<td>11.75</td>
<td>[R1-R5 r19,c2,r17,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>11.67</td>
<td>11.75</td>
<td>[R1-R5 r20,c2,r17,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>12.00</td>
<td>12.25</td>
<td>[R1-R5 r21,c2,r18,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>12.33</td>
<td>12.75</td>
<td>[R1-R5 r22,c2,r19,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>12.67</td>
<td>12.75</td>
<td>[R1-R5 r23,c2,r19,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>13.00</td>
<td>13.25</td>
<td>[R1-R5 r24,c2,r20,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>13.33</td>
<td>13.75</td>
<td>[R1-R5 r25,c2,r21,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>13.67</td>
<td>13.75</td>
<td>[R1-R5 r26,c2,r21,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>14.00</td>
<td>14.25</td>
<td>[R1-R5 r27,c2,r22,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>14.33</td>
<td>14.75</td>
<td>[R1-R5 r28,c2,r23,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>14.67</td>
<td>14.75</td>
<td>[R1-R5 r29,c2,r23,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>15.00</td>
<td>15.25</td>
<td>[R1-R5 r30,c2,r24,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>15.33</td>
<td>15.75</td>
<td>[R1-R5 r31,c2,r25,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>15.67</td>
<td>15.75</td>
<td>[R1-R5 r32,c2,r25,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>16.00</td>
<td>16.25</td>
<td>[R1-R5 r33,c2,r26,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>16.33</td>
<td>16.75</td>
<td>[R1-R5 r34,c2,r27,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>16.67</td>
<td>16.75</td>
<td>[R1-R5 r35,c2,r27,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>17.00</td>
<td>17.25</td>
<td>[R1-R5 r36,c2,r28,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>17.33</td>
<td>17.75</td>
<td>[R1-R5 r37,c2,r29,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>17.67</td>
<td>17.75</td>
<td>[R1-R5 r38,c2,r29,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>18.00</td>
<td>18.25</td>
<td>[R1-R5 r39,c2,r30,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>18.33</td>
<td>18.75</td>
<td>[R1-R5 r40,c2,r31,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>18.67</td>
<td>18.75</td>
<td>[R1-R5 r41,c2,r31,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>19.00</td>
<td>19.17</td>
<td>[R1-R5 r42,c2,r32,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>19.33</td>
<td>20.08</td>
<td>[R1-R5 r43,c2,r33,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>19.92</td>
<td>20.08</td>
<td>[R1-R5 r44,c2,r33,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
<tr>
<td>20.50</td>
<td>21.00</td>
<td>[R1-R5 r45,c2,r34,c1]</td>
<td>0</td>
<td>15</td>
<td>0.46</td>
</tr>
</tbody>
</table>
Table 5.10 Top 10 Optimized Route Pairs

<table>
<thead>
<tr>
<th>Route Pair</th>
<th>BCT Route Pair*</th>
<th>Location</th>
<th>Transfer Ridership per Trip</th>
<th>Transfer Ridership Per Day</th>
<th>Total Savings (Hours)</th>
<th>Average Savings (Min.)</th>
<th>Percent Improve</th>
</tr>
</thead>
<tbody>
<tr>
<td>62, 63</td>
<td>360B-400B</td>
<td>Lauderhill Mall</td>
<td>5.14</td>
<td>257.00</td>
<td>50.46</td>
<td>11.78</td>
<td>62.17%</td>
</tr>
<tr>
<td>70, 48</td>
<td>720B-100B</td>
<td>US1 &amp; Oak Park Blvd.</td>
<td>1.76</td>
<td>88.00</td>
<td>23.4</td>
<td>15.95</td>
<td>83.30%</td>
</tr>
<tr>
<td>66, 2</td>
<td>560B-21B</td>
<td>West Regional Terminal</td>
<td>3.04</td>
<td>88.16</td>
<td>21.8</td>
<td>14.84</td>
<td>100.00%</td>
</tr>
<tr>
<td>62, 14</td>
<td>360B-181B</td>
<td>Lauderhill Mall</td>
<td>4.44</td>
<td>222.00</td>
<td>20.34</td>
<td>5.50</td>
<td>53.20%</td>
</tr>
<tr>
<td>41, 63</td>
<td>10B-400B</td>
<td>Broward Terminal</td>
<td>2.71</td>
<td>130.08</td>
<td>20.19</td>
<td>9.31</td>
<td>43.42%</td>
</tr>
<tr>
<td>35, 50</td>
<td>881B-12OB</td>
<td>West Regional Terminal</td>
<td>3.67</td>
<td>62.39</td>
<td>18.35</td>
<td>17.65</td>
<td>83.33%</td>
</tr>
<tr>
<td>49, 48</td>
<td>11OB-100B</td>
<td>Broward Terminal</td>
<td>2.05</td>
<td>69.70</td>
<td>17.26</td>
<td>14.86</td>
<td>54.60%</td>
</tr>
<tr>
<td>66, 42</td>
<td>560B-20B</td>
<td>West Regional Terminal</td>
<td>1.71</td>
<td>49.59</td>
<td>17.14</td>
<td>20.74</td>
<td>80.70%</td>
</tr>
<tr>
<td>49, 56</td>
<td>11OB-22OB</td>
<td>Broward Terminal</td>
<td>2.37</td>
<td>80.58</td>
<td>16.18</td>
<td>12.05</td>
<td>90.90%</td>
</tr>
<tr>
<td>19, 2</td>
<td>30IB-21B</td>
<td>West Regional Terminal</td>
<td>1.40</td>
<td>47.60</td>
<td>15.89</td>
<td>20.03</td>
<td>70.94%</td>
</tr>
</tbody>
</table>

*IB=INBOUND, OB=OUTBOUND

Table 5.11 lists the top 13 routes that correspond to the top 10 route pairs in Table 5.10. This table gives the route numbers in the GA and from the actual system, the existing total daily trips, the headways, and the time shifts that have been optimized. In the last column, an index is assigned to each route, based on whether it is part of the route pairs that realize the largest transfer time savings. It needs to be pointed out that no savings can be attributed to a single route because transfers always involve a pair of routes. Table 5.12 shows the ranking of the routes and their involvement in the time savings. The first column identifies the 13 routes. The next three columns give the transfer time savings that they contribute to. The fourth column is the total time savings each route has contributed. The last column is the ranking. Note that some of the average savings in Table 5.10 are close to the headway values in Table 5.11. This is mainly due to the fact that most trips from the connecting routes were scheduled at the same time and the algorithm used a walking time constraint. Therefore, most of these transfers were missed, having to wait for the next bus. This points the need to obtain more accurate walking time at different locations, especially at transfer locations where schedules may
have been coordinated and where the prevailing conditions at other transfer locations do not apply.

It is worthwhile to point out that the benefits from transfer time optimization will not be uniform to all transit users. At certain locations, the benefits will be greater than those at other locations. It is also possible that transfer time at a few locations might even get longer, especially where ridership is low.

Table 5.11 Operating Characteristics of Top 13 Routes

<table>
<thead>
<tr>
<th>GA Routes</th>
<th>BCT Routes</th>
<th>Daily Trips</th>
<th>Headways (Min.)</th>
<th>Time Shifts (Min.)</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>48</td>
<td>20</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>38</td>
<td>30</td>
<td>-9</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>35</td>
<td>30</td>
<td>-9</td>
<td>4</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>34</td>
<td>30</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>31</td>
<td>40</td>
<td>-13</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>18</td>
<td>31</td>
<td>15</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
<td>50</td>
<td>20</td>
<td>-13</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>30</td>
<td>34</td>
<td>30</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>36</td>
<td>50</td>
<td>20</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>40</td>
<td>62</td>
<td>30</td>
<td>-10</td>
<td>2</td>
</tr>
<tr>
<td>26</td>
<td>56</td>
<td>29</td>
<td>30</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>72</td>
<td>50</td>
<td>20</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>35</td>
<td>88</td>
<td>17</td>
<td>45</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5.12 Ranking of Routes by Their Involvement in Transfer Time Savings

<table>
<thead>
<tr>
<th>Route</th>
<th>Transfer Time Saving (hours)</th>
<th>Total Savings (hours)</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st Route</td>
<td>2nd Route</td>
<td>3rd Route</td>
</tr>
<tr>
<td>1</td>
<td>18.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>19.80</td>
<td>17.14</td>
<td>15.89</td>
</tr>
<tr>
<td>8</td>
<td>23.40</td>
<td>17.26</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>17.26</td>
<td>16.18</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>18.35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>20.34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16.18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>15.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>50.46</td>
<td>20.34</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>50.46</td>
<td>20.19</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>21.80</td>
<td>17.14</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>23.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>18.35</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The computer application also allows for testing different schedule shift scenarios. This can be easily accomplished by simply entering the desired time shifts into a specific table in the database. The change in the transfer time can be obtained by pressing the Manual Calc. button on the graphical user interface (see Figure 4.7).

5.8 Transfer Times of Best Performing Routes

Intuitively, the top 13 routes, in Table 5.11 would account for most of the transfer time savings resulted from the optimization, as they were derived from the top 10 route pairs given in Table 5.10. However, the savings for the top 10 route pairs in the original test problem of system-wide optimization were obtained when time shifts also occurred for the other routes in the system. Therefore, to assess the time savings impact of these top 13 routes, the time shift values from Table 5.11 were used to calculate their transfer times. The result confirms that these 13 routes did generate most of the time savings. In fact, the total savings for all 13 routes is 243.45 hours, which is close to 60% of the total system-wide savings. Figure 5.18 depicts the transfer time savings based on the number of routes shifted. The number on the x-axis represents the priority index of the top 13 routes and the values on the y-axis represent the cumulative time savings of these routes. The index was created based on the routes that contributed the most to the transfer time savings, with priority index 1 being the route that was involved in the most savings. This figure suggests that savings at different locations are uneven, as there are other routes involved in the overall calculations.
It would be interesting to find out what the transfer time savings will be if the time shifts of these routes are included in the GA initial population. The hypothesis for this test is that using these routes as an initial solution would have a positive impact on either the overall optimization or the performance of the GA. Therefore, to test the impact of these top 13 routes on the system-wide optimization, their time shift values were included in the chromosomes of the first generation.

The GA program was run with 20 generations and 80 chromosomes. The results of this experiment are shown in Figure 5.19. In the figure, previous results from the GA optimization depicted in Figure 5.10 are also plotted for comparison. It may be seen that the partially optimized initial population began at a much lower total transfer time. However, this did not lead to a better solution as it gradually flattened out and approached the same time savings obtained from the previous GA optimization.
However, the optimal solution was consistently found within 6.17 hours of CPU time with a significant improvement of approximately 25%.

Figure 5.19 Results of Using Top 13 Routes as Initial Population
CHAPTER 6

SUMMARY AND CONCLUSIONS

Most of the research in this area has been focused on timed transfers at a terminal or designated facilities where most transfers occur, or on theoretical formulations of the problem. The research described in this dissertation introduces a systematic approach to the synchronization of transfers in a transit system with multiple bus routes and numerous time points. The methodology was developed considering current scheduling practices and makes use of the genetic algorithm to find an optimal solution. The methodology has been implemented as a computer application that is flexible in terms of the type of optimization performed and may be easily improved or modified. This facilitates the incorporation of new ideas and computing methods. For instance, a route or a group of routes can be kept unchanged, giving schedulers the flexibility of leaving the timetables of sensitive routes as they are.

