To the Graduate Council:

I am submitting herewith a thesis written by Vickie Dawn Wester entitled "A Genetic Algorithm for the Vehicle Routing Problem." I have examined the final copy of the thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Management Science.

Dr. Charles E. Noon, Major Professor

We have read this thesis
and recommend its acceptance:

Accepted for the Council:

Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Vicki D. West

Date 11/30/93

A GENETIC ALGORITHM FOR THE VEHICLE ROUTING PROBLEM

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Vickie Dawn Wester
December 1993

ABSTRACT

The purpose of this research was to develop a version of a genetic algorithm (GA) which would provide near optimal solutions for Vehicle Routing Problems (VRP) with both time and weight constraints. The genetic algorithm used for the experimentation was adapted from a GA which had been developed by James Bean at the University of Michigan to solve machine scheduling problems. The VRP data sets used in this research were obtained from the literature. Various aspects of the GA were experimented with in order to develop a version which would perform consistently well for all the data sets. The results of the final version of the genetic algorithm were then compared to the results presented in the original papers.

The results from this research indicated that the genetic algorithm seems to perform relatively well for smaller problems with 50 or fewer customers. However, the results seem to become progressively worse as the problem becomes larger.

TABLE OF CONTENTS

PAGE

INTRODUCTION 1

CHAPTER

I. THE GENETIC ALGORITHM AND THE VEHICLE ROUTING PROBLEM. 4

The Vehicle Routing Problem. 4

The Genetic Algorithm 9

The Genetic Algorithm Related to the Vehicle Routing Problem 12

Data Format 13

Encoding of Solutions 14

Evaluating the Solutions 17

Reproduction 17

II. EXPERIMENTATION RESULTS 23

Penalization for Infeasibility 25

Sampling Procedures 27

Reproduction 28

Parameters 28

Infeasibility 31

Duplicate Solutions 32

Reverse Paths 32

Feasibility vs. Infeasibility 33

Gene Selection Parameter 34

Summary. 35

III. ANALYSIS 36

Parameter Settings. 36

Initial Population. 39

Search Space. 40

Elitism. 41

Encoding of Solutions. 42

Crossover Methods. 43

Convergence. 44

The Final Program 50
Results. 52
Proposed Improvements. 56
Conclusion. 58
REFERENCES. 60
APPENDICES 66
APPENDIX 1. GENETIC ALGORITHM 67
APPENDIX 2. REPRESENTATION OF TOURS 80
VITA. 92

LIST OF TABLES

TABLE		PAGE
3. 1.	Problems used for experimentation	53
3. 2.	Comparison of GA results with other methods of solving problems	54
A2. 1.	22 customers, 3 vehicles without time constraint	81
A2. 2.	29 customers, 3 vehicles without time constraint	81
A2. 3.	32 customers, 3 vehicles with time constraint	81
A2. 4.	32 customers, 3 vehicles without time constraint	81
A2. 5.	50 customers, 6 vehicles with time constraint	82
A2. 6.	50 customers, 5 vehicles without time constraint	82
A2. 7.	75 customers, 11 vehicles with time constraint	82
A2. 8.	75 customers, 10 vehicles without time constraint	83
A2. 9.	75 customers, 14 vehicles without time constraint	83
A2.10.	100 customers, 9 vehicles with time constraint	84
A2.11.	100 customers, 8 vehicles without time constraint	84
A2.12.	100 customers, 11 vehicles with time constraint	85
A2.13.	100 customers, 10 vehicles without time constraint	85
A2.14.	100 customers, 14 vehicles without time constraint	86
A2.15.	120 customers, 11 vehicles with time constraint	86
A2.16.	120 customers, 7 vehicles, without time constraint	87
A2.17.	150 customers, 14 vehicles, with time constraint	87
A2.18.	150 customers, 12 vehicles, without time constraint	88
A2.19.	199 customers, 18 vehicles, with time constraint	89
A2.20.	199 customers, 16 vehicles, without time constraint	90
A2.21.	199 customers, 17 vehicles, without time constraint	91

Introduction

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem in which a number of customers, requiring either pick-ups or deliveries, must be serviced by a set of vehicles. The objective is to route the vehicles in such a manner that each customer is visited by exactly one vehicle and the total distance traveled is minimized. The vehicles may be constrained by a load capacity or a maximum time spent on the route. Vehicle routing problems are complex to solve, particularly to optimality, causing many "algorithm designers" to settle for an approximation of the optimal solution [Haimovich, et al 1988]. The focus of this thesis is to experiment with the genetic algorithm (GA) as a means of solving the VRP.

According to Goldberg, the goal of genetic algorithms is to be efficient and robust over different environments in order to eliminate costly redesigns in the programs. Genetic algorithms are described as "computationally simple yet powerful" and are not limited by restrictive assumptions such as, "continuity, existence of derivatives, unimodality, and other matters" [Goldberg 1988]. However, according to Davis, "..., in general, the robustness of a genetic algorithm and its performance on a particular problem are inversely related" [Davis 1991]. Genetic algorithms are robust in that they can be used to solve several different problem types without changing the algorithm. A slight change in the problem could make a nonrobust algorithm inoperative [Davis 1987].

Genetic algorithms first appeared in theory in the early 1970's, but John Holland is said to have founded the field of genetic algorithms in 1975. In more recent years, work on genetic algorithms has been focused on application [Davis 1991]. Holland's original idea in developing the algorithm was to create a program which would adapt to its environment [Goldberg 1988]. The genetic algorithm was first used in industry to optimize the design of a communications network [Davis 1987]. There are several areas in which GA performance has been studied. The following is a partial list of areas for which genetic algorithms have been studied:

1. Davis (1985) - Job shop scheduling
2. Glover (1987) - Keyboard configuration systems
3. Goldberg (1983) - Optimizing gas pipeline systems
4. Grefenstette (1985) - Traveling salesman problem
5. Nygard and Kadaba (1990) - Multi-vehicle routing problem [Nygard 1992].

Atidel Ben Hadj-Alouane (1992) at the University of Michigan successfully used a genetic algorithm to solve multiple choice integer programs with nonlinear relaxation. The algorithm successfully solved 100% dense problems and had computation times superior to IBM's Optimization Subroutine Library (OSL). Hadj-Alouane noted three advantages of the genetic algorithm when compared to OSL after running the genetic algorithm on three facility location problems:

1. The optimal solution was found for all three, yet less time was taken than with OSL.
2. There was a small variation in solutions for different random seeds.

3. The GA was more scalable (i.e. The time needed to solve the problem was predictable based on the size of the problem.) [Hadj-Alouane and Bean 1992].

Genetic algorithms have been shown to work effectively on problems such as function optimization problems, but only recently has experimentation moved into combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) or the Vehicle Routing Problem [Suh and Gucht 1987]. The purpose of this research is to present and test a genetic algorithm for the vehicle routing problem. In addition, this work examines the obstacles encountered when applying GA's to vehicle routing problems, as well as possible methods for handling them. The first chapter gives an overview of the vehicle routing problem and genetic algorithm and provides a discussion of how the two are related. Chapter 2 describes key elements of the GA which were studied in order to gain an understanding of their impact on algorithm performance. Chapter 3 presents a final version of the GA as well as other alternatives which may be studied in future research. The results of this version of the GA are also given for selected data sets which have appeared in various VRP articles. These results are compared with the best known solutions obtained by the other methods of solving the VRP.

CHAPTER 1
THE GENETIC ALGORITHM AND THE VEHICLE ROUTING PROBLEM

Chapter 1 is an introduction to the genetic algorithm and the type of vehicle routing problem addressed in this thesis. The first section describes the VRP and some of the common heuristic methods currently being used to solve problems of this type. There are several important aspects of the GA which must be considered during development. A few of these aspects are encoding of solutions, evaluation function, parameter values, selection methods, crossover methods, and mutation methods. The second section explains the terminology of the GA and describes various ways of representing some of these important aspects. The last section of this chapter specifically describes the aspects of the GA which were used to solve the VRP.