Using actual data from a transit system with 40 bus routes (or 80 directional routes) and 255 transfer stops, the proposed GA-based approach was tested and evaluated, and was demonstrated to be capable of solving the transfer synchronization problem, which is difficult to solve with classical optimizing algorithms. The results show that significant transfer time savings can be obtained. The approach is computationally feasible and is relatively easy to implement. Because it is easily understood and can be directly incorporated into the scheduling process, it is more likely to be accepted by transit
professionals than results from methods that do not consider the existing schedules or change the entire network layout.

While on average the genetic algorithm is capable of finding an optimal solution and at a much faster rate than the random search, an optimal solution may not be always found. This sometimes is due to the fact that genetic algorithms stop after running for a certain amount of time or when no improvements have been observed. This is one of the weaknesses of the algorithm, as it is difficult to know with certainty when the optimal solution will be reached. Nevertheless, GAs are still a viable solution to problems that cannot be solved or are difficult to be solved by other methods.

The inclusion of schedule deviations in the optimization using a Monte Carlo Simulation proved feasible, but time consuming. To be practical for real-world applications, the algorithm must be much faster. If there is a need to simulate schedule deviations, this may be done as a separate process to generate simulated schedule deviation values similar to the data collected from AVL or APC systems.

With minor modifications in the code of the computer program, the optimization of all routes, a route, or a group of selected routes is possible. The results from the numerical experiments involving 10 routes produced, as expected, smaller time savings than if the schedules of all the routes were optimized (i.e., 122.28 Hours vs. 207.73 Hours).
It is generally accepted that a larger population will produce better results because the chance of getting trapped into a local optimum is reduced due to an expanded search. However, perhaps due to the randomness in the genetic algorithm, there is no guarantee that a larger population always leads to better results. For instance, results from one of the tests, shown in Figure 5.14, show that the best result was obtained with a population of 90 chromosomes instead of the one with 140 chromosomes.

The GA method proposed in this dissertation is able to find a good solution in a reasonable amount of time, but the computing time is still long. In particular, the computer run for the case with 20 generations and a population size of 140C took about 15 hours, using a Dell Precision M90 with a Centrino Duo processor and 2 MB of memory. However, it is expected that with advancements in computing technologies and faster processors, the computing time will be reduced.

The computer application allows transfer times to be computed at the trip level; thus, locations where transfer-time savings occur can be easily identified. This information provides additional opportunities in determining which locations can produce the greatest benefits. Furthermore, the application allows the user to test what-if scenarios by entering a possible solution and quickly estimating transfer time savings. This is a useful feature that provides quick feedback on potential improvements at particular location and when there is no time to run the entire genetic algorithm process.
Future research will involve investigations of possible improvements of the genetic algorithm in terms of selection of the initial population, optimal crossover and mutation rates, number of generations, and chromosomes, and incorporating the migration operator. Sensitivity analysis may also be performed to identify which genes in a chromosome are conducive to producing higher transfer time savings. This will allow transit agencies to concentrate on route combinations that produce greater benefits.

Additional research into innovative algorithms like simulated annealing or other evolutionary strategies are also worth pursuing to improve the efficiency of the computational process. These algorithms could be applied in combination with a genetic algorithm.

Parallel computing presents great opportunities to increase the computational efficiency of the model. By having several computers running simultaneously, the GA process can be much faster. For instance, the algorithm could be setup to use a common database, so that the best chromosomes from each computer at a particular generation can be incorporated into the population of the other computers. This type of computational methods can help reduce the time needed to produce solutions.
The data gathering and manipulation process may be improved to reduce the amount of effort required, particularly obtaining Ridership and schedule adherence data. The best approach would be for the vendors of Advanced Public Transportation Systems (APTS) technologies to continue improving their products and provide transit agencies with more sophisticated tools to allow for advanced data mining capabilities.

Another area that needs to be investigated further is the determination of actual walking times. In this research, averages walking times were used in the case study. In practice, there is a need to use real values to avoid suboptimal or erroneous results. For instance, if the average walking time is too long, it reduces the time that can be optimized. On the other hand, if the time is too small, there is a possibility that transferring passengers miss the connection, thus defeating the purpose of the whole optimization. Ideally, a set of walking times based on field data is needed, which may be estimated using GIS and considering roadway geometry and traffic signalization.

Lastly, the current model uses one transfer location between each pair of directional routes. However, routes may cross at several locations and at different times during a day. Reflecting the reality of multiple locations for transfers is an area for future research. While the connectivity matrix $C_{ijkl}$ may be modified to allow for multiple transfer locations, the key will be the collection of ridership data, which may become easier and better in the future as transit agencies continue to deploy Advanced Public Transportation Systems (APTS).
LIST OF REFERENCES


http://www.economicsnetwork.ac.uk/cheer/ch13_1/ch13_1p16.htm


http://cs.felk.cvut.cz/~xobitko/ga


APPENDIX A

SAS Code

/***********************************************************
* Fabian Cevallos
* Ph.D. Dissertation
* This procedure uses Schedule Deviation Data to generate
* Normal, Lognormal, Gamma, and Weibull Distributions. *
*******************************************************************/
ods listing;
ods html;
title j=center color=black height=16pt "Bus Schedule Deviations";
title2 j=center color=black height=12pt "BCT Route 1 NB at Broward Terminal";
title3 j=center color=black height=10pt "Weekdays (7:00 AM - 9:00 AM)";
footnote j=left height=10pt color=black "Prepared by: Fabian Cevallos";
footnote2 j=left height=10pt color=black "Ph.D. Dissertation";
footnote3 j=center height=10pt color=black "%sysfunc(date(),worddate.)";
goptions ftext=SWISS ctext=BLACK htext=1.5 cback=White cells;
symbol v=SQUARE c=BLUE h=1 cells;
proc univariate data=PhD_data.F_bt_rtel_nB gout= PhD_out.F_bt_rtel_nB
noprint;
label Deviation = 'Deviation (min)';
var Deviation;
HISTOGRAM Deviation/ caxes=BLACK waxis= 1
cbarline=BLACK
vaxislabel = 'Frequency'
midpoints = -10 to 10 by 1
vaxis = 0 to 5 by 1
vscale = count
   LOGNORMAL (theta=est zeta=est sigma=est w=2 l=1)
   NORMAL (mu=est sigma=est w=2 l=2)
   GAMMA (theta=est sigma=est alpha=est w=2 l=3)
   WEIBULL (theta=est sigma=est c=est w=2 l=4);
inset LOGNORMAL NORMAL GAMMA WEIBULL / pos=ne header ='Legend'
position = NE height = 2;
run;
ods html close;
ods listing close;
ods listing;
  title1 'Lognormal Probability Plot for Random Bus Arrivals';
  title2 'Route 1 NB at Broward Terminal';
  proc univariate data=PhD_data.F_bt_rtel_NB noprint;
    probplot Deviation / lognormal(theta=est zeta=est sigma=est)
      grid
        lgrid = 3
        vminor = 4
        href = 95
      lhret = 1
      chref = red
      cframe = ligr
      square;
  run;
ods listing close;
Appendix B

Excel Macro

Rem Random Bus Arrival/Departure Generator
Rem Fabian Cevallos
Rem Ph.D. Dissertation
Rem Florida International University

Sub MonteCarlo()

Windows("MonteCarlo_Bus_Arriv.xls").Activate

For i = 1 To 10

For j = 2 To 56

Sheets("Processing").Select

Dev_Cell = "A" & j
Sim_Cell = "C" & j

Range("F1").Select
ActiveCell.FormulaR1C1 = "=Rand()"
Rem Uniform Random number generator

Range("G1").Select
ActiveCell.FormulaR1C1 = "=LOGINV(RC[-1], LogNormal!R[7]C[-3],LogNormal!R[8]C[-3])"
Rem Loginv(p, mean, sd) - Use the normal distributed values of the Mean and
Rem Standard Deviation from y = ln(x)

LogInv_Val = Range("G1").Value
Rem Limit extreme tail values using the values from p=0.10 and p=0.90 in LogInv

If LogInv_Val > 12.8214638 And LogInv_Val < 25.78820906 Then
    Range(Sim_Cell).Value = LogInv_Val
Else
End If

Next j

Sheets("Processing").Select
Columns("A:A").Select
Selection.Copy
Sheets("Result").Select
Columns("A:A").Select
ActiveSheet.Paste
Sheets("Processing").Select
Columns("C:C").Select
Selection.Copy
Sheets("Result").Select
Columns("C:C").Select
ActiveSheet.Paste

Next i
End Sub

Excel Help

LOGINV

Returns the inverse of the lognormal cumulative distribution function of x, where ln(x) is normally distributed with parameters mean and standard_dev. If p = LOGNORMDIST(x,...) then LOGINV(p,...) = x.

Use the lognormal distribution to analyze logarithmically transformed data.

Syntax

LOGINV(probability,mean,standard_dev)

Probability is a probability associated with the lognormal distribution.

Mean is the mean of ln(x).

Standard_dev is the standard deviation of ln(x).

Remarks

If any argument is nonnumeric, LOGINV returns the #VALUE! error value.
If probability < 0 or probability > 1, LOGINV returns the #NUM! error value.
If standard_dev <= 0, LOGINV returns the #NUM! error value.

The inverse of the lognormal distribution function is:

$LOGINV(p, \mu, \sigma) = e^{\mu + \sigma \sqrt{2} \cdot \text{NORMSINV}(p)}$
Appendix C

Transfer Optimization Code

TT.wfm

** END HEADER -- do not remove this line

//
// Generated on 05/28/2005
//

parameter bModal
local f
f = new TTForm()
if (bModal)
   f.mdi = false // ensure not MDI
   f.readModal()
else
   f.open()
endif

class TTForm of FORM
   with (this)
      onOpen = class::FORM_ONOPEN
      height = 17.28
      left = 50.8571
      top = 0.2273
      width = 75.0
      text = "Transfer Optimization"
      autoCenter = true
      mdi = false
      sizeable = false
      maximize = false
   endwith

   this.RECTANGLE3 = new RECTANGLE(this)
   with (this.RECTANGLE3)
      left = 7.0
      top = 12.72
      width = 42.0
      height = 4.0
      text = "Output"
      fontBold = true
      fontItalic = true
   endwith

   this.RECTANGLE2 = new RECTANGLE(this)
   with (this.RECTANGLE2)
      left = 9.0
      top = 7.08
      width = 38.0
      height = 5.0
      text = "Options"
      fontBold = true
      fontItalic = true
   endwith

   this.RECTANGLE1 = new RECTANGLE(this)
   with (this.RECTANGLE1)
      left = 1.0
      top = 0.24
      width = 53.0
      height = 3.2727
      text = "Parameters"
      fontBold = true
      fontItalic = true
   endwith

   this.ROUTESPIN = new SPINBOX(this)
   with (this.ROUTESPIN)
height = 1.0
left = 2.0
top = 2.0
width = 12.0
step = 2.0
rangeMax = 10000
rangeMin = 2
value = 80
spinOnly = true
rangeRequired = true
endwith
this.RUNSPIN = new SPINBOX(this)
with (this.RUNSPIN)
  height = 1.0
  left = 20.0
  top = 2.0
  width = 12.0
  rangeMax = 1000000
  rangeMin = 1
  value = 20
  spinOnly = true
  rangeRequired = true
endwith
this.CHROMSPIN = new SPINBOX(this)
with (this.CHROMSPIN)
  height = 1.0
  left = 38.0
  top = 2.0
  width = 12.0
  step = 2.0
  rangeMax = 10000
  rangeMin = 4
  value = 50
  spinOnly = true
  rangeRequired = true
endwith
this.RESETBUTTON = new PUSHBUTTON(this)
with (this.RESETBUTTON)
  onClick = class::RESETBUTTON_ONCLICK
  height = 1.0
  left = 11.0
  top = 8.08
  width = 15.3333
  text = "Reset Tables"
endwith
this.MANBUTTON = new PUSHBUTTON(this)
with (this.MANBUTTON)
  onClick = class::MANBUTTON_ONCLICK
  height = 1.0
  left = 11.0
  top = 9.32
  width = 15.3333
  text = "Manual Calc."
endwith
this.EXISTBUTTON = new PUSHBUTTON(this)
with (this.EXISTBUTTON)
  onClick = class::EXISTBUTTON_ONCLICK
  height = 1.0
  left = 30.0
  top = 8.08
  width = 15.3333
  text = "Calc. Existing TT"
endwith
this.CALCBUTTON = new PUSHBUTTON(this)
with (this.CALCBUTTON)
  onClick = class::CALCBUTTON_ONCLICK
  height = 1.0
  left = 30.0
  top = 8.08
  width = 15.3333
  text = "Calc. New TT"
endwith
Random Calc.