The Vehicle Routing Problem

The vehicle routing problem addressed in this thesis is one consisting of a single depot, n customers, and m vehicles. For each customer, the vehicle must pick up a certain amount of weight, w_j, where j is the customer number. For problems with time considerations there is a constant stop time, s, at each customer. The objective (1) is to minimize the total distance traveled by all vehicles where d_{ij} is the distance from customer i to customer j. The binary variable y_{mij} will equal 1 if vehicle m

goes from customer i to customer j and 0 otherwise. A formulation for the VRP is given below:

(1) $\min \sum_m \sum_{ij} d_{ij} y_{mij}$

(2) s.t. $\sum_m \sum_{i,i<>j} y_{mij} + \sum_m \sum_k y_{mjk} = 2$ for all j

(3) $\sum_j y_{m0j} = 1$ for all m

(4) $\sum_{i,i<>j} y_{mij} \leq 1$ for all m, j

(5) $\sum_i y_{mi0} = 1$ for all m

(6) $\sum_i y_{mij} - \sum_k y_{mjk} = 0$ for all m, j

(7) $\sum_i \sum_j y_{mij} \leq |S| - 1$ for all subsets S, for all m

(8) $\sum_{i,j} y_{mij} w_j \leq w$ for all m

(9) $\sum_{i,j} (d_{ij} + s) y_{mij} \leq t$ for all m

[Noon, et al 1991]. Constraint (2) ensures that exactly one vehicle visits and leaves each customer. Each vehicle is forced to leave the depot by constraint (3). Constraint (4) ensures that a vehicle does not visit a particular customer more than once and constraint (5) ensures that the vehicle returns to the depot [Noon, et al 1991]. Flow conservation for each vehicle tour is enforced by constraint (6). Subtours are eliminated by constraint (7). The capacity constraint (8) ensures that the total amount of weight picked up by the vehicle does not exceed a weight limit of w. Constraint (9) ensures that the total time on the route for each vehicle can not exceed a time limit of t, where the time on a route is calculated by the distance on the route plus the sum of all the stop times on the route.

"The vehicle routing problem is a hard combinatorial problem and to this day, only relatively small VRP instances can be solved to optimality." [Gendreau, et al 1991] There are four groups in which heuristic methods for solving the VRP can be divided. They are

constructive, two-phase, incomplete optimization, and improvement algorithms. The first of these four is the constructive algorithm in which an unrouted city is selected to be added to the tour based on some criterion [Gendreau, et al 1991]. One of the most common of these methods is the Clarke and Wright method in which n back and forth routes between a city and the depot are merged according to a savings criterion. The major disadvantage with this method is the amount of time required to find a near optimal solution. However, data structures can be used to reduce the amount of time to run the algorithm [Gendreau, et al 1991].

There are four types of two-phase algorithms. The first of these is the cluster first - route second method in which each vehicle is first assigned the customers which it must visit. The TSP is then used to sequence each of the routes. The TSP is a combinatorial optimization problem in which a single vehicle leaves a depot and must visit each customer exactly once and return to the depot. The objective is to sequence the customers in such a way that the distance traveled is minimized. The second method is the route first - cluster second approach in which the TSP is first used to sequence the customers. The route is then broken into feasible segments for each vehicle available. The third method is an integer linear programming approach using the Generalized Assignment Problem and the TSP. This approach was developed by Fisher and Jaikumar (1981). Finally, the fourth method of two-phase algorithms is a Lagrangean Relaxation Approach used by Noon, Mittenthal, and Pillai (1991) [Gendreau, et al 1991]. The method relies on solving a Traveling Salesman Subset Tour Problem with one additional constraint. The TSSP is a variant of the TSP in which the constraints that require each customer

to be visited are relaxed. The idea is to have a dispatcher who assigns each vehicle an initial customer to visit and then assigns reward values to every other customer. The objective when preassigning these customers is to maximize the minimum distance between any two of them. Each vehicle driver then decides which customers to visit and the corresponding sequence of visits. The objective of the dispatcher is to assign the rewards so that each customer will be visited by exactly one vehicle. The major difference between this approach and that of Fisher and Jaikumar is that the dispatcher in the Fisher method decides which customers each driver visits, and the driver is only responsible for sequencing the route. In the Lagrangean Relaxation approach, the driver has the additional responsibility of deciding which customers to visit [Noon, et al 1991].

The third heuristic method is Incomplete Optimization. This approach uses an enumerative algorithm to find a good solution by means of an incomplete search tree [Gendreau, et al 1991].

Finally, the fourth heuristic method used is Improvement Methods which is the category in which tabu search falls. Among the tabu search methods that exist are one developed by Pureza and Franca (1991) in which cities are swapped between two routes and one developed by Semet and Taillard (1991) in which a city is moved from one route to an alternate route [Gendreau, et al 1991]. The algorithm developed by Gendreau, Hertz, and Laporte inserts a node into a tour from another tour using a generalized insertion procedure (GENI). A tour improvement procedure which was also developed by Gendreau, Hertz, and Laporte, is used to improve each route. Once a customer is taken out of a particular vehicle's tour, it cannot be put back into that tour for a certain number of iterations.

One difference between this method and the other tabu search methods is that it allows infeasible solutions whereas the others do not. An advantage to using this method, called TABROUTE, is that the risk of converging to a local optima is reduced in two ways. The first is by allowing infeasible solutions through the use of a penalty function. The second is by using GENI to perform the insertion of the customer into a different route [Gendreau, et al 1991].

Another improvement method which has successfully been used was developed by Ibrahim Osman at the University of Canterbury [Osman 1993] and solves the vehicle routing problem using simulated annealing and tabu search. This method finds a route by first using a heuristic followed by an improvement method in which a portion of one route is exchanged with a portion of a second route. An insertion/deletion procedure is used to recalculate the objective value, and the 2-opt arc exchange heuristic of Lin [Osman 1993] is used to correct any paths that are crossed. There are two selection strategies used for selecting alternative solutions: best improvement and first improvement. The tabu search consists of a forbidding strategy, a freeing strategy, a short-term strategy, and a stopping criterion. The forbidding strategy keeps a list of the moves which are forbidden. The freeing strategy removes the moves from the tabu list after a certain number of iterations. The short-term strategy uses an aspiration criterion to overrule the tabu list and includes two possible selection strategies: Best Admissible (BA) and First Best Admissible (FBA). BA selects the move resulting in the greatest improvement or the least nonimprovement. FBA selects the first move resulting in an improvement in the objective value if one exists; otherwise, the best

nonimproving move is selected. The tabu list sizes are calculated as a function of population size, number of vehicles, and capacity ratio of required demands to vehicle capacities. The stopping criterion is based on a maximum number of iterations in which the best solution does not improve after the best solution was found. One potential drawback of tabu search is that quality of the final solution depends on the initial solution. Hence, the method sometimes finds a local optimal which is not close to the global optimal. This is the same problem which Gendreau addressed in his method by allowing infeasible solutions. Osman's method uses simulated annealing (SA) to overcome this problem. SA accepts a nonimprovement move based on a certain probability which is determined by a control parameter which decreases according to a schedule [Osman 1993].

The Genetic Algorithm

The genetic algorithm was developed by John Holland in 1975 and uses the idea of genetics and "survival of the fittest" to produce near optimal solutions to problems such as the traveling salesman problem, machine scheduling problems, vehicle routing problems, and many others. The basic concept behind this algorithm is that good solutions will remain in the population and continue reproducing to form better solutions while the most undesirable solutions eventually become extinct. Initially, a population of solutions is randomly generated, and each solution in the population is called a *chromosome*. For example, consider a solution which is encoded as a sequence of customer numbers:

$$(5\ \ 1\ \ 4\ \ 3\ \ 6\ \ 2).$$

This chromosome represents a solution in which the sequence of customer visits is 5,1,4,3,6,2. Each position of this chromosome is called a *gene*, and the value of each gene is called an *allele*. For example, 5 is the allele of the first gene [Nygard 1992]. At each generation there are a number of methods which can be used to produce a new population of solutions. Although all genetic algorithms use some form of reproduction, crossover, and mutation, there are many different ways of carrying out these operations. The next section describes some of the alternate methods.

First, there are a number of different ways to select the chromosomes to be added to the mating pool. The challenge is to select the parents in such a way that the good parents reproduce enough to survive, but not so much as to cause the population to prematurely converge [DeJong 1985]. There is still disagreement among researchers on the best method of parent selection. Four of the most common methods are listed below.

1. Random selection of the chromosomes.
2. Roulette sampling in which the probability of selecting a particular chromosome increases with its fitness.
3. Rank based sampling which uses the roulette wheel to select two chromosomes, of which the one with the best fitness is added to the mating pool.
4. Tournament sampling in which solutions are sequentially chosen with the one having the higher fitness being added to the mating pool [Nygard 1992].