this.ROUTELABEL = new TEXTLABEL(this)
with (this.ROUTELABEL)
  height = 1.0
  left = 2.0
  top = 1.0
  width = 21.0
  text = "No. of I/O Routes"
endwith

this.RUNLABEL = new TEXTLABEL(this)
with (this.RUNLABEL)
  height = 1.0
  left = 21.0
  top = 1.0
  width = 11.0
  text = "No. of Loops"
endwith

this.CHROMLABEL = new TEXTLABEL(this)
with (this.CHROMLABEL)
  height = 1.0
  left = 35.0
  top = 1.0
  width = 18.0
  text = "No. of Chromosomes"
endwith

this.PREVIEWBUTTON = new PUSHBUTTON(this)
with (this.PREVIEWBUTTON)
onClick = class::PREVIEWBUTTON::ONCLICK
  height = 1.0
  left = 20.0
  top = 13.28
  width = 15.3333
  text = "View Results"
endwith

this.GABUTTON = new PUSHBUTTON(this)
with (this.GABUTTON)
onClick = class::GABUTTON::ONCLICK
  height = 1.0
  left = 21.0
  top = 10.6
  width = 15.0
  text = "GA"
endwith

this.EXISTRADIO = new RADIOBUTTON(this)
with (this.EXISTRADIO)
  height = 1.08
  left = 11.0
  top = 14.32
  width = 14.0
  text = "Existing"
group = true
  value = true
endwith

this.MANRAD = new RADIOBUTTON(this)
with (this.MANRAD)
  height = 1.08
  left = 34.0
  top = 14.32
  width = 14.0
  text = "Manual"
this.RANRADIO = new RADIOBUTTON(this)
with (this.RANRADIO)
    height = 1.08
    left = 11.0
    top = 15.4
    width = 14.0
    text = "Random"
endwith

this.GARADIO = new RADIOBUTTON(this)
with (this.GARADIO)
    height = 1.08
    left = 34.0
    top = 15.4
    width = 14.0
    text = "GA Results"
endwith

this.CONTAINER1 = new CONTAINER(this)
with (this.CONTAINER1)
    left = 55.0
    top = 0.68
    width = 19.0
    height = 14.0
endwith

this.CONTAINER1.LINE1 = new LINE(this.CONTAINER1)
with (this.CONTAINER1.LINE1)
    left = 0.0
    right = 18.5556
    top = 3.88
    bottom = 3.88
    width = 1
    pageno = 0
endwith

this.CONTAINER1.LINE2 = new LINE(this.CONTAINER1)
with (this.CONTAINER1.LINE2)
    left = 0.0
    right = 18.0
    top = 8.8
    bottom = 8.8
    width = 1
    pageno = 0
endwith

this.CONTAINER1.RADIOBUTTON1 = new RADIOBUTTON(this.CONTAINER1)
with (this.CONTAINER1.RADIOBUTTON1)
    height = 1.0909
    left = 1.5714
    top = 5.8636
    width = 15.7143
    text = "5-min. Interval"
    group = true
    value = true
    pageno = 0
endwith

this.CONTAINER1.RADIOBUTTON2 = new RADIOBUTTON(this.CONTAINER1)
with (this.CONTAINER1.RADIOBUTTON2)
    height = 1.0909
    left = 1.5714
    top = 7.3636
    width = 15.7143
    text = "1-min. Interval"
    pageno = 0
endwith

this.CONTAINER1.TEXTLABEL1 = new TEXTLABEL(this.CONTAINER1)
with (this.CONTAINER1.TEXTLABEL1)
text = "Const."
pageno = 0
endwith

this.CONTAINER1.RADIOBUTTON5 = new RADIOBUTTON(this.CONTAINER1)
with (this.CONTAINER1.RADIOBUTTON5)
  height = 1.0909
  left = 1.5714
  top = 12.3636
  width = 10.0
  text = "Random"
  pageno = 0
endwith

this.CONTAINER1.TEXTLABEL4 = new TEXTLABEL(this.CONTAINER1)
with (this.CONTAINER1.TEXTLABEL4)
  height = 1.0
  left = 3.0
  top = 9.3182
  width = 13.5714
  text = "Seed Values"
  fontBold = true
  pageno = 0
endwith

this.CONTAINER1.SPINBOX1 = new SPINBOX(this.CONTAINER1)
with (this.CONTAINER1.SPINBOX1)
  height = 1.0
  left = 9.5714
  top = 11.3636
  width = 8.0
  rangeMax = 100
  rangeMin = 1
  value = 1
  spinOnly = true
  rangeRequired = true
  pageno = 0
endwith

this.DEFAULT = new PUSHBUTTON(this)
with (this.DEFAULT)
  onClick = class::DEFAULT_ONCLICK
  height = 1.0909
  left = 57.0
  top = 15.5
  width = 15.2857
  text = "Default Values"
endwith

this.RECTANGLE4 = new RECTANGLE(this)
with (this.RECTANGLE4)
  left = 1.0
  top = 3.92
  width = 25.4444
  height = 2.2
  text = "Sched. Deviation"
  fontBold = true
  fontItalic = true
endwith

this.RADIOBUTTON1 = new RADIOBUTTON(this)
with (this.RADIOBUTTON1)
  height = 1.08
  left = 1.5556
  top = 4.84
  width = 8.5556
  text = "Average"
  group = true
  value = true
endwith

this.RADIOBUTTON2 = new RADIOBUTTON(this)
with (this.RADIOBUTTON2)
  height = 1.08
  left = 10.5
  top = 4.84
  width = 9.3333
  text = "Random"
endwith

this.RADIOBUTTON3 = new RADIOBUTTON(this)
with (this.RADIOBUTTON3)
  height = 1.08
  left = 19.4444
  top = 4.84
  width = 6.2222
  text = "As Is"
endwith

this.RECTANGLE5 = new RECTANGLE(this)
with (this.RECTANGLE5)
  left = 27.7778
  top = 3.92
  width = 12.4444
  height = 2.2
  text = "Crossover"
  fontBold = true
  fontItalic = true
endwith

this.CRSPIN = new SPINBOX(this)
with (this.CRSPIN)
  height = 1.0
  left = 28.5278
  top = 4.84
  width = 8.0
  rangeMax = 100
  rangeMin = 0
  value = 50
  spinOnly = true
  rangeRequired = true
endwith

this.TEXTLABEL2 = new TEXTLABEL(this)
with (this.TEXTLABEL2)
  height = 1.0
  left = 37.0
  top = 5.0
  width = 2.6667
  text = "%"
  fontBold = true
endwith

this.RECTANGLE6 = new RECTANGLE(this)
with (this.RECTANGLE6)
  left = 41.5556
  top = 3.88
  width = 12.4444
  height = 2.2
  text = "Mutation"
  fontBold = true
  fontItalic = true
endwith

this.MUSPIN = new SPINBOX(this)
with (this.MUSPIN)
  height = 1.0
  left = 42.3333
  top = 4.8
  width = 8.0
  rangeMax = 100
  rangeMin = 0
  value = 10
  spinOnly = true
rangeRequired = true
endwith
this.TEXTLABEL3 = new TEXTLABEL(this)
with (this.TEXTLABEL3)
    height = 1.0
    left = 50.8056
    top = 4.96
    width = 2.6667
    text = "%"
    fontBold = true
endwith
this.TEXTLABEL1 = new TEXTLABEL(this)
with (this.TEXTLABEL1)
    height = 18.5
    left = 0.0
    top = 0.0
    width = 77.0
    text = "Processing...
    colorNormal = "white/black"
    fontItalic = true
endwith

defunction CALCBUTTON_onClick
    rr = form.routespin.value
    loops = form.runspin.value
    variation = val(form.container1.rangecombobox1.value)
    time_st = form.container1.timespin.value/60
    if form.container1.radiobutton1.value = true
        int_shif = 5
        num_shif = ((2*variation)/5)+1 //Number of Shiftings
    else
        int_shif = 1
        num_shif = (2*variation)+1 //Number of Shiftings
    endif
    if form.container1.radiobutton3.value = true
        Rand(179757)
    elseif form.container1.radiobutton4.value = true
        Rand(179757+form.container1.spinbox1.value)
    else
        Rand(Int(1000000*Rand()))
    endif
    if counter = 0
        counter = 1
        do calc_Exist.prg // Existing conditions
    else
        do calc_random.prg // "Heuristic" calculation (random time shifts)
    endif
    return
endfunction

defunction DEFAULT_onClick
    form.routespin.value 80
    form.runspin.value = 20
    form.chromspin.value = 50
    form.container1.rangecombobox1.value '15'
    form.container1.timespin.value = 3
    form.radiobutton1.value = true
    form.container1.radiobutton2.value true
    form.existradio.value = true
    form.container1.radiobutton3.value true
    form.container1.spinbox1.value = 1
    form.crsspin.value 50
    form.muspin.value = 10
    return
endfunction

defunction EXISTBUTTON_onClick
    rr = form.routespin.value
    time_st = form.container1.timespin.value/60
    do calc_Exist.prg // Existing conditions

function form_onOpen
Public RR, SS, loops, counter, SchDev, crate, mrate
rr = form.routespin.value
form.container1.rangecombobox1.value = '15'
form.container1.radiobutton2.value = true
// do calc_exist.prg // Existing conditions
// form.textLabel1.visible = false
// form.visible = true

if file('Generation.dbf') or file('G1.dbf')
  if type("WinExec") # "FP"
    extern CINT WinExec(CSTRING, CINT) Kernel32
    run(false, "Del_G.bat")
  endif
  else
endif

form.textLabel1.visible = false
counter = 0
return

function GABUTTON_onClick
rr = form.routespin.value // number of directional routes
ss = form.chromspin.value // number of chromosomes
loops = form.runspin.value // number of generations
variation = val(form.container1.rangecombobox1.value) // e.g., 15 minutes (5, 10, 15, 30, etc.)
time_st = form.container1.timespin.value/60

// Crossover and Mutation Rates
crate = form.crspin.value
mrate = form.muspin.value

if form.container1.radiobutton1.value = true
  int shif 5
  num_shif = ((2*variation)/5)+1 // Number of Shiftings (7) - 15-10-5-0-(5)-(10)-(15)
else
  int shif 1
  num_shif = (2*variation)+1 // Number of Shiftings (31) - 15-14-13-12----0---(-12)-(-13)-(-14)-(-15)
endif
if form.container1.radiobutton3.value = true
  Rand(179757)
elseif form.container1.radiobutton4.value = true
  Rand(179757+form.container1.spinbox1.value)
else
  Rand(Int(100000*Rand()))
endif

//
  if counter = 0
  //
  counter = 1
  //
  do calc_Exist.prg // Existing conditions
  //
  else
  //
  endif

if form.radiobutton1.value = true
  SchDev = 1
elseif form.radiobutton2.value = true
  SchDev = 2
elseif form.radiobutton3.value = true
  SchDev = 3
endif

do Calc.ga.prg // GA
return

function MANBUTTON_onClick
rr = form.routespin.value
loops = form.runspin.value
time_st = form.container1.timespin.value/60
if counter = 0
    counter = 1
    do calc_Exist.prg // Existing conditions
else
endif

do calc_ttm.prg // Manual Calculation
return

function PREVIEWBUTTON_onClick
    do case
    case form.existradio.value = true
        run(true,'notepad curr tt.txt')
    case form.manradio.value = true
        run(true,'notepad man_gen.txt')
    case form.ranradio.value = true
        run(true,'notepad random.txt')
    case form.garadio.value = true
        run(true,'notepad calc ga.txt')
    otherwise
        run(true,'notepad calc ga.txt')
    endcase
    return

function RESETBUTTON_onClick
    run reset.bat
    return
endclass
calc_exist.prg

/*
Calc_Exist.prg

Author: Fabian Cevallos, August 2, 2004

Ph.D. Dissertation
Florida International University

Description:

This program is designed to calculate transfer times for an existing Mass Transit System, based on information provided from public timetables. This is part of a project that attempts a global optimization of transfer times.