Next, there are several different methods of crossing over the two parent chromosomes. There is the one-point crossover in which a point on

the chromosome is randomly selected, and the two chromosomes exchange the genes following this point. The disadvantage of using the one-point crossover is that if good genetic material is at both ends of the chromosome, these two good traits will be separated during the crossover. The two-point crossover solves the one-point crossover problem by enabling two genes on opposite ends of the chromosome to remain on the same chromosome after the crossover. This is accomplished since two points are randomly selected on the chromosome, and the genes between these two points are exchanged between the two chromosomes. However, this still may present a problem if, for example, all of the good traits are on one of the chromosomes [Davis 1991]. The best crossover method seems to be the uniform crossover in which a random number (between 1 and 100) is generated for each gene. If this number is less than a certain user defined number (which is defined at the beginning of the GA as the gene selection parameter), the child will receive this gene from the first parent. If the number is greater than the gene selection parameter, the child receives the gene from the second parent. Unlike the one-point and two-point crossovers, this crossover method has the ability to combine good traits irrespective of where they are located on the chromosome [Davis 1991].

Also, there are differences in the methods of producing mutations. One method is to simply mutate a single gene at a certain rate (i.e. 1 out of every 1000 genes). However, since mutation is the main means of producing variation in the population, mutating a single gene does not seem to be very efficient. Another, more efficient method is to mutate the entire chromosome for a low percentage of the chromosomes in the population

[Davis 1991]. This method of mutation is referred to as immigration [Bean 1992]. This seems to provide more diversity in the population.

Two very important links to the genetic algorithm and the problem to be solved are the method of evaluating new solutions and the method used to encode a solution [Davis 1991]. These two aspects are crucial because they must be tailored to the problem being solved. The other aspects of the GA such as parameter values, selection methods, crossover methods, and mutation methods do not represent the problem being solved and may be exactly the same for a variety of different problems. The simple genetic algorithm seems to be powerful despite the lack of knowledge of the problem to be solved [Goldberg and Richardson 1987].

The Genetic Algorithm Related to the Vehicle Routing Problem

James Bean at the University of Michigan [Bean 1992] used a genetic algorithm to solve machine scheduling problems. It was his GA which was modified in this thesis to solve vehicle routing problems. Bean was successful using this program on scheduling and resource allocation problems, and he had moderate success on quadratic assignment problems. However, his tests on several traveling salesman problems were not as successful. He reported difficulty in getting closer than 8% to the optimal solution; however, the results did not seem to worsen as the problem size increased. In addressing this problem, he states that, "we conjecture that these difficulties are caused by the complexity of interrelationship between pairs of genes (cities or agents)." [Bean 1992] The issue now addressed is how this genetic algorithm relates to the vehicle routing problem described earlier. The aspects of the GA which must be considered are: the input

format of the data set to be used by the genetic algorithm, the method of encoding a solution, the method of evaluating a solution, and the method of reproduction.

Data Format

The genetic algorithm code first reads VRP problem data in the following format. The first line of any problem data set contains the following information:

1. The number of customers + 1 (for the depot)
2. The weight limit of each vehicle
3. The number of vehicles
4. The amount of time at each stop
5. The time limit of each route

The next n lines of the data set (where n is the number of customers) contain the following:

1. The x coordinate of the customer
2. The y coordinate of the customer
3. The amount of weight to be picked up at the customer
4. The customer number

The last line of the data set contains the x and y coordinates of the depot. Figure 1.1 is an example data set for the 32 customer, 3 vehicle VRP. The first line indicates that there are 32 customers plus 1 for the depot, there is a weight limit of 38000 units per vehicle, there are 3 vehicles, there is a stop time of 20 units of time at each stop, and there is a time limit of 1000 units per vehicle. Lines 2 through 33 consist of the x and y coordinates of each customer, the amount of weight to be picked up at each customer, and

the customer number. The last line indicates the x and y coordinates of the depot.

33	38000	3	20 1000
10	260	3500	1
65	248	1260	2
22	255	629	3
50	249	250	4
205	254	2267	5
275	34	447	6
269	262	1847	7
293	269	1437	8
333	212	3720	9
304	202	1115	10
286	207	273	11
288	191	5494	12
295	235	1944	13
467	67	713	14
484	179	1500	15
447	189	3585	16
215	204	140	17
313	382	25705	18
267	316	479	19
391	196	17456	20
399	122	1143	21
363	187	1919	22
355	236	826	23
378	203	3264	24
458	218	1570	25
383	181	2215	26
240	326	1239	27
273	349	580	28
278	374	5000	29
352	271	100	30
324	295	201	31
249	250	6747	32
250	200	0	DEPOT

Figure 1.1. 32 customer, 3 vehicle problem data set.

Encoding of Solutions

The algorithm uses a method of encoding a solution called *random keys* which was developed by Bean [Bean 1992]. Random keys is a

technique designed to address the problem of producing infeasible solutions during reproduction because some customers are visited more than once while some are not visited at all [Bean 1992]. To illustrate the problem of infeasibility during reproduction consider a problem with only one vehicle and six customers. When the solution is encoded using the customer number, reproduction can lead to infeasible offspring as shown in the example below. Consider two parents whose solutions are encoded as sequences of customer numbers.

parent 1: (5 1 4 3 6 2)

parent 2: (4 5 1 3 6 2)

In parent 1, the sequence of customer visits is 5, 1, 4, 3, 6, 2. In parent 2, the vehicle visits customers in the order 4, 5, 1, 3, 6, 2. When these two undergo reproduction, a child is produced by selecting a gene from each parent with a certain pre-defined probability. For each gene, a random number is generated. If the random number is greater than the probability assigned to parent number 1, the gene is taken from parent number 2; otherwise, the gene is taken from parent number 1. If, for example, the probability of selecting a gene from parent 1 is .70 and .30 from parent 2, the following situation might occur:

random number:	.86	.55	.40	.12	.73	.23
parent number:	2	1	1	1	2	1
child:	4	1	4	3	6	2

The child produced is infeasible since customer number 4 is visited twice during the tour and customer number 5 is not visited at all [Bean 1992].

Random keys is designed to prevent this type of reproduction infeasibility. The idea behind random keys is to generate a random

number between 0 and 1 for each customer in a solution. The order in which the customers are visited is represented by sorting the random numbers in ascending order. For example, in the previous problem, parent 1 might be represented as follows:

(.31 .95 .76 .51 .15 .85)

where customer number 5 is the first visited, customer number 1 is the second visited, etc. The two parents would then be represented as follows:

parent 1: (.31 .95 .76 .51 .15 .85)

parent 2: (.33 .83 .49 .08 .25 .71)

These two parents reproduce as follows:

random number:	.86	.55	.40	.12	.73	.23
parent number:	2	1	1	1	2	1
child:	.33	.95	.76	.51	.25	.85

The new child solution is now feasible, with the order in which the customers are visited represented by the order of the random numbers [Bean 1992].

The problem of representing multiple vehicles in a solution can also be solved using random keys. A random integer, between 1 and the number of vehicles, is added to each random number. The integer represents the vehicle which visits that customer. For example, if two vehicles are available, the solution might be represented as:

(2.31 2.95 1.76 1.51 1.15 2.85)

where vehicle 1 visits customer 5 followed by customer 4, then customer 3, and vehicle 2 visits customers 1, 6, and 2.

According to Bean, "we have successfully generalized this approach to the job shop with precedence, release times, sequence dependent setups,

and nonregular measures such as a sum of weighted earliness and tardiness." [Bean 1992] Random keys also appears to be an efficient means of encoding solutions for the vehicle routing problem; therefore, it is the method used for our research.

Evaluating the Solutions

One important feature which relates the genetic algorithm to the problem being solved is the method of evaluating the fitness of each chromosome (solution). The method used for this particular problem is to first calculate the total Euclidean distance traveled by all vehicles on the tour. Then, the amount of time each vehicle spends on the tour and the weight that each vehicle picks up throughout the tour is calculated. From these calculations it can be determined how much each vehicle exceeds the time limit and weight limit, as well as how many vehicles have infeasible tours. The fitness of the solution is then calculated by a function of the distance traveled in combination with a penalty function for infeasibility with respect to weight and time.

Reproduction

The method of reproduction is another important aspect of the genetic algorithm. The method which Bean uses in his algorithm keeps a consistent number of solutions in the population throughout the algorithm. A certain percentage of the top solutions are copied to the next generation. This is called *elitism* or *clonal propagation* and enables the best solutions to be preserved [Davis 1991]. Another percentage of the new generation is produced by mutation. The method of mutation used is to randomly

generate completely new chromosomes by the same method in which the population was initialized at the beginning of the algorithm. Since the elitist strategy is being used, a fairly high mutation rate must be used in order to maintain diversity in the population [Bean 1992]. The remaining percentage of solutions in the new population are produced through reproduction. The parent chromosomes are selected randomly with equal probability of being selected and then the uniform crossover method is used for best results.

EXAMPLE 1.1.

In order to illustrate some of the important aspects of the genetic algorithm which were discussed in this chapter, consider an example VRP which consists of 5 customers and 2 vehicles with a best known solution of 63. Suppose at the beginning of the algorithm the following parameters are defined:

5 members in the population

25 generations

1 solution copied into the next generation

1 solution mutated each generation

Gene selection parameter of 50

At the beginning of the algorithm an initial population is randomly generated. The solutions are evaluated and ordered so that the solution with the best evaluation is first and the solution with the worst evaluation is last. Suppose that after the solutions are ordered, the initial population is as follows:

	Solution	Evaluation
#1:	(1.53 2.14 2.07 1.10 1.76)	97
#2:	(1.08 1.56 2.58 2.13 2.33)	113
#3:	(2.82 2.91 2.67 2.15 1.01)	125
#4:	(2.61 1.50 1.39 2.43 1.59)	156
#5:	(1.89 2.17 1.25 2.95 1.52)	208

The following shows an example of how the next generation of five solutions may be produced from the initial population.

Solution # 1

The best solution from the initial population is copied to form one member of the new population. (The number to be copied was defined at the beginning of the algorithm to be 1.)

(1.53 2.14 2.07 1.10 1.76) Evaluation = 97

Solution #2

Solutions #2 and #5 are randomly selected from the initial population to reproduce with each other. Reproduction occurs in the following manner using uniform crossover.

Random number:	22	56	08	34	76	
Parent #1:	(1.08	1.56	2.58	2.13	2.33)	
Parent #2:	(1.89	2.17	1.25	2.95	1.52)	
Child:	(1.08	2.17	2.58	2.13	1.52)	Evaluation = 128

Solution #3

Solutions #1 and #2 are randomly selected from the initial population to reproduce with each other. Reproduction occurs in the following manner.

Random number:	58	64	19	36	79	
Parent #1:	(1.53	2.14	2.07	1.10	1.76)	
Parent #2:	(1.08	1.56	2.58	2.13	2.33)	
Child:	(1.08	1.56	2.07	1.10	2.33)	Evaluation = 150

Solution #4

Solutions #3 and #1 are randomly selected to reproduce with each other.

Random number:	70	18	83	95	34	
Parent #1:	(2.82	2.91	2.67	2.15	1.01)	
Parent #2:	(1.53	2.14	2.07	1.10	1.76)	
Child:	(1.53	2.91	2.07	1.10	1.01)	Evaluation = 90

Solution #5

This solution is formed by mutation which means a random number is generated for each gene on the chromosome. (The number of solutions to be mutated was defined at the beginning of the algorithm to be 1.) The following numbers are generated:

(2.54 2.99 1.76 2.11 1.69) Evaluation = 200

This new population would then be ordered by evaluation as follows:

	Solution	Evaluation
#1:	(1.53 2.91 2.07 1.10 1.01)	90
#2:	(1.53 2.14 2.07 1.10 1.76)	97
#3:	(1.08 2.17 2.58 2.13 1.52)	128
#4:	(1.08 1.56 2.07 1.10 2.33)	150
#5:	(2.54 2.99 1.76 2.11 1.69)	200

This same method of reproduction, in which the current population goes through a reproduction phase to form a new population, is used for the next 24 generations. (The number of generations was defined at the beginning of the algorithm to be 25.)

Premature convergence is a problem which must be addressed in the genetic algorithm. Convergence is the reason the mutation operator is necessary. Suppose that after the 10th generation the population for this example problem is the following:

	Solution	Evaluation
#1:	(1.53 2.91 2.07 1.10 1.01)	90
#2:	(1.53 2.91 2.07 1.10 1.01)	90
#3:	(1.53 2.91 2.07 1.10 1.76)	90
#4:	(1.23 1.14 2.19 2.57 1.18)	97
#5:	(2.89 2.75 1.28 2.15 1.87)	210

This is an example of a population which is prematurely converging. Since the best known solution is 63 and the best solution the algorithm has found is 90, it is obvious that the algorithm should not yet be converging. Notice

that the first two solutions are identical, and the third solution is only different on the fifth gene. The last two chromosomes are now the only means for diversity in this population. This example will be referred to later in the thesis to demonstrate the dynamics of the population under certain conditions.

CHAPTER 2
EXPERIMENTATION RESULTS

In the previous chapter the VRP and the GA were described, and an example was given to illustrate some of the important aspects. The purpose of chapter 2 is to describe some of the experimentation which was performed on the GA. This chapter should provide an idea of the various ways in which the GA can be altered and the effects these alterations may have on the results. Figure 2.1 provides a summary of the aspects which were altered during the experimentation.

---

1. PENALIZATION FOR INFEASIBILITY
    - Assigning a penalty to all infeasible solutions
2. SAMPLING PROCEDURES
    - Roulette, Tournament
3. REPRODUCTION
    - Single Crossover, Double Crossover
4. PARAMETERS
    - Changing number of mutations as a function of generation count
5. INFEASIBILITY
    - Replacing infeasible solutions with the best solution
6. DUPLICATE SOLUTIONS
    - Mutating duplicate solutions
7. REVERSE PATHS
    - Ensuring against paths which visit the same customers only in opposite order
8. FEASIBILITY VS. INFEASIBILITY
    - Ensuring all feasible sols. evaluate better than all infeasible sols.
9. GENE SELECTION PARAMETER
    - Producing offspring at different gene selection parameters

---

Figure 2.1. Summary of experimentation.

The purpose of the experimentation was to produce a version of the genetic algorithm which would find a nearly optimal solution for most vehicle routing problem data sets of the format discussed in Chapter 1. There were two data sets most commonly used for the experimentation. The first data set was a 50 customer, 6 vehicle, VRP with a weight limit of 160 per vehicle, a time limit of 200 per vehicle and a stop time of 10. The best known solution from the literature has a total cost (distance) of 555.43 [Gendreau, et al 1991]. The best solution known for this same data set with only 5 vehicles and no time constraint is 524.61 [Gendreau, et al 1991]. The second data set had 32 customers and 3 vehicles with a weight limit of 38000 per vehicle, a time limit of 1000 per vehicle, and a stop time of 20. The best known solution for this data set has a total rounded cost of 2086. Without the time constraint, the best known solution to this problem is 2009.31 [Noon, et al 1991]. There were several procedures in the GA which were believed to have some effect on the performance of the algorithm; namely, calculating the fitness of the chromosome, infeasibility of solutions, sampling, reproduction, operator fitness, format of the solutions (duplicates, reverse paths), and the gene selection parameter value (probability of selecting a gene from chromosome number 1).

First, the program was run using Bean's algorithm with the exception of a few changes necessary to run the algorithm on the vehicle routing problem rather than the machine scheduling problem. These changes primarily involved the input of the data set and the evaluation of solutions. Also, Bean suggested that a few of the parameter values be changed. The parameter values were as follows:

Number of generations - 2000

Number of members in the population - 100

Number of chromosomes repeated in the next generation - 20

Number of mutations each generation - 2

Gene selection parameter value - 70.

The method of sampling and reproduction remained unaltered. The procedure for calculating chromosome fitness was to calculate the total distance traveled on the tour, with no penalty for infeasibility of the solution, and assign this value as the fitness value. (Note: For the initial experimentation, rounded solutions are given. The exact solutions are given for the final version of the program in the results section of Chapter 3.) This version produced a solution with a cost of 555 for the 50 customer, 6 vehicle problem which was a good solution; however, it was infeasible with 3 vehicles exceeding the time limit and one exceeding the weight limit. The solution for the 32 customer, 3 vehicle problem was 2014; however, it was also infeasible with 1 vehicle over the time limit and 2 over the weight limit. Since the algorithm seemed to be producing good results with the exception of infeasibility, the first task undertaken was to develop a method of penalizing for infeasibility in order to allow the feasible solutions to move to the top of the gene pool.