This particular utility computes transfer time from one route to another for all available trips in a weekday period.

History:

12/17/2001 -- original code
02/02/2002 -- included ridership data
08/02/2004 -- improved code and solved memory problems

Usage:

To run this program, double click the TT.wfm form or the TT.exe. You need to have all the Timetables in dBase format (dbf).

The time format is xx.xx hours in military time (e.g., 13.34 h.).

!!!!!!!!!!!!IMPORTANT REMARK!!!!!!!!!!!!
The performance of this program depend largely on the size of the dataset as well as the computer performance. And even though, every attempt has been made to optimize computer performance, the intend of this program is to assist with the calculations of transfer time, not to develop a sophisticated computer program.

-----------------------------------------------

*/

local cTimeStart, cTimeEnd //Time variables

//rr = 80 //Transit Routes by direction

public aR_Matrix, Rid_dat, ExisTT

aR Matrix = new array (rr, rr) //Route Matrix - T_matrix.dbf
Rid_dat = new array (rr, rr) //Ridership Matrix - riders.dbf
aMove = new array (rr, rr) //Transfer Time for a particular movement

if file('curr tt.txt')
    delete file curr tt.txt
endif

set alternate to curr_tt //Current transfer times
set alternate on //Start Recording

cTimeStart = datetime() //Initial DateTime in fractions of a day

? " Transfer Times
Loops

Transfers

Totals"

?" --------------------- -----

; -----------------

-----" 
?"
set alternate off  //Stop Recording
set alternate on   //Start Recording

sum=0  //Initialize the sum variable
ExisTT = 0  //Initialize the variable that stores existing travel times
sum_move = 0  //Initialize the sum of movements
MoveTT = 0  //Initialize movement transfer time

//Read the T matrix transfer table and store values in the aR_Matrix[i, j] array
ql = new Query()
ql.sql = "select * from T_matrix" //Transfer Table
ql.active = true
r = ql.rowset
for i = 1 to rr
    for j = 1 to rr
        xx = "R" + j
        aR_Matrix[i, j] = r.fields[xx].value
    next j
next i

//Read the aR Matrix[ii, jj] array and store values in variables to determine routes and
time points from which transfers are made.
aa = 0  //Initialize aa

for ii = 1 to rr
    for jj = 1 to rr
        store_loc = 1  // Initialize variable to store j rows to be used in the TT calculation
        aAR = aR_Matrix[ii, jj]
        aa = left(trim(aAR) + "xxx", 4)
        if not (aa = 'xxxx' or aa = '0xxx')  //Disregard 0 and x values in the Matrix
            //aa , ii + ""," + jj
            bb = val(left (aa,2))  //column for Route i
            cc = val(right (aa,2))  //column for Route j
            // ?bb, cc, bb+cc
            dd = "r"+ii  //Route ii - Rte number in row of Transfer Table
            ee = dd + ".dbf"  //rii.dbf
            //?"dd =" + dd
            ff = "r"+jj  //Route jj - Rte number in column of Transfer Table
            gg = ff + ".dbf"  //rjj.dbf
ql.sql = "select * from " + dd + ""
ql.active = true

r = ql.rowset
n = r.count() //number of records
a = r.fields.size //number of fields

aRte_ii = new array(n, a) //Array that stores timetable values for Rte rii (n rows, a columns)

for i = 1 to n
  for j = 1 to a
    xx = "T" + bb
    aRte_ii[i, j] = r.fields[xx].value
  next j

r.next()
next i

ql.sql = "select * from " + ff + ""
ql.active = true

r = ql.rowset
m = r.count() //number of records in table test
a = r.fields.size //number of fields in table test

aRte_jj = new array(m, a) //Array that stores timetable values for Rte rjj (m rows, a columns)

for i = 1 to m
  for j = 1 to a
    xx = "T" + cc
    aRte_jj[i, j] = r.fields[xx].value
  next j

r.next()
next i

for f = bb to bb //Calculate Transfer Time TT from column bb (tranfer point for route i)
  for g = cc to cc //to column cc (tranfer point for route j)
    for i = 1 to n //for all rows i (trips in route i)
      for j = 1 to m //all rows j (trips in route j)
        //if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + 3/60 <=
        //aRte_jj[j, g] //If time values are not zero and
        if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + time_st <=
          aRte_jj[j, g] //If time values are not zero and

        //Rii <= Rjj // Transfer time TT calculation
        do while aRte_ii[i, f] + time_st <= aRte_jj[j, g]
          TT = (Rid_dat[ii, jj] * (aRte_jj[j, g] - aRte_ii[i, f])) //Transfer Times x riders from i to j
          sum = sum + TT //Cumulative Transfer Times
          sum_move = sum_move + TT
          ?" TT = " + TT + chr(9) + "hours" + chr(9) + "Arr.Time " + aRte_ii[i, f] +
          chr(9) + "Dep.Time " + aRte_jj[j, g] +;
          chr(9) + "[ii-jj,i,f,j,g] = " + "[" + "R" + ii + ":" + "R" + jj + ":" + "r" +
          i + ":" + "c" + f + ";"
          + "r" + j + ":" + "c" + g + "]" + chr(9) + " Total= " + sum + chr(9) +
          + "hours"

          if aRte_ii[i, f], i, f
          //aRte_ii[i, f], i, f
          //aRte_jj[j, g], j, g
if aRte_ii[j, g] >= aRte_ii[i, f]
    exit
else
    endif
else
    endif

next j
next i
next g
next f

aMove[ii, jj] = sum_move  //Transfer time from Rte rii to Rte rjj
Move_TT = aMove[ii, jj]

set alternate off  //Stop recording
set alternate on   //Start recording

?"aMove["+ii+","+ jj+"] =" + chr(9) + Move_TT + chr(9) + "Hours"  //Transfer Time form rii to rjj
?

sum_move = 0
else
endif

next jj
next ii

?"Existing Transfer Times = " + sum + " Hours"

ExisTT = sum

release object ql
release object r
ql = null
r = null

set alternate off  //Stop recording
set alternate on   //Start recording

CTimeEnd = datetime()  //End DateTime
//Comp_Time = elapsed(CTimeEnd, cTimeStart, 1)  //Elapsed computing time
Comp_Time = (CTimeEnd-cTimeStart)*24*3600
?
? "Computing Time = " + Comp_Time + " sec."
close alternate  //Close log file
calc_ttm.prg

/*
  Calc_ttm.prg

  Author:  Fabian Cevallos, August 6, 2004

  Ph.D. Dissertation
  Florida International University

  Description:

  This program is designed to calculate optimal transfer times, using random
time shifting and from a pre-defined number of "trials" of a Mass Transit
System, based on information from public timetables and ridership data.
This is part of a project that attempts a global optimization of transfer
times.

  This particular utility computes transfer times based on values
stored in the man_gen.dbf table.

  History:

  12/17/2001 -- original code
  02/02/2002 -- include ridership data
  04/27/2002 -- add seed value for random numbers
  07/14/2002 -- Miscellaneous improvements
  08/06/2004 -- improved code and solved memory problems
  07/23/2005 -- Minor text modifications

  Usage:

  To run this program, double click the TT.wfm form or the TT.exe. You need
to have all the Timetables in dBase format (dbf).
The time format is xx.xx hours in military time (e.g., 13.34 h.).

  !!!!!!!!!!!!!!!!!IMPORTANT REMARK!!!!!!!!!!!!!!
  The performance of this program depend largely on the size of the dataset as
well as the computer performance. And even though, every attempt has been
made to optimize computer performance, the intend of this program is to assist
with the calculations of transfer time, not to develop a sophisticated
computer program.

  */

local cTimeStart, cTimeEnd       //Time variables
public total, sum

set procedure to calc_Exist.prg addi

//rr = 80       //Transit Routes by direction

//aR_Matrix = new array (rr, rr)       //Route Matrix
//Rid_dat = new array (rr, rr)        //Ridership Matrix - riders.dbf
aGene = new array (rr)               //Gene Time Cost
aMove = new array (rr, rr)           //Transfer Time for a particular movement
Genes = new array (rr)               //Stored Genes with lowest Transfer Times

if file('man_gen.txt')
  delete file man_gen.txt
endif

set alternate to man_gen       //File with new transfer times
set alternate on               //Start Recording

cTimeStart = DateTime()       //Initial DateTime
total=0 //Initialize the total variable

//loops = 3
//for k = 1 to loops

sum=0 //Initialize the sum variable

//? "Transfer Times
//? Loops
//? Transfers

//?

//?

q1 = new Query()

q1.sql = "select * from man_genes" //Use man_genes Table
q1.active = true

r = q1.rowset

for i = 1 to rr/2
    aGene[i] = r.fields["genes"].value/60
    //?i++ "+aGene[i]
    r.next()
next i

for i = (rr/2)+1 to rr
    aGene[i] = aGene[i-(rr/2)]
    //?i++ "+aGene[i]
next i

sum=0 //Initialize the sum variable
sum_move = 0 //Initialize the sum of movements
Move_TT = 0 //Initialize movement transfer time

//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array
q1 = new Query()

q1.sql = "select * from T_matrix" //Transfer Table
q1.active = true

r = q1.rowset

for i = 1 to rr
    for j = 1 to rr
        xx = "R" + j
        aR_Matrix[i, j] = r.fields[xx].value
    next j
    r.next()
next i

//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array
q1 = new Query()

q1.sql = "select * from Riders" //Use Ridership Table
q1.active = true

r = q1.rowset

for i = 1 to rr
    for j = 1 to rr
        xx = "R" + j
\[
\text{Rid}_\text{dat}[i, j] = \text{r.fields}[xx].value
\]

\[\text{next } j\]

\[\text{r.next()}\]

\[\text{next } i\]