Penalization for Infeasibility

The method of calculating the fitness of the solution was the first procedure tested to observe the effect this would have on the results. The first method of penalization attempted was to penalize each infeasible solution by assigning its cost to be a large value. This method prevented

effective reproduction because the same large number was assigned to each solution regardless of its degree of infeasibility. At the start of the algorithm virtually all solutions are infeasible since they have all been randomly generated. Therefore, the "survival-of-the-fittest" philosophy with this type of penalization is not very effective since no solution's fitness is better than any other. The next method of assigning a fitness to the solution was to add a penalty to the total distance traveled on the tour based on type and degree of infeasibility of the solution. The weight penalty was calculated by summing the amount each vehicle exceeded the weight limit and raising this value to some power. The total penalty was evaluated by adding the weight penalty to the time penalty which was calculated by summing the amount each vehicle exceeded the time limit and raising this value to the same power. First, a power of two was used which produced a solution of 617 for the 50 customer problem; however, the solution was still slightly infeasible with one vehicle being one unit over the time limit. The solution to the 32 customer problem was 2317 and was feasible. Using a power of three, the solution to the 50 customer problem was 654 and was feasible, and the solution to the 32 customer problem was 2267 and was also feasible. Finally, the solution to the two problems using a power of four was 639 for the 50 customer problem and 2344 for the 32 customer problem with both solutions being feasible. The next method of penalization was the same as the previous method using a power of 2, except that the penalty was multiplied by the number of vehicles which were infeasible with respect to weight plus the number vehicles which were infeasible with respect to time. This method gave an improvement in the solutions of both data sets. The solution to the 50 customer, 6 vehicle

problem was 601, and the solution to the 32 customer, 3 vehicle problem was 2298.

Sampling Procedures

Next, the sampling procedure was changed to observe the effects this would have on the solution. The first sampling procedure tested was the roulette wheel sampling method [Davis 1991]. Below is a list of the steps followed in this method:

1. Find the largest solution value of the n solutions in the population, **lval**.
2. For each solution's value, **sval(i)**, find **fval(i) = lval - sval(i)**. This is a measure of the fitness of solution **i** relative to the other solutions in the population.
3. Assign each solution a range of numbers, the size of which corresponds to the size of its relative fitness value.
   Range(i) = $[\sum^{i-1} \mathbf{fval(k)}, \sum^{i} \mathbf{fval(k)}]$
4. Generate a random number between 1 and $\sum^{n}$ **fval(i)**.
5. Pick the solution whose range contains this random number.

On the data sets tested, the performance of the roulette wheel sampling procedure was inferior to the performance of the sampling procedure in which all solutions had equal weight. The best solution for the 50 customer problem was 807 and was infeasible. A tournament sampling method also was used in which eight solutions were chosen to reproduce with each other resulting in four solutions which reproduced to form two. These two solutions then reproduced to form one solution which was added to the population. This procedure was only run on the 50 customer problem due

to time limitations. The resulting solution was relatively good with value, 602, and was feasible; however, the extensive amount of time it took to run the algorithm with this sampling procedure made it very impractical.

Reproduction

Two alternate methods of reproduction were tested to determine their impact on performance. These two methods, single crossover and double crossover reproduction, were described in Chapter 1 along with their disadvantages. Although these methods have disadvantages, the experiments were run in order to observe the changes in the results by using these methods. The results of the double crossover method were significantly better than the single crossover; however, it produced a solution of 2370 for the 32 customer problem and 849 for the 50 customer problem, which were significantly worse than with the uniform crossover previously used.

Parameters

Two important parameters which seem to significantly affect the results of the algorithm are *nrep* and *nmut*. The parameter *nrep* controls how many chromosomes are copied from one generation to the next. The parameter *nmut* controls how many solutions will be mutated in each generation. A key consideration for nrep is that it must be high enough to keep the best solutions but not so high as to cause the population to converge prematurely. For instance, consider Example 1.1. If the number of solutions to be copied is increased from 1 to 2, the population might have the following appearance after the 10th generation.

	Solution	Evaluation
#1:	(1.53 2.91 2.07 1.10 1.01)	90
#2:	(1.53 2.91 2.07 1.10 1.01)	90
#3:	(1.53 2.91 2.07 1.10 1.01)	90
#4:	(1.53 2.91 2.07 1.10 1.18)	101
#5:	(2.89 2.75 1.28 2.15 1.87)	210

By setting nrep too high, the population may prematurely converge more quickly than it would have otherwise. However, if the number to be copied was reduced to 0, the best solution might be lost.

The mutation operator, nmut, must be high enough to induce variation but not so high as to cause the population to converge towards poor solutions. Consider Example 1.1 again. Suppose the number of mutations per generation is increased from 1 to 2. After 10 generations, the population might have the following appearance.

	Solution	Evaluation
#1:	(1.53 2.91 2.07 1.10 1.01)	90
#2:	(1.09 1.51 2.07 1.10 2.58)	102
#3:	(2.54 2.32 1.14 1.39 1.21)	125
#4:	(1.67 1.28 2.19 2.57 1.18)	153
#5:	(2.89 2.75 1.28 2.15 1.87)	210

In this situation, the increased mutations provides such diversity that the better solutions are overwhelmed by the poorer solutions which were produced by mutation.

According to Davis, the speed of convergence of the population and the nearness of the individuals to local optima are related to these two parameter values [Davis 1991]. In order to prevent premature convergence of the population, experiments were run to determine which mixture of parameter values gave the best results for the two data sets. The version of the GA used for these experiments included the sampling procedure in which all solutions had an equal probability of being selected and employed uniform crossover as the reproduction method. The method of penalization for infeasibility for these experiments took into consideration the amount the time and weight limits were exceeded and the number of vehicles not meeting the constraints. After testing the operators over a range of values, the best results were found when nrep was set at 20, and nmut changed as a function of the generation count. The operator, nmut, was initially set at 2 and changed to 5 at generation count 500 and then to 8 at generation count 1000. The best solution for the 32 customer problem was found to be 2321, and the best solution for the 50 customer problem was 587, but was infeasible because one vehicle was over the time limit by one unit of time. Another method tested was to change the nmut parameter value based on the number of generations without an improvement in the best solution. For this version, the nmut parameter was increased by ten each time the number of generations without improvement exceeded 100. However, the results for this version were not an improvement over the previous results.

Infeasibility

Several methods for handling infeasibility were examined. One experiment involved replacing all infeasible solutions with the best solution when the number of infeasible solutions fell below a certain pre-defined number. The results of the method were satisfactory but not very encouraging because there were no consistent improvements over previous methods. The primary reason for using this criterion for replacement was to avoid certain problems which would be associated with other criterion. For example, if the infeasible solutions were replaced as a result of the generation count, all of the solutions may still be infeasible at that particular generation. In this situation, all solutions in the population would have been replaced with the best solution.

A second experiment allowed the program to run through the first 2000 generations and then took the best solution and replicated it 100 times, effectively replacing the current population with these replicas. The algorithm was then run through another 2000 generations. Again, the results of the attempt did not significantly change from the previous results. However, one interesting discovery was made while running the second experiment. After observing the populations of successive generations, it became apparent an increasing number of solutions became identical to the best solution until the majority of the population had converged to this solution. This would explain why replication of solutions did not produce significantly different results. The population naturally converges and by the 2000th generation most of the solutions are already identical to the best solution. This discovery led to the next experiment which removes duplicate solutions from the population. This is because

once the mating pool is dominated by a single solution, there is insufficient variation in the population to allow improvement over the current best solution.

Duplicate Solutions

This experiment involved removing duplicate solutions from the population. In the first experiment, an entire chromosome was mutated if it was found to be a replica of one which already existed in the population. This version did not show improvement; therefore, a second version was developed. In this version, rather than mutating the entire chromosome of the replicates, two genes on the chromosome were randomly selected to be mutated. This caused a significant increase in the amount of time to run the algorithm which was a key disadvantage of removing duplicate solutions. The program took significantly longer to run since each solution produced must be checked to make sure it does not already exist. Because of this time factor, the population size had to be decreased from 100 to 50. There were no significant improvements in the solution. The only advantage this version displayed over the other versions was for the 50 customer, 6 vehicle problem. After 4000 generations, a feasible solution was found with a value of 580, the best feasible solution which had been found in the experimentation up to this point.

Reverse Paths

Another version of the algorithm was developed to insure against reverse paths. For example, problems can occur if one solution represents a vehicle making a tour, and another solution represents the vehicle making

the same tour only the customers are visited in reverse order from the first one. If these two solutions reproduce with each other, the child will probably not represent an efficient solution even though both parents may have represented good solutions. When the program was run which checked for reverse paths, the results again did not show improvement, and the time factor increased significantly. The population size had to again be reduced to 50 in order to decrease the time to run the algorithm.

Feasibility vs. Infeasibility

Another version of the algorithm was developed as a result of a problem encountered when running the versions discussed in the parameters section. The problem with this earlier version was that occasionally a feasible solution would be found but would be replaced by an infeasible solution having a better evaluation. The new version used the same penalty function which had been used in the earlier version where the infeasibility was raised to a power of 2 and multiplied by the number of vehicles infeasible with respect to time and weight. It solved the feasibility problem by using this penalty function in combination with a procedure which would not allow a feasible solution to have a value which was worse than an infeasible one. The function of this procedure was to subtract the best infeasible solution from the worst feasible solution. If this value was greater than 0, it was an indication that there was at least one infeasible solution with a better evaluation than some feasible solutions. In this case, the value was added to all infeasible solutions to force them to have a worse evaluation than all of the feasible solutions. For the two data sets being tested, this version produced relatively desirable results while keeping both

solutions feasible. This version produced the solutions 597 and 2321 for the 50 and 32 customer problems respectively.