//Read the aR_Matrix[ii, jj] array and store values in variables to determine routes and
//time points from which transfers are made.
\[\text{aa} = 0\] //Initialize aa

for \[ii = 1\] to \[rr\]
for \[jj = 1\] to \[rr\]
    \[\text{store_loc} = 1\] // Initialize variable to store j rows to be used in the TT
calculation
\[\text{aaR} = \text{aR.Matrix}[ii, jj]\]
\[\text{aa} = \text{left}(\text{trim}(\text{aaR}) + "xxx", 4)\]

if not (\[\text{aa} = 'xxxx'\] or \[\text{aa} = '0xxx'\]) //Disregard 0 and x values in the Matrix
    \[\text{aa} , ii + "," + jj\]

\[\text{bb} = \text{val}(\text{left}(\text{aa},2))\] //column for Route i
\[\text{cc} = \text{val}(\text{right}(\text{aa},2))\] //column for Route j

\[?\text{bb, cc, bb+cc}\]
\[\text{dd} = "r"+\text{ii} //Route ii - Rte number in row of Transfer Table\]
\[\text{ee} = \text{dd} + ",.dbf" //rri.dbf\]
\[?\text{?dd = "} + \text{dd}\]

\[\text{ff} = "r"+\text{jj} //Route jj - Rte number in column of Transfer Table\]
\[\text{gg} = \text{ff} + ",.dbf" //rjj.dbf\]

\[\text{q1.sql = "select * from " + dd + ","}\
\[\text{q1.active = true}\]

\[\text{r = qu.rowset}\]
\[\text{n = r.count()} //number of records\]
\[\text{a = r.fields.size} //number of fields\]

\[\text{aRte_\text{ii}} = \text{new array}(\text{n, a}) //Array that stores timetable values for Rte rii (n rows, a columns)\]

\[\text{for } i = 1 \text{ to } n\]
\[\text{for } j = 1 \text{ to } a\]
\[\text{xx = "T" + bb}\]
\[\text{aRte_\text{ii}[i, j] = r.fields[xx].value}\]
\[\text{next } j\]

\[\text{r.next()}\]
\[\text{next } i\]

\[\text{q1.sql = "select * from " + ff + ","}\
\[\text{q1.active = true}\]

\[\text{r = qu.rowset}\]
\[\text{m = r.count()} //number of records in table test\]
\[\text{a = r.fields.size} //number of fields in table test\]

\[\text{aRte_\text{jj}} = \text{new array}(\text{m, a}) //Array that stores timetable values for Rte rjj (m rows, a columns)\]

\[\text{for } i = 1 \text{ to } m\]
\[\text{for } j = 1 \text{ to } a\]
\[\text{xx = "T" + cc}\]
\[\text{aRte_\text{jj}[i, j] = r.fields[xx].value}\]
\[\text{next } j\]

\[\text{r.next()}\]
next i

for f = bb to bb //Calculate Transfer Time TT from column bb
for g = cc to cc //to column cc
for i = 1 to n //for all rows i to
for j = 1 to m //all rows j

    if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + time_st <=
aRte_jj[j, g] //If time values are not zero and

    //Rii <= Rjj
    // Transfer time TT calculation

    //do while aRte_ii[i, f] <= aRte_jj[j, g]
do while (aRte_ii[i, f] + aGene[ii]) + time_st <= (aRte_jj[j, g] + aGene[jj])

    TT = (Rid dat[ii, jj] * ((aRte_jj[j, g] + aGene[jj]) - (aRte_ii[i, f] + aGene[ii]))) //Transfer Times

    sum = sum + TT //Cumulative Transfer Times
    sum_move = sum_move + TT

    //"TT = " + TT + chr(9) + "hours" + chr(9) + "Arr.Time " + aRte_ii[i, f] +
    chr(9) + "Dep.Time " + aRte_jj[j, g] +
    chr(9) + "[ii-j, i, f, j, g] = " + "[" + "R" + ii + "-" + "R" + jj + " +" + "r" + i + "," + "c" + f +;
    "," + "r" + j + "," + "c" + g + "] " + chr(9) + " Total= " + sum + chr(9) +
    "hours"

    //?aRte_ii[i, f], i, f
    //?aRte_jj[j, g], j, g

    exit
    endif

endif

next j
next i
next g
next f

aMove[ii, jj] = sum_move //Transfer time from Rte rii to Rte rjj
Move_TT = aMove[ii, jj]

//?
//?"aMove["+ii+","+ jj+"] = " + Move_TT + " hours" //Transfer Time from rii to rjj
//?

sum_move = 0
else
endif

next jj
next ii

if total = 0

    total = sum

    for l = 1 to rr/2

        Genes[l] = aGene[l]

        //?"Genes (" + l + ") = " + Genes[l] + " h."

    next l
elseif total > sum
    total = sum
    for l = 1 to rr/2
        Genes[l] = aGene[l]
        //?"Genes (" + l + ") = " + Genes[l] + " h."
    next l
    else
endif

  //set alternate off   //Stop recording
  //set alternate on    //Start recording

  //next loops

  "**** Solution ****"
  "Exist. Total TT = " + ExisTT + " Hours"
  "Calc. Total TT = " + total + " Hours"
  //?"Number of Trials = " + loops
  "Savings = " + (ExisTT - total) + " Hours" + " (" + (100*((ExisTT - total)/ExisTT)) + ")%"
  "Time Shifts"
  "Genes (Minutes)"
  "----- ---------"

  for l = 1 to rr/2
    "Genes (" + l + ") = " + chr(9) + round(Genes[l]*60,0)
  next l

  CTimeEnd = DateTime() //End DateTime
  //Comp_Time = elapsed(CTimeEnd, cTimeStart, 1) //Elapsed computing time
  Comp_Time = (CTimeEnd-cTimeStart)*24*3600
  "Computing Time = " + Comp_Time + " sec."
  close alternate  //Close man_gen.txt
calc_random.prg
/*
Calc_random.prg

Author: Fabian Cavalllos, August 6, 2004

Ph.D. Dissertation
Florida International University

Description:
This program is designed to calculate optimal transfer times, using random time shifting and from a pre-defined number of "trials" of a Mass Transit System, based on information from public timetables and ridership data. This is part of a project that attempts a global optimization of transfer times.

This particular utility computes transfer time from one route to another for all available trips in a weekday period.

History:
12/17/2001 -- original code
02/02/2002 -- include ridership data
04/27/2002 -- add seed value for random numbers
07/14/2002 -- Miscellaneous improvements
08/06/2004 -- improved code and solved memory problems

Usage:
To run this program, double click the TT.wfm form or the TT.exe. You need to have all the Timetables in dBase format (dbf). The time format is xx.xx hours in military time (e.g., 13.34 h.).

!!!!!!!!!!!!!!!!IMPORTANT REMARK!!!!!!!!!!!!
The performance of this program depend largely on the size of the dataset as well as the computer performance. And even though, every attempt has been made to optimize computer performance, the intend of this program is to assist with the calculations of transfer time, not to develop a sophisticated computer program.

*)
local cTimeStart, cTimeEnd //Time variables
public total, sum, final_TT

set procedure to calc_Exist.prg addi

//rr = 80 //Transit Routes by direction

//aR_Matrix = new array (rr, rr) //Route Matrix
//Rid_dat = new array (rr, rr) //Ridership Matrix - riders.dbf
aGene = new array (rr) //Gene Time Cost
aMove = new array (rr, rr) //Transfer Time for a particular movement
Genes = new array (rr) //Stored Genes with lowest Transfer Times

if file('man genes.dbf')
drop table man_genes
endif

create table man_genes (id autoinc, genes numeric (4, 0))

use man_genes
generate(rr/2)

use

if file('random.txt')
delete file random.txt
endif
set alternate to Random //File with new transfer times
set alternate on //Start Recording
cTimeStart = DateTime() //Initial DateTime
total=0 //Initialize the total variable
//loops = 3
for k = 1 to loops
sum=0 //Initialize the sum variable
//?
//? "Transfer Times
<table>
<thead>
<tr>
<th>Loops</th>
<th>Transfers</th>
</tr>
</thead>
</table>
//? " Totals"
|-------|----------|
for i = 1 to rr //Gene Values from 1 to rr (all routes both directions)
if i <= rr/2
rnd = int((num_shif)*rand()) //Random Number generator
for n = 0 to (num_shif-1)
do case
case rnd = n
aGene[i] = (variation-(n*int_shif))/60
//?aGene[i] endcase
next n
//?"aGene (" + i + ") = " + aGene[i] + "h."
else
aGene[i] = aGene[i-(rr/2)]
//?"aGene (" + i + ") = " + aGene[i] + "h."
endif
next i
//for i = 1 to rr //Gene Values from 1 to rr (all routes both directions)
//if i <= rr/2
//
// rnd = int(7*rand()) //Random Number generator
//
// do case
// case rnd = 0
// aGene[i] = 15/60
// case rnd = 1
// aGene[i] = 10/60
// case rnd = 2
// aGene[i] = 5/60
// case rnd = 3
// aGene[i] = 0/60
// case rnd = 4
aGene[i] = -5/60

// case rnd = 5
// aGene[i] = -10/60
// case rnd = 6
// aGene[i] = -15/60
// endcase
// */aGene (" + i + ") = " + aGene[i] + "."
else
/*
 aGene[i] = aGene[i-(rr/2)]
// */aGene (" + i + ") = " + aGene[i] + "."
// endif

// next i

sum = 0 // Initialize the sum variable
sum_move = 0 // Initialize the sum of movements
Move_TT = 0 // Initialize movement transfer time

// Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array
ql = new Query()
ql.sql = "select * from T_matrix" // Transfer Table
ql.active = true
r = ql.rowset

for i = 1 to rr
    for j = 1 to rr
        xx = "R" + j
        aR_Matrix[i, j] = r.fields[xx].value
    next j
next i

// Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array
ql = new Query()
ql.sql = "select * from Riders" // Use Ridership Table
ql.active = true
r = ql.rowset

for i = 1 to rr
    for j = 1 to rr
        xx = "R" + j
        Rid_dat[i, j] = r.fields[xx].value
    next j
next i

// Read the aR_Matrix[ii, jj] array and store values in variables to determine routes and
// time points from which transfers are made.
aa = 0 // Initialize aa

for ii = 1 to rr
    for jj = 1 to rr
        store_loc = 1 // Initialize variable to store j rows to be used in the TT calculation
        aaR = aR_Matrix[ii, jj]
        aa = left(trim(aaR) + "xxx", 4)
        if not (aa = 'xxxx' or aa = '0xxx') // Disregard 0 and x values in the Matrix
            // aa, ii + "," + jj

// 127
bb = val(left (aa,2)) //column for Route i
cc = val(right (aa,2)) //column for Route j
//?bb, cc, bb+cc

dd = "r"+ii //Route ii - Rte number in row of Transfer Table
ee = dd + ".dbf" //rri.dbf
//?"dd =" + dd

ff = "r"+jj //Route jj - Rte number in column of Transfer Table
gg = ff + ".dbf" //rjj.dbf

ql.sql = "select * from " + dd + ""
ql.active = true

r = ql.rowset
n = r.count() //number of records
a = r.fields.size //number of fields

aRte_ii = new array(n, a) //Array that stores timetable values for Rte rii (n rows, a columns)

for i = 1 to n
  for j = 1 to a
    xx = "T" + bb
    aRte_ii[i, j] = r.fields[xx].value
  next j

r.next()
next i

ql.sql = "select * from " + ff + ""
ql.active = true

r = ql.rowset
m = r.count() //number of records in table test
a = r.fields.size //number of fields in table test

aRte_jj = new array(m, a) //Array that stores timetable values for Rte rjj (m rows, a columns)

for i = 1 to m
  for j = 1 to a
    xx = "T" + cc
    aRte_jj[i, j] = r.fields[xx].value
  next j

r.next()
next i

for f = bb to bb //Calculate Transfer Time TT from column bb
for g = cc to cc //to column cc
  for i = 1 to n //for all rows i to
    for j = 1 to m //all rows j
      if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + time_st <= aRte_jj[j, g] //If time values are not zero and

//Rii <= Rjj
// Transfer time TT calculation

//do while aRte_ii[i, f] <= aRte_jj[j, g]
do while (aRte_ii[i, f] + aGene[ii]) + time_st <= (aRte_jj[j, g] + aGene[jj])