Gene Selection Parameter

Throughout most of the experimentation, the solution quality was affected by the value assigned to the gene selection parameter. The gene selection parameter designates the likelihood of selecting a gene from parent 1. For example, the best results for the 50 customer problem were produced with a gene selection parameter of 50, while the best results for the 32 customer problem were produced with a gene selection parameter of 40. There were several different approaches examined to observe the effects of changes in the gene selection parameter on the results of the algorithm.

The first approach was to allow the two parents to produce several different offspring at different gene selection parameter values, and select the child with the best evaluation to be added to the next generation. For the first version, nine children were produced starting with a gene selection parameter of 10 and at each increment of 10 up to 90. The results of this version were not an improvement and took significantly longer to run. The next attempts were to produce 3 children at increments of 30 in the gene selection parameter, and then 4 children at increments of 20. Neither of these attempts showed improvements either.

Another attempt was to change the gene selection parameter when the number of generations without improvement exceeded a certain number. For the first version, a number between 1 and 100 was randomly generated for the gene selection parameter when the number of generations

without improvement exceeded 100. The next version was to change the gene selection parameter value by 10 each time the number of generations without improvement exceeded 100. Neither of these versions showed improvement.

Summary

In summary, there were some aspects of the experimentation which provided improvements to the algorithm obtained from Bean. Following are a few of these aspects.

1. Penalizing infeasible solutions
2. Changing the number of mutations as the number of generations increases
3. Ensuring all infeasible solutions evaluate better than all infeasible solutions

The gene selection parameter was also observed to be an important factor in determining the solution for a particular problem. However, no consistency was found for using the same gene selection parameter over a number of different problems.

CHAPTER 3
ANALYSIS

The purpose of this chapter is to present the final version of the genetic algorithm. This version was selected because it demonstrated a better overall performance than any of the other versions on the VRP problems used for testing. A summation of these testing methods was presented in chapter 2. First, a discussion is presented describing alternate methods of representing various aspects of the GA and why certain methods seem to perform better for our purposes. A few of the issues addressed are parameter settings, initial population, elitism, and crossover methods. Some causes of premature convergence are also given, as well as proposed methods of reducing the probability of its occurrence. Next, the results are presented for the selected data sets and are compared to the results from the literature. The last section provides a discussion of future direction for research.

Parameter Settings

The settings of the parameter values appear to have a great influence on the genetic algorithm for the VRP. Parameters that are commonly known to have significant effects on the outcome of the algorithm are population size, crossover rate and mutation rate [Schaffer, et al 1989]. In addition to these, an additional parameter of interest in this study was the gene selection parameter. For the 32 customer, 3 vehicle problem with both time and weight constraints, the best solution found with the gene

selection parameter set at 50 was 2261, much larger than the current best solution known of 2086. However, using the same algorithm with the gene selection parameter being dropped to 40, the solution improved to 2130, which is within about 2% of the best known solution. Lawrence Davis developed a procedure which evaluates the effectiveness of the parameter settings on a particular problem and changes the parameters accordingly to produce the best results for the problem. According to Davis, "Genetic algorithms are stochastic, and the same parameter settings used on the same problems by the same genetic algorithm generally yield different results. A consequence of this fact is that it can take a tremendous amount of computer time to find good parameter settings across a number of problems." [Davis 1989]. If a method could be developed for determining the best parameter values for a particular problem, the performance of the genetic algorithm should improve significantly.

There is also concern about whether the parameter values should change during the run of the genetic algorithm and what should initiate the change [DeJong 1985]. According to a study by DeJong: "Increasing the population size was shown to reduce the stochastic effects [of random sampling on a finite population] and improve long-term performance at the expense of slower initial response...,and reducing the crossover rate resulted in an overall improvement in performance, suggesting that producing a generation of completely new individuals was too high a sampling rate." [Schaffer, et al 1989].

This led to a set of experiments involving changing the parameter values on the genetic algorithm in order to observe the effects these changes would have on the results of the VRP data sets selected.

Population size was one parameter which, when altered, had a consistent effect on the results. All previous experiments discussed in earlier chapters used a population size of 100. Consistently, for all data sets tested, increasing the population size to 200 gave better results and decreasing the population size to 50 gave worse results than the population size of 100. This seemed to be consistent for small problems, as well as, large problems.

Another parameter tested was the mutation parameter. For the final version of the program, an additional method of incorporating mutation into the genetic algorithm was used along with the method discussed earlier. In the earlier method, a certain number of completely new solutions are randomly generated in each generation. This new method works by performing a count every tenth generation to determine how much the population has converged. For each gene of the best solution, the allele is compared to the corresponding allele on each of the other solutions in the population. Each time an allele is found to be identical to the allele on the best solution, the counter is incremented by 1. If this counter exceeds a certain number (70 was used for this program), then randomly replace a certain number of these genes (20 was used in this case). The top solutions, which were automatically copied into the next generation, were excluded from this random replacement. The objective was to maintain diversity in the population and help to prevent premature convergence. It was observed for the 32 customer problem that after only 30 generations there were 6 genes which were repeated on at least 70 chromosomes, and this number continued to increase until it stabilized in the range between 28 and 32. In addition to this method of mutation, another method was used

which has previously been mentioned. This method involved increasing the number of mutations as the number of generations increased. This method of increasing the number of mutations did not seem to work well in combination with the method of mutating converged alleles. For all of the data sets for which this combination was tested, the results were either worse than the results when the program was run without this change, or were infeasible where the previous results had been feasible.

In experimenting with the number of best solutions copied into the next generation, it appears that there is no consistent effect on the result by increasing or decreasing this number. This is probably because the solutions have such a tendency to converge that they maintain themselves without being copied into the next generation. However, for these data sets the results seemed to be more consistent with this number set at 20, so this is the number used for the experimentation.

Initial Population

Another aspect of the genetic algorithm which significantly affects the final solution is the generation of the initial population. Liepins, et al studied how the initial population affected the results in their experimentation with the crossover method for the TSP. They discovered that by changing the initial population, a 13% to 17% variation was observed with a conventional crossover, and an approximately 8% variation was observed with a greedy crossover [Liepins, et al 1987]. It appears, however, that a randomly generated initial population produces satisfactory results since the population is heterogeneous at the beginning of the algorithm [Davis 1991]. As observed by Liepins, there appears to be

little benefit in seeding the population with locally optimal solutions. In an experiment by Booker, he was able to find better results when all initial solutions were randomly generated [Liepins and Potter 1991]. For the genetic algorithm used in our experiments, the initial population was always randomly generated in order to introduce diversity into the population. For the smaller problems with time and capacity constraints and the larger problems with only capacity constraints, a randomly generated initial population did not seem to present a problem. However, for larger problems with both time and weight constraints, a feasible solution could not be found. A possible solution to this problem would be to place a few feasible solutions in the initial population while still randomly generating most solutions.

Search Space

One of the problems encountered in using genetic algorithms is the size of the search space. The search space here refers to the number of combinations of possible solutions for the given VRP. An example of the problem of a large search space was presented in an article by Cleveland and Smith involving experiments they had performed on scheduling flow shop releases [Cleveland and Smith 1989]. The Hinton and Nowlan Model [Belew 1989] attempts to solve problems with binary solutions and claims an improved solution if learning is combined with evolution. They refer to a problem which has 2^L (where L is the number of genes on the chromosome) possible combinations as a "needle-in-a-haystack" problem and do not feel that the genetic algorithm alone would perform very well. However, by combining the genetic algorithm with learning, the search

space could be narrowed [Belew 1989]. The difficulty of incorporating learning is that it is not always easy to determine which criterion help define a good solution [Belew 1989].

Vehicle routing problems have large search spaces. As an example, for an n customer, m vehicle problem, the possible number of combinations of only selecting which vehicles will visit which customers is m^n. This does not include the large number of sequencing possibilities within each route. In spite of the fact that there is such a large search space, the genetic algorithm seems to produce results which are relatively close to the best known solutions for the smaller problems. However, for the larger problems, the quality of the solutions and the likelihood of finding feasible solutions decrease.

In addition to changing parameter values, there are several other aspects of the genetic algorithm which are believed to have a significant impact on the performance of the GA. Among these are the representation of solutions, the issue of elitism, and methods of crossover. The following are alternate methods of dealing with these aspects which were presented in the literature.