TT = (Rid_dat[ii, jj] * ((aRte_jj[j, g] + aGene[jj]) - (aRte_ii[i, f] + aGene[ii]))) //Transfer Times

sum = sum + TT
sum_move = sum_move + TT //Cumulative Transfer Times

128
TT = " + TT + chr(9) + "hours" + chr(9) + "Arr.Time " + aRte_ii[i, f] + chr(9) + "Dep.Time " + aRte_jj[j, g] + ;
chr(9) + "[ii-]j,i,f,j,g] = " + "[" + "R" + ii + "]" + "R" + jj + "r" + i + "]" + "c" + f + ;
" + "r" + j + "])" + "c" + g + "]" + chr(9) + "Total= " + sum + chr(9) + "hours"

//?aRte_ii[i, f], i, f
//?aRte_ii[j, g], j, g
exit
enddo

if aRte_jj[j, g] >= aRte_ii[i, f]
exit
else
endif
else
endif

next j
next i
next g
next f

aMove[ii, jj] = sum_move //Transfer time from Rte rii to Rte rjj
Move_TT = aMove[ii, jj]

//?
//?"aMove["+ii+","+ jj+"] = " + Move_TT + " hours" //Transfer Time from rii to rjj
//?

sum_move = 0
else
endif

next jj
next ii

if total = 0
  total = sum
  for l = 1 to rr/2
    Genes[l] = aGene[l]
    //?"Genes (" + l + ") = " + Genes[l] + " h."
    next l
  elseif total > sum
    total = sum
    for l = 1 to rr/2
      Genes[l] = aGene[l]
      //?"Genes (" + l + ") = " + Genes[l] + " h."
      next l
    else
      endif

//set alternate off //Stop recording
//set alternate on //Start recording
next loops

?"**** Solution ****"
? "Exist. Total TT = " + ExisTT + " Hours"
? "Calc. Total TT = " + total + " Hours"
? "Number of Trials = " + loops
?
? "Savings = " + ((ExisTT - total) + " Hours" + " 
+ (100*((ExisTT - total)/ExisTT)) + " %")"
?

q = new Query()

q.sql = "select * from man_genes"
q.active = true

x = q.rowset

do while not x.endOfSet
for l = 1 to rr/2
  //?"Genes (" + l + ") = " + Genes[l] + " h.", round(Genes[l]*60,0)
?"Genes (" + l + ") = " + chr(9) + round(Genes[l]*60,0)

r.fields["genes"].value = round(Genes[l]*60,0)

r.next()
next l
enddo

CTimeEnd = DateTime()  //End DateTime
//Comp_Time = elapsed(CTimeEnd, cTimeStart, 1)  //Elapsed comuting time
Comp_Time = (CTimeEnd-cTimeStart)*24*3600
?
? "Computing Time = " + Comp_Time + " sec."
close alternate  //Close random.txt
calc_ga.prg

/*
Calc_ga.prg

Author: Fabian Cevallos, November 14, 2006

Ph.D. Dissertation
Florida International University

Description:

This program is designed to calculate optimal transfer times of a Mass Transit System, based on information from public timetables and ridership data. This is part of a project that attempts a global optimization of transfer times.

This particular utility uses the genetic algorithm to compute transfer times from one route to another for all available trips in a weekday period.

History:

12/17/2001 -- Original code
02/02/2002 -- Included ridership data
04/27/2002 -- Added seed value for random numbers
07/14/2002 -- Miscellaneous improvements
08/06/2004 -- Improved code and solved memory problems
09/01/2004 -- Started the development of the GA Crossover and Mutation Operators
09/01/2004 -- Implemented Final and Man Genes Tables to store Generations and Chromosomes
05/25/2005 -- Implemented Random Bus Arrivals, using OLE Automation to access the Excel LOGINV function
05/26/2005 -- Added Functionality for Random Bus Arrivals (Average, Random, As Is)
05/27/2005 -- Implemented the Crossover and Mutation Operators
05/27/2005 -- Added Functionality to allow for Crossover and Mutation Rates
05/28/2005 -- Added Functionality to allow some routes to remain fixed
07/23/2005 -- Fixed incorrectly creation of Generation tables
01/13/2006 -- Added Functionality to capture the results at different stages
11/14/2006 -- Added code to allow both Random and Manual Initial Solution

Usage:

To run this program, double click the TT.wfm form or the TT.exe. You need to have all the Timetables in dBase format (dbf).
The time format is xx.xx hours in military time (e.g., 13.34 h.).

!!!!!!!!!!IMPORTANT REMARK!!!!!!!!!!
The performance of this program depend largely on the size of the dataset, the set parameters, as well as the computer performance. And even though, every attempt has been made to optimize computer performance, the intent of this program is to assist with the calculations of transfer times, not to develop a sophisticated computer program.

---------------------------------------------------------------------------

* /

local cTimeStart, cTimeEnd //Time variables

cTimeStart = DateTime() //Initial DateTime

public total, sum, ExisTT, rvalue

//set procedure to calc_Exist.prg addi

//rr = 80 //Transit Routes by direction

aR Matrix = new array (rr, rr) //Route Matrix
Rid_dat = new array (rr, rr) //Ridership Matrix - riders.dbf
aGene = new array (rr) //Gene Time Cost
aMove = new array (rr, rr) //Transfer Time for a particular movement
Genes = new array (loops, rr) //Stored Genes with lowest Transfer Times

if file('man genes.dbf')
    drop table man_genes
endif

create table man_genes (id autoinc, genes numeric (4, 0)) // Table to store chromosomes of the final generation

use man_genes

generate(rr/2)
use

for i = 1 to ss //number of chromosomes
    new File().delete("G"+i".dbf")
    new File().delete("G_temp"+i".dbf")
    new File().delete("Generation"+i".dbf")
    new File().delete("Generation.dbf")
    new File().delete("Generationbk.dbf")
next i

//if file('Generation.dbf') or file('Generationbk.dbf')
//    drop table Generation
//    drop table Generationbk
//endif

create table Generation (;
    ID Autoinc,;
    TT Numeric (16,4),;
    CPU Numeric (16,4))

For i = 1 to (rr/2) //1 autoinc, 1 TT, and rr/2 gene fields - 1 chromosome per row
    fNewField = "G"+i
    ALTER TABLE GENERATION ADD &fNewField Numeric(4, 0)
Next i

use generation

generate(ss) //Generate "dummy" chromosomes
use

if file('Final.dbf')
    drop table Final
endif

create table Final (;
    ID Autoinc,;
    Generation CHAR(10),;
    TT Numeric (16,4),;
    CPU Numeric (16,4))

use Final

generate(loops) //Generate "dummy" Generations
use

if file('calc ga.txt')
    delete file calc ga.txt
endif

if file('final.txt')
    delete file final.txt
endif

set alternate to Calc ga //GA File
set alternate on     //Start Recording

//cTimeStart = DateTime() //Initial DateTime
//For k = 1 to loops  //First loop creates the first generation
dc = new Database()
dc.databaseName = "Transfers"
dc.active = true
qc = new Query() //add chromosomes to generation table
qc.database = dc
qc.sql = "select * from generation"
qc.active = true
rc = qc.rowset

total=0 //Initialize the total variable

For ch = 1 to ss

//?"ch ="+ch

sum=0 //Initialize the sum variable

//?

//? "Transfer Times

// Loops

// Transfers

// Totals"

//? "------------------------

//? "

//?"

if ch = 1

for i = 1 to rr  //Gene Values from 1 to rr (all routes both directions)

aGene[i] = 0 //Existing Conditions (no shifting)

next i

else

// Original Settings

for i = 1 to rr  //Gene Values from 1 to rr (all routes both directions)

if i <= rr/2

//Uncomment the following code to keep selected routes fixed

// if i = 2 or i = 10 //fix routes - no shifting

// aGene[i] = 0

//

// else

// rnd = int((num_shif)*rand()) //Random Number generator

for n = 0 to (num_shif-1) //Random Number generator

do case

case rnd = n

aGene[i] = (variation-(n*int_shif))/60  //All possible shiftings
(negative and positive)

//?aGene[i]

endcase

next n

//

endif

//?"aGene (" + i + ") = " + aGene[i] + "."

else

aGene[i] = aGene[i-(rr/2)]

//?"aGene (" + i + ") = " + aGene[i] + "."

endif

next i
endif
for i = 1 to rr //Gene Values from 1 to rr (all routes both directions)
if i <= rr/2
   aGene[i] = aGene[i-(rr/2)]
   //"aGene (" + i + ") = " + aGene[i] + "."
endif
next i
endif

//int_shif = 1 //Interval of Shiftings
//variation = 15 //Range of variation, e.g., +/- 15 minutes
//num_shif = (2*variation)+1 //Number of Shiftings
//time_st = 5/60 //Time to a transferring stop

//for i = 1 to rr //Gene Values from 1 to rr (all routes both directions)
//if i <= rr/2
   // rnd = int((num_shif)*rand()) //Random Number generator
   // for n = 0 to (num_shif-1)
```cpp
// do case
// case rnd = n
//   aGene[i] = (variation-(n*int_shift))/60
// endcase
// next n
// //"aGene (" + i + ") = " + aGene[i] + "."
// //"aGene (" + i + ") = " + round(aGene[i]*60,0)
// else
//   aGene[i] = aGene[i-(rr/2)]
// //"aGene (" + i + ") = " + aGene[i] + "."
// endif
// //next i

//for i = 1 to rr //Gene Values from 1 to rr (all routes both directions)
//if i <= rr/2
// do case
//   case rnd = 0
//     aGene[i] = 15/60
//   case rnd = 1
//     aGene[i] = 10/60
//   case rnd = 2
//     aGene[i] = 5/60
//   case rnd = 3
//     aGene[i] = 0/60
//   case rnd = 4
//     aGene[i] = -5/60
//   case rnd = 5
//     aGene[i] = -10/60
//   case rnd = 6
//     aGene[i] = -15/60
// endcase
// //"aGene (" + i + ") = " + aGene[i] + "."
// else
//   aGene[i] = aGene[i-(rr/2)]
// //"aGene (" + i + ") = " + aGene[i] + "."
// endif
// //next i

sum=0 //Initialize the sum variable
sum_move = 0 //Initialize the sum of movements
Move_TT = 0 //Initialize movement transfer time

//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array

d1 = new Database()
d1.databaseName = "Transfers"
d1.active = true
q1 = new Query()
q1.database = d1
q1.sql = "select * from T_matrix" //Transfer Table
q1.active = true
r = q1.rowset
for i = 1 to rr
  for j = 1 to rr
    ttx = "R" + j
    aR_Matrix[i, j] = r.fields[ttx].value
  next j
next i
```

135
//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array

d1 = new Database()
d1.databaseName = "Transfers"
d1.active = true
q1 = new Query()
q1 = new Query()
q1.database = d1
q1.sql = "select * from Riders" //Use Ridership Table
q1.active = true

r = q1.rowset
for i = 1 to rr
    for j = 1 to rr
        rtx = "R" + j
        Rid_dat[i, j] = r.fields[rtx].value
    next j
    r.next()
next i

//Read the aR_Matrix[ii, jj] array and store values in variables to determine routes and //time points from which transfers are made.
aa = 0 //Initialize aa
for ii = 1 to rr
    for jj = 1 to rr
        store_loc = 1 // Initialize variable to store j rows to be used in the TT calculation
        aaR = aR_Matrix[ii, jj]
        aa = left(trim(aaR) + "xxx", 4)
        if not (aa = 'xxxx' or aa = '0xxx') //Disregard 0 and x values in the Matrix
            //aa , ii + "," + jj
            bb = val(left(aa, 2)) //column for Route i
            cc = val(right(aa, 2)) //column for Route j
            //?bb, cc, bb+cc
            dd = "r"+ii //Route ii - Rte number in row of Transfer Table
            ee = dd + ".dbf" //rii.dbf
            //?\"dd\"=" + dd
            ff = "r"+jj //Route jj - Rte number in column of Transfer Table
gg = ff + ".dbf" //rjj.dbf
            q1.sql = "select * from " + dd + "" //select * from rii
            q1.active = true
            r = q1.rowset
            n = r.count() //number of records
            a = r.fields.size //number of fields

        aRte_ii = new array(n, a) //Array that stores timetable values for Rte rii (n rows, a columns)
        for i = 1 to n
            for j = 1 to a
                ti = "T" + bb //timepoint i of Rii
                aRte_ii[i, j] = r.fields[ti].value
            next j
r.next()

next i

ql.sql = "select * from " + ff + "" //select * from rjj
ql.active = true
r = ql.rowset
m = r.count() //number of records in table test
a = r.fields.size //number of fields in table test

aRte_jj = new array(m, a) //Array that stores timetable values for Rte rjj (m rows, a columns)

for i = 1 to m
  for j = 1 to a
    tj = "T" + cc //timepoint j of Rjj
    aRte_jj[i, j] = r.fields[tj].value
  next j
r.next()
next i

for f = bb to bb //Calculate Transfer Time TT from column bb
  for g = cc to cc //to column cc
    for i = 1 to n //for all rows i to
      for j = 1 to m //all rows j
        if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + time_st <= aRte_jj[j, g] //If time values are not zero and
          //Rii <= Rjj // Transfer time TT calculation
            do while aRte_ii[i, f] <= aRte_jj[j, g]
              do while (aRte_ii[i, f] + aGene[ii]) + time_st <= (aRte_jj[j, g] + aGene[jj])
                TT = (Rid_dat[ii, jj] * (aRte_jj[j, g] + aGene[jj]) - (aRte_ii[i, f] + aGene[ii])) //Transfer Times
                sum = sum + TT //Cumulative Transfer Times for all transfer combinations
                sum_move = sum_move + TT
                //?" TT = " + TT + chr(9) + "hours" + chr(9) + "Arr.Time " + aRte_ii[i, f] + chr(9) + "Dep.Time " + aRte_jj[j, g] + "r" + i + "," + "f" + i + "c" + f + "j" + "g" + "]" + chr(9) + "Total= "+ sum + chr(9) + "hours"
            endif
            endif
            next j
          endif
          next i
        endif
        next g
      next f
    next j
  next i
aMove[ii, jj] = sum_move //Transfer time from Rte rii to Rte rjj
Move_TT = aMove[ii, jj]

//?
//?"aMove["+ii+","+ jj+"] = " + Move_TT + " hours" //Transfer Time form rii to rjj
//?

sum_move = 0
else
endif

next jj
next ii

if ch = 1
    ExisTT = sum
else
endif

rc.fields["TT"].value = sum //Transfer Time
rc.fields["CPU"].value = (DateTime() - cTimeStart) * 24 * 3600 //CPU Time
//?"sum ="+sum
//?
//?"Row " + rc.rowno() + ", Value = " + rc.fields["TT"].value
//?

for j = 1 to rr/2
    chrom = "G" + j
    rc.fields[chrom].value = round(aGene[j] * 60, 0)
next j

rc.next()

if total = 0
    total = sum
    for l = 1 to rr/2
        Genes[l] = aGene[l]
        //?"Genes (" + l + ") = " + Genes[l] + " h."
    next l
elseif total > sum
    total = sum
    for l = 1 to rr/2
        Genes[l] = aGene[l]
        //?"Genes (" + l + ") = " + Genes[l] + " h."
    next l
else
endif

//
//set alternate off  //Stop recording
//set alternate on   //Start recording

next ch

dg = new Database() // //create one generation per loop
dg.databaseName = "Transfers"
dg.active = true
qg = new Query()
qg.database = dg
qg.sql = "select * from Generation order by TT ASC" //Sort by TT - Keep
    //the best chromosomes on top
qg.active = true
ug = new UpdateSet()
ug.source = qg.rowset
ug.destination = "Generationl.DBF" //Generation table
ug.copy()

Alter Table Generationl Add Dummy Blob
Alter Table Generationl drop id
Alter Table Generationl Add id autoinc
Alter Table Generationl Drop Dummy

 Appalachias[ 1 ].copyTable("Generation", "GenerationBk")
 Appalachias[ 1 ].copyTable("Generation1", "Gl")
 Appalachias[ 1 ].copyTable("Generation1", "G_temp1")

 ds = new Database()
ds.databaseName = "Transfers"
ds.active = true
s = new Query()
s.database = ds
s.sql = "select * from Final" //Use Final Table
s.active = true

t = s.rowset

For i = 1 to 1
    dgen = new Database()
dgen.databaseName = "Transfers"
dgen.active = true
qgen = new Query()
qgen.database = dgen
genertable = "G"+i
qgen.sql = "select * from &genertable"

qgen.active = true
rgen = qgen.rowset

t.fields["TT"].value = rgen.fields["TT"].value
t.fields["CPU"].value = rgen.fields["CPU"].value
t.fields["Generation"].value = genertable

t.next()

Next i

for l = 1 to rr/2
    Genes[l, 1] = 0
next l

 Appalachias[ 1 ].copyTable("G_temp1", "Temp") //Include Children and Parents
use "G_temp1"
Append from Temp
use

final TT = total

/* GA Operators Crossover and Mutation */
//---------------------------------------------------------------------------------------------

local va, vb, vc, vd, oo, pp, mu

For k = 2 to loops //for all Generations

dcc = new Database() //create one generation per loop
dcc.databaseName = "Transfers"
dcc.active = true
qcc = new Query() //crossover calculation
qcc.database = dcc

gtablek = "G_temp"+(k-1) //Generation table from previous loop
qcc.sql = [select * from &gtablek] //Transfer Table
qcc.active = true
rcc = qcc.rowset

//\"gtablek =\"+gtablek+" G_temp+(k-1)\"

//Crossover Operator

//Rand(179757) initialize pseudo random number
if crate = 0 //No Crossover
else
  for i = 1 to ss step 2//Pairs of Chromosomes
    for j = 1 to rr/2 //Genes
      gValue = "G" + j

      //Uncomment the following code to keep selected routes fixed
      // if gValue = "G2" or gValue = "G10" //fix routes - no shifting
      //   rcc.fields[gValue].value = 0
      //   rcc.next()
      //   rcc.fields[gValue].value = 0
      //   rcc.next(-1)
      // else
      //   Rand(179757 + i + j)
      coin = Rand()
      if coin <= (crate/100)
        x1 = rcc.fields[gValue].value //gene 1
        rcc.next()
        x2 = rcc.fields[gValue].value //gene 2
        rcc.fields[gValue].value = x1
        rcc.next(-1)
        rcc.fields[gValue].value = x2
      else
        endif
    //   endif
  // next j
rcc.next(2)
next i
endif

//Mutation Operator
if mrate = 0
else
rcc.first() //
    for i = 1 to (ss)
        for j = 1 to rr/2 //Genes
            gValue = "G" + j

            //Uncomment the following code to keep selected routes fixed
            //
            // if gValue = "G2" or gValue = "G10" //fix routes 2 and 10 - no shifting
            //
            // rcc.fields[gValue].value = 0
            //
            // else
            //
            Rand(179757 + i + j)
            coin = Rand()
            if coin <= (mrate/100)
                mu = "G" + j
                rnd = int((num_shif)*rand()) //Random Number generator
                for n = 0 to (num_shif-1)
                    do case
                        case rnd = n
                            rcc.fields[mu].value = variation-(n*int_shif) // New mutated gene value
                        //?
                            rcc.fields[mu].value = rnd // New mutated gene value
                            endcase
                    next n
                    else
                    endcase
            //
        endif
    next j
rcc.next()
next i
endif

do = new Database()
do.databaseName = "Transfers"
do.active = true
qo = new Query()
qo.database = do
qo.sql = "select * from &gtablek" //Use Generation Table
qo.active = true
ro = qo.rowset
total=0 //Initialize the total variable

//?"gtablek ="+gtablek" Generation+(k-1)"

//Random Arrivals
if SchDev = 1
    rValue = 0 // Average Deviation in minutes
elseif SchDev = 2
    oExcel = new oleAutoclient("Excel.Application")
oExcel.visible = .f.
oExcel.Workbooks.Add()
elseif SchDev = 3
    rValue = 0
endif

For ch = 1 to ss
    //?"ch ="+ch
    for i = 1 to (rr/2)
        fG = "G" + i
        aGene[i] = ro.fields[fG].value/60
        //?i+" +aGene[i]
    next i
    for i = (rr/2)+1 to rr
        aGene[i] = aGene[i-(rr/2)]
        //?i+" +aGene[i]
    next i

sum=0 //Initialize the sum variable

//? "Transfer Times
Loops
Transfers
 Totals"
//?" " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " " 

sum=0 //Initialize the sum variable

sum_move = 0 //Initialize the sum of movements
Move_TT = 0 //Initialize movement transfer time

//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array

d1 = new Database()
d1.databaseName = "Transfers"
d1.active = true
q1 = new Query()
q1.database = d1
q1.sql = "select * from T_matrix" //Transfer Table
q1.active = true

r = q1.rowset

for i = 1 to rr
    for j = 1 to rr
        ttx = "R" + j
        aR_Matrix[i, j] = r.fields[ttx].value
    next j
    r.next()
next i

//Read the T_matrix transfer table and store values in the aR_Matrix[i, j] array
d1 = new Database()
d1.databaseName = "Transfers"
d1.active = true
q1 = new Query()
q1.database = d1

q1.sql = "select * from Riders"  //Use Ridership Table
q1.active = true

r = q1.rowset
for i = 1 to rr
    for j = 1 to rr
        Rtx.dat[i, j] = r.fields[Rtx].value
    r.next()
next i