Elitism

Elitism is the idea of preserving the best members of the population by copying them into future generations. An alternative to elitism is to use a "refresh" operator which works by copying the best member of the population to a location other than the current population. This copy is maintained and occasionally brought back into the population [Sirag and Weisser 1987]. It was decided for our experiments to use elitism rather

than this method in order to keep the best members in the population at all times. An alternative method of representing the solution in the Traveling Salesman Problem is by assigning a customer number to each gene. The order of the tour is then the order of the customer numbers on the chromosome [Sirag and Weisser 1987]. During crossover, a crossover point is randomly selected and all the alleles up to this point are copied from parent # 1; the remaining alleles are copied from parent # 2. One problem with this method is the increased amount of time the crossover takes because when copying from parent # 2, each allele must be checked to see if it has already been copied from parent # 1. Since some genes on parent # 2 are skipped (only the ones which have not already been copied from parent # 1 are copied), the resulting chromosome may not be representative of either parent [Sirag and Weisser 1987]. The problems with this method are also increased when dealing with the VRP which has the added requirement of representing which vehicle visits which customers.

Encoding of solutions

The method of encoding solutions used for this GA was the random keys method which was discussed in Chapter 1. This appeared to be the most efficient means of representing VRP solutions since crossovers could be performed without the added task of ensuring that each customer was visited exactly once. This constraint was automatically met with the random keys representation.

Crossover Methods

There is no agreement on best method of crossover. This section will discuss the greedy crossover and the uniform crossover. According to DeJong, the number of crossover points required to produce better solutions seems to increase with the length of the chromosome [DeJong 1985]. Uniform crossover appears to be better than one-point or two-point crossover; even though, in theory, the other two crossover methods should perform better than uniform. The reason for this is the schema survival rate is better for the one and two point crossover. One advantage with uniform crossover is it does not need to be combined with inversion (reversing the order of the genes on a segment of the chromosome). This is because alleles which are far apart on the chromosome have an equal chance of staying together on the new chromosome as alleles which are close together [Syswerda 1989].

The greedy crossover is one type of crossover which is a possible area of exploration for future research. This crossover was developed by Liepins, et al [Liepins, et al 1987] and uses the idea of a greedy algorithm which, according to L. Davis is "an optimization algorithm that proceeds through a series of alternatives by making the best decision, as computed locally, at each point in the series." [Davis 1991]. They compared the performance of this crossover method with the conventional crossover on the TSP. This crossover method is a modification of one which Grefenstette developed. It begins by starting the tour with the same city every time. At this point, the shortest edge is selected from the two parents, if a cycle is not introduced. If a cycle is introduced, the edge is selected from the other parent, unless it also causes a cycle. If the choice

of either parents results in a cycle, the tour is extended by a random city. This process is repeated until the tour is completed. The advantage to using the greedy crossover is that it "allows problem specific information to be used in the crossover operation." Greedy genetics seem to perform better when the greedy algorithm being used is powerful, meaning it finds a good solution with only one run of the algorithm. However, conventional genetics performs better when the greedy algorithm is weak [Liepins, et al 1987].

In summation, for this genetic algorithm, the only method of encoding a solution which was used in the experimentation was the random keys representation. The method of elitism used was to copy the top 20 solutions into the next generation. The uniform crossover was preferred over one and two point crossovers. This is because the uniform crossover can produce a greater number of combinations of solutions, providing more diversity within the population. The greedy crossover was not used for any of the experimentation.

Convergence

A common problem with genetic algorithms is premature convergence to a solution that is not optimal. This has been a recurring problem when running our experiments; therefore, this section discusses some of the causes of convergence and possible ways of preventing premature convergence. There are two types of alleles which contribute to this convergence: lost and converged. An allele is referred to as lost if every member of the population has the same value for a particular gene. When this occurs, the possible genotypes are severely restricted. An allele

is said to have converged if at least 95% of the population has the same value for a particular gene. Two possible causes of this convergence is that a "super individual" starts producing too many offspring or, in contrast, the other individuals are not producing enough offspring. One solution to this problem is to keep the population as diverse as possible [Baker 1985]. The mutation operator serves as protection against convergence by helping to keep the population diverse [Goldberg 1988]. An example of this problem is shown by Ackley in a comparison between a genetic algorithm and a hillclimbing algorithm. Over a convex solution space, the genetic algorithm took longer to run primarily because a loss of an allele caused a long run to be necessary. The probability of this occurring was reduced by increasing the mutation rate [Ackley 1985].

One cause of convergence is to focus too much on rapid improvement which can cause premature convergence on the wrong strain by driving out alternative genetic material. A good balance must be found. If performance is not sufficiently emphasized, the best members of the population can be lost [Davis 1987]. The manner in which infeasibility is handled is a very important consideration with respect to convergence. Most work with genetic algorithms has been performed on unconstrained problems. Convergence is a difficulty with using the GA on constrained VRP's [Liepins and Potter 1991]. Under a high infeasibility rate, a feasible solution tends to drive other possibilities out of the population. This is due to the fact that the probability of infeasible members reproducing with each other is continuously decreasing [Davis 1987]. According to Liepins and Potter, there are three methods of dealing with infeasibility:

1. Force feasible solutions into the population by using "specialized recombination operators."
2. Do not allow infeasibility by repeating reproduction until a feasible solution is generated.
3. Use a penalty function for infeasibility.

They found that of these three methods, only the first and third were effective [Liepins and Potter 1991]. Davis dealt with infeasibility for job shop scheduling problems by only allowing feasible solutions by selecting the first legal action available from a list of actions for each work station [Davis 1985]. However, since genetic algorithms function by combining information from all members of the population, infeasible members should remain in the population to reproduce with the feasible members [Richardson, et al 1989].

For our version of the genetic algorithm, infeasible members were allowed, but were characterized with a penalty function so as to give an advantage to feasible members of the population. Two penalty functions were tested to observe their effects on the convergence of the population. The first penalty function involved squaring the amount the solution exceeds the weight limit plus the amount the solution exceeds the time limit and multiplying this by the number of vehicles whose routes are infeasible with respect to weight plus the number infeasible with respect to time. The second penalty function involves multiplying the amount the solution exceeds the weight limit by 0.25 plus the amount the solution exceeds the time limit times 0.25. The larger penalty seemed to work better overall when used in combination with the procedure of keeping a count of duplicate genes and randomly replacing them when necessary in order to

maintain diversity in the population. The only exception to this was the 50 customer, 6 vehicle problem, which produced a better solution with the smaller penalty function. The advantage of the larger penalty was particularly obvious with the larger data sets. It appeared that they need a larger penalty function in order to be driven to feasibility. Although a feasible solution was not found for the larger problems with a time constraint, the solution came closer to feasibility when the larger penalty function was used.

When infeasible members are left in the population, it is common practice to use some penalty function in order to give the feasible members of the population an advantage over the infeasible members. One alternative to the standard procedure of combining the cost function and the penalty function into one is to treat the cost as one objective and treat the penalty as a separate objective [Richardson, et al 1989]. According to Richardson, Palmer, Liepins, and Hilliard , there are four guidelines for designing a penalty function:

1. Penalties which are functions of the distance from feasibility are better performers than those which are merely functions of the number of violated constraints.
2. For a problem having few constraints, penalties which are solely functions of the number of violated constraints are not likely to find solutions.
3. Good penalty functions can be constructed from two quantities, the maximum completion cost and the expected completion cost.
4. Penalties should be close to the expected completion cost, but should not frequently fall below it. The more accurate the penalty, the better will be the solutions found. When the penalty often underestimates the completion cost, then the search may not find a solution [Richardson, et al 1989].

Another cause of convergence is genetic drift. This term refers to one allele winning out over the others even though it has no real significant advantage. Normally with the GA, the problem is convergence causing unequal alleles in the population. However, with genetic drift the problem can be even greater, resulting in the bad alleles surviving instead of the good alleles [Goldberg and Segrest 1987]. There is a theorem which states that the best individuals will increase exponentially in the number of times they reproduce assuming that the population is infinitely large [Goldberg and Richardson 1987]. This is believed to be a cause of genetic drift.