//Read the aR_Matrix[ii, jj] array and store values in variables to determine routes and
//time points from which transfers are made.
for ii = 1 to rr
    for jj = 1 to rr
        store_loc = 1  // Initialize variable to store j rows to be used in the TT
calculation
        aaR = aR_Matrix[ii, jj]
        aa = left(trim(aaR) + "xxx", 4)
        if not (aa = 'xxxx' or aa = 'Oxxx')  //Disregard 0 and x values in the Matrix
            //aa , ii + "," + jj
            bb = val(left (aa,2))  //column for Route i
            cc = val(right (aa,2))  //column for Route j
            ?bb, cc, bb+cc
            dd = "r"+ii  //Route ii - Rte number in row of Transfer Table
            ee = dd + ".dbf"  //rii.dbf
            //?dd =" + dd
            ff = "r"+jj  //Route jj - Rte number in column of Transfer Table
            gg = ff + ".dbf"  //rjj.dbf

            q1.sql = "select * from " + dd + ""
            q1.active = true
            r = q1.rowset
            n = r.count() //number of records
            a = r.fields.size //number of fields

            aRte_ii = new array(n, a) //Array that stores timetable values for Rte rii (n rows, a
            columns)
            for i = 1 to n
                for j = 1 to a
                    ti = "T" + bb  //timepoint i of Rii
                    //Random Arrivals
                    if SchDev = 2
                        oExcel.range("A1").select()
                        oExcel.ActiveCell.FormulaR1C1 = "=LOGINV(RAND(), 2.900519 ,0.272637)"
                        rValue = oExcel.Range("A1").Value
                        if rValue <= (13.70758) and rValue >= (24.12115)
 // Limit tail values smaller than 15th percentile and greater than 85th percentile  
  
  rValue = 0  
  else  
   rValue = rValue - 20  
  endif  
  else  
   endif  
  if r.fields[ti].value = 0  
   aRte_ii[i, j] = r.fields[ti].value  
  else  
   aRte_ii[i, j] = r.fields[ti].value + (rValue/60)  
     ?"aRte_ii[i, j] = "+i+", "+j+" "+(r.fields[ti].value + (rValue/60))  
  endif  
  next j  
  r.next()  
  next i  
  
  ql.sql = "select * from " + ff + ""  
  q1.active = true  
  r = q1.rowset  
  m = r.count() //number of records in table test  
  a = r.fields.size //number of fields in table test  
  aRte_jj = new array(m, a) //Array that stores timetable values for Rte rjj (m rows, a columns)  
  for i = 1 to m  
    for j = 1 to a  
      tj = "T" + cc //timepoint j of Rjj  
      //Random Arrivals  
      If SchDev = 2  
      oExcel.range("Al").select()  
      oExcel.ActiveCell.FormulaRlCl = "=LOGINV(RAND(), 2.900519 ,0.272637)"  
      // LOGINV(RAND(), u, SD)  
      rValue = oExcel.Range("Al").Value  
      if rValue <= (13.70758) and rValue >= (24.12115)  
        // Limit tail values smaller than 15th percentile and greater than 85th percentile  
        rValue = 0  
        else  
         rValue = rValue - 20  
        endif  
        else  
        endif  
        if r.fields[tj].value = 0  
         aRte_jj[i, j] = r.fields[tj].value  
        else  
         aRte_jj[i, j] = r.fields[tj].value + (rValue/60)  
            ?"aRte_jj[i, j] = "+i+", "+j+" "+(r.fields[tj].value + (rValue/60))  
        endif  
        next j  
        r.next()  
        next i  
  for f = bb to bb //Calculate Transfer Time TT from column bb  
  for g = cc to cc //to column cc  
  for i = 1 to n //for all rows i to  
  for j = 1 to m //all rows j  
    if aRte_ii[i, f] > 0 and aRte_jj[j, g] > 0 and aRte_ii[i, f] + time_st <=  

aRte_jj[j, g] //If time values are not zero and
//Rji <= RJj
// Transfer time TT calculation
    //do while aRte_ii[i, f] <= aRte_jj[j, g]
    do while (aRte_ii[i, f] + aGene[ii]) + time_st <= (aRte_jj[j, g] + aGene[jj])
        TT = (Rid_dat[ii, jj] * ((aRte_jj[j, g] + aGene[jj]) - (aRte_ii[i, f] + aGene[ii]))) //Transfer Times
        sum = sum + TT                  //Cumulative Transfer Times for all transfer combinations
        sum_move = sum_move + TT
        if aRte_jj[j, g] >= aRte_ii[i, f]
            TT = "" + TT + chr(9) + "hours" + chr(9) + "Arr.Time " + aRte_ii[i, f] + chr(9) + "Dep.Time " + aRte_jj[j, g] + chr(9) + "[ii-]j,i,f,j,g[" + chr(9) + "[" + "R" + ii + "-" + "R" + jj + "r" + i + "," + "c" + f + "," + "r" + j + "," + "c" + g + "] " + chr(9) + "Total= " + sum + chr(9) + "hours"
            //?aRte_ii[i, f], i, f
            //?aRte_jj[j, g], j, g
            exit
        enddo
        if aRte_jj[j, g] >= aRte_ii[i, f]
            exit
        endif
        else
            endif
        endif
    next j
    next i
    next g
    next f

    aMove[ii, jj] = sum_move  //Transfer time from Rte rii to Rte rjj
    Move_TT = aMove[ii, jj]

//?
//?"aMove["+ii+","+ jj+"] = " + Move_TT + " hours" //Transfer Time form rii to rjj
//?

    sum_move = 0
    else
    endif
    next jj
    next ii

//?"sum = " + sum
    ro.fields["TT"].value = sum  // Transfer Time
    ro.fields["CPU"].value = (DateTime()-cTimeStart)*24*3600  //CPU Time

//?
//?"Row " + ro.rowno() + ", Value = " + ro.fields["TT"].value
//?
    ro.next()

if total = 0
    total = sum
for l = 1 to rr/2
    Genes[k, l] = aGene[l]
    //?"Genes (" + k + ", " + l + ") = " + Genes[k, l] + ", h."
next l
elseif total > sum
    total = sum
for l = 1 to rr/2
    Genes[k, l] = aGene[l]
    //?"Genes (" + k + ", " + l + ") = " + Genes[k, l] + ", h."
next l
else
endif

next ch

//Close Excel - Random Arrivals
If SchDev = 2
    oExcel.ActiveWorkbook.Saved = True
    oExcel.ActiveWorkbook.close()
oExcel.quit()
    release object oExcel
    release oExcel
    oExcel = null
else
endif
dg = new Database() // //create one generation per loop
dg.databaseName = "Transfers"
dg.active = true
qg = new Query()
qg.database = dg
qg.sql = "select * from &gtablek order by TT ASC" //Sort by TT - Keep the best chromosomes on top
qg.active = true
ug = new UpdateSet()
ug.source = qg.rowset
gtablex = "Generationx"+k //Generation table
ug.destination = "Generationx"+k+.DBF"
ug.copy()
    Alter Table &gtablex Add Dummy Blob
    Alter Table &gtablex drop id
    Alter Table &gtablex Add id autoinc
    Alter Table &gtablex Drop Dummy
dg1 = new Database() // //create one generation per loop
dg1.databaseName = "Transfers"
dg1.active = true
qg1 = new Query()
qg1.database = dg1
qg1.sql = [select * from &gtablex where ID <= "] + (ss) + ["]
qg1.active = true
ug1 = new UpdateSet()
ug1.source = qg1.rowset
gtable = "Generation"+k //Generation table
ug1.destination = "Generation"+k+.DBF"
ug1.copy()
    if k = loops
        final_TT = qg1.rowset.fields["TT"].value
    else
endif
_app.databases[1].copyTable( gtable, "G"+k )
_app.databases[1].copyTable( gtable, "G_temp"+k )

_app.databases[1].copyTable( "G_temp"+k, "Temp" ) ///Include Children and Parents

use "G_temp"+ k
Append from Temp
use

dgen = new Database()
dgen.databaseName = "Transfers"
dgen.active = true
qgen = new Query()
qgen.database = dgen

genertable = "G"+k
qgen.sql = "select * from &genertable"
qgen.active = true
rgen = qgen.rowset

t.fields["TT"].value = rgen.fields["TT"].value
t.fields["CPU"].value = rgen.fields["CPU"].value
t.fields["Generation"].value = genertable

next k

??**** Solution ****??
??
??"Exist. Total TT = " + ExisTT + " Hours"
??
??"Final Total TT = " + final_TT + " Hours"
??
??"Number of Generations = " + loops
??
??"Number of Chromosomes = " + ss
??
??
??"Savings = " + (ExisTT - final_TT) + " Hours" + " (" + (100*((ExisTT - final_TT)/ExisTT)) + " %)"
??
??"Genes
??"-----
??"Time Shifts"
??"---------
??""'

// Results from the Genetic Algorithm

d = new Database() //create one generation per loop
d.databaseName = "Transfers"
d.active = true
q = new Query()
q.database = d
q.sql = "select * from man_genes"
q.active = true

r = q.rowset

df = new Database()
df.databaseName = "Transfers"
df.active = true
f = new Query()
f.database = df
gloops = "G"+loops
f.sql = "select * from &gloops"
f.active = true
rf = f.rowset

do while not r.endOfSet
  for k = 1 to 1
    for l = 1 to (rr/2) //Gene values of last Gk table ("G"+loops)
      // ? "Genes (" + l + ") = " + Genes[l] + " h.", round(Genes[l]*60,0)
      // ? "Genes (" + l + ") = " + chr(9) + round((Genes[k, l])*60,0)
      x = "G" + l
      //
      r.fields["genes"][l].value = round((rf.fields[x].value)*60,0)
      r.fields["genes"][l].value = rf.fields[x].value
      // "Genes (" + l + ") = " + chr(9) + r.fields["genes"][l].value
    r.next()
  next l
next k
enddo

use final
copy to final.txt delimited with ","
// copy to "final"+dmy(datetime())+seconds()+".txt" delimited with ","
use

CTimeEnd = DateTime() //End DateTime
// Comp_Time = elapsed(CTimeEnd, cTimeStart, 1) // Elapsed computing time
Comp_Time = (CTimeEnd-cTimeStart)*24*3600
?
? "Computing Time = " + Comp_Time + " sec."
close alternate // Close calc_ga.txt
Appendix D

dBASE On-Line Help

Random()

Returns a pseudo-random number between 0 and 1 exclusive (never 0 and never 1). Same as RAND().

Syntax

RANDOM([<expN>])

<expN>

The number with which you want to seed RANDOM().

Description

Computers cannot generate truly random numbers, but you can use RANDOM() to generate a series of numbers that appear to have a random distribution. A series of pseudo-random numbers relies on a seed value, which determines the exact numbers that appear in the series. If you use the same seed value, you get the same series of numbers.

Pseudo-random numbers, when considered as a whole series, appear to be random; that is, you cannot tell from one number what the next will be. But the first number in the series is related to the seed value. Therefore, you should seed RANDOM() only once at the beginning of each series, like before simulating a card shuffle or randomly assigning work shifts. Seeding during a series defeats the design of the random number generator.

If you specify a positive <expN> value, RANDOM() uses that <expN> as the seed value, so a positive value should be used for testing, since the numbers will be the same each time. If <expN> is negative, RANDOM() uses a seed value based on the number of seconds past midnight on your computer system clock. As a result, a negative <expN> value most likely will give you a different series of random numbers each time.

If you don't specify <expN>, or use zero, RANDOM() returns the next number in the series.

When dBASE Plus first starts up, the random number generator is seeded with a fixed internal seed value of 179757.

Use SET DECIMALS to set the number of decimal places RANDOM() displays.
VITA

FABIAN CEVALLOS

February 13, 1957
Born, Manta, Ecuador

1980-1985
B.S., Civil Engineering
Guayaquil State University
Guayaquil, Ecuador

1992-1993
M.S., Transportation Engineering
San Jose State University
San Jose, California

Research Assistant
San Jose State University
San Jose, California

1993-1995
Research Assistant
University of Nevada, Las Vegas
Las Vegas, Nevada

1996-2002
Transit Planner
Broward County Transit
Pompano Beach, Florida

2002-2006
Senior Research Associate
Center for Urban Transportation Research
University of South Florida
Tampa, Florida

2006-present
Transit Program Director
Lehman Center for Transportation Research
Florida International University
Miami, Florida

PUBLICATIONS AND PRESENTATIONS


HONORS AND AWARDS

Who is Who in American Colleges and Universities.
Dean’s List, San Jose State University.
Chi Epsilon National Civil Engineering Honor Society.
Delta Epsilon Iota Academic Honor Society.
Panel Member of TCRP Project A-29, Controlling System Costs: Basic and Advanced Scheduling Manuals and Contemporary Issues in Transit Scheduling.
Panel Member of TCRP Synthesis Topic SH-03: Applications of Geographic Information Systems (GIS) in Transit Planning and Operations.
Panel Member of TCRP Project H-28 (Uses of Archived AVL/APC Data to Improve Transit Performance and Management).
Bus Stop Inventory National Task Force.
Student Paper Award for the paper titled “Truck Factor: Basic Freeway Segments”, ITE District 6, Las Vegas, 1993.