One method of reducing the probability of premature convergence is by incorporating the idea of niche and species into the genetic algorithm. The concept of niche and species comes from the natural definition in which different species have separate niches which are composed of different environmental features. By forcing subpopulations to exist, the probability of convergence is reduced [Goldberg and Richardson 1987]. There are several methods of incorporating this idea into the genetic algorithm. One such method, known as preselection, was developed by Cavicchio (1971). With preselection, the offspring only replaces the parent if it gets a better fitness value than the parent. This maintains diversity by only replacing solutions which are similar to themselves. DeJong (1975) developed the concept known as crowding. Each member of the population is assigned a crowding factor based on its similarity to the other members. When an offspring is produced, it replaces the individual which is most similar to itself in a randomly drawn subpopulation of individuals with the same crowding factor. Goldberg and Richardson introduced the idea of sharing to induce niche and species on members of a population [Goldberg

and Richardson 1987]. The idea of sharing is that solutions receive a reward based on their performance; however, the reward must be shared among all of the similar solutions. Therefore, a solution's reward will be reduced corresponding to the number of similar solutions [Deb and Goldberg 1989]. Goldberg and Richardson demonstrated that a genetic algorithm with sharing maintains subpopulations around different peaks, while without sharing, the population converges to a single peak [Goldberg and Richardson 1987].

In addition to niche and species, there are several other methods which have been introduced for dealing with convergence. One idea presented by Bickel and Bickel is to characterize a population as *converged* if the evaluation of all the solutions is within a certain range. If it is determined that the population has converged by this definition, then a certain percentage are replaced with new solutions [Bickel and Bickel 1987]. Baker proposed three additional methods of solving the problem of premature convergence. The first method is standard selection in which there is a limit to the maximum or the minimum offspring produced by a particular parent. The second method is ranking. With ranking, the rank rather than the value of the solution determine an individual's expected number of offspring [Baker 1985]. The third method, the hybrid method, has two alternatives. The first alternative is to use ranking during periods of rapid convergence and to use standard selection the other times. The second alternative is to change the number in the population in order to reach the desirable percentage involvement (the ratio of the number of the best members in the population to the total number of members in the population) [Baker 1985]. The disadvantage with this second alternative is

that a super individual can still control the population. Even though other individuals are not completely lost, their significance can be greatly reduced because the super individual is still dominant [Baker 1985].

Eshelman and Schaffer proposed a method of preventing premature convergence by preventing incest, using uniform crossover, and removing duplicate solutions from the population. In this method, an evaluation is performed to determine the difference between each of the individuals in the population. This difference is referred to as the "Hamming distance". Incest prevention only allows two individuals to reproduce with each other if their "Hamming distance" is greater than a certain amount. This amount will decrease as the population converges. Eshelman and Schaffer produced successful results with this method. However, they determined that it was not necessary to remove duplicate solutions in combination with incest prevention. This was because when the two procedures were combined, results did not significantly improve, and the run time was increased because of excessive comparisons [Eshelman and Schaffer 1991].

The Final Program

This section will summarize the final version of the genetic algorithm. This version was selected because it seemed to perform better than the other versions of the GA on the majority of the vehicle routing problems used in the experimentation. A copy of this version is presented in Appendix A. Following is a list of the parameters selected:

Number of member in population - 200

Number of generations - 2000

Number of best solutions repeated in next generation - 20

Number of solutions mutated each generation - 2

Gene selection parameter value - 50.

The method of encoding solutions used was the random keys representation. A penalty function was used during evaluation to penalize infeasible solutions. This penalty was calculated by the square of the amount the time limit was exceeded plus the square of the amount the weight limit was exceeded times the number of vehicles not meeting the time constraint plus the number of vehicles not meeting the weight constraint. The method of reproduction was to copy the top 20 solutions to the next generation, mutate two complete solutions, and to produce the remaining 178 of the solutions by uniform crossover. In order to decrease the problem of convergence, a particular gene was mutated for 20 of the chromosomes if more than 70 chromosomes in the population had an identical allele to the best member of the population for that particular gene. This version seemed to perform better overall; however, there were a few exceptions in which a slight modification to this version improved performance on the problem. One exception was the 32 customer, 3 vehicle problem which performed better with a gene selection parameter of 40 rather than 50. Also, the 50 customer problem with and without the time constraint, as well as the 100 customer, 8 vehicle problem without the time constraint performed better with the smaller penalty function. The smaller penalty function was the one in which the amount the constraints were exceeded was multiplied by 0.25.

Results

Table 3.1 presents the problems analyzed. The first column of this table lists the problem number which was assigned to each problem. If the number is followed by "-t", this is an indication that the problem is the same one as the previous problem only without the time constraint. The number in brackets beside the problem number indicates the source from which the best known solution value is reported. The next two columns respectively list the number of customers and the number of vehicles for the corresponding problem. The weight capacity for each vehicle is given in column 4, and the time limit for each vehicle route is given in column 5 (where the dotted lines indicate that the problem has no time constraint). Column 6 lists the stop times at each customer. The capacity ratio in column 7 is calculated by dividing the total amount of weight to be picked up by the total vehicle capacity available.

Table 3.2 presents the results of the genetic algorithm compared with the best known solutions of the problems obtained from the literature. The results from the GA were obtained from running the final version of the GA, which was written in C programming language, on a Sparc II UNIX workstation. The first column lists the problem number from Table 3.1. The best known solution which was obtained from the literature is given in column 2. Column 3 gives the solution obtained using the GA. If the final solution was infeasible, this number includes the penalty. The 4th column lists the amount of time the algorithm took to complete the run. Column 5 gives the actual distance for the problems. If the solution was infeasible, the number in parentheses represents the number of vehicles infeasible with respect to time plus the number infeasible with respect to weight. The last

Table 3.1. Problems used for experimentation.

Problem	No. of Customers	No. of Vehicles	Weight	Time	Stop Time	Capacity Ratio
P1[30]	22	3	4500	-------	10	.76
P2[32]	29	3	4500	-------	10	.94
P3[30]	32	3	38000	1000	20	.86
P3-t [30]	32	3	38000	-------	20	.86
P4[17]	50	6	160	200	10	.80
P4-t [17]	50	5	160	-------	10	.97
P5 [32]	75	11	140	160	10	.88
P5-t [17]	75	10	140	-------	10	.97
P6 [30]	75	14	100	10000	10	.97
P7[17]	100	9	200	230	10	.81
P7-t [17]	100	8	200	-------	10	.91
P8 [17]	100	11	200	1040	90	.82
P8-t [32]	100	10	200	-------	90	.90
P9[30]	100	14	112	10000	10	.92
P10 [32]	120	11	200	720	50	.62
P10-t [32]	120	7	200	-------	50	.98
P11[32]	150	14	200	200	10	.80
P11-t [17]	150	12	200	-------	10	.93
P12[32]	199	18	200	200	10	.88
P12-t[17]	199	17	200	-------	10	.92
P12-t[32]	199	16	200	-------	10	.98

Table 3.2. Comparison of GA results with other methods of solving problems.

Problem	Best Solution	GA Solution	Time	Actual	% GA above best known soln
P1	568.56	569.7	512.5	569.7	0.2
P2	534	548.5	699.3	548.5	2.7
*P3	2086	2130.5	996.1	2130.5	2.1
P3	2086	2261.9	802.2	2261.9	8.4
P3-t	2009.31	2009.3	780.5	2009.3	0.0
*P4	555.43	561.3	1230.5	561.3	1.1
P4	555.43	587.9	1333.0	587.9	5.8
*P4-t	524.61	656.8	1228.1	656.8	25.2
P4-t	524.61	749.2	1236.3	749.2	42.8
P5	909	6210.9	2081.8	1000 (6)	INFEAS
P5-t	836.37	1380.8	2108.6	1380.8	65.1
P6	1042	1722.6	2150.7	1722.6	65.3
P7	865.94	6014.4	52 min	1100 (1)	INFEAS
P7-t	826.14	984.0	51 min	984.0	19.1
P8	866.37	1226.9	52 min	1226.9	41.6
P8-t	819	1254.1	52 min	1254.1	53.1
P9	1113	1697.9	54 min	1697.9	52.6
P10	1545	95344.7	1hr. 5min	1974 (1)	INFEAS
P10-t	1042	2960.0	1hr. 4min	2960.0	184.0
P11	1164	569277.9	1hr. 21 min	1578 (6)	INFEAS
P11-t	1034.90	2256.4	1hr. 15min	2256.4	118.0
P12	1417	4578660.0	2hr. 34min	2280 (8)	INFEAS
P12-t1	1329.29	3537.4	2hr. 30min	3537.4	166.1
P12-t2	1334	4980.0	2hr. 29min	4890 (2)	INFEAS

Notes: *P3 are the results of the 32 customer, 3 vehicle problem with a gene selection parameter of 40 instead of 50. *P4 are the results of the 50 customer problem with the smaller penalty function rather than the larger one.

column presents the percentage which the GA solution was above the best known solution. Refer to Appendix 2 for tables showing which customers are visited by which vehicles.

The problems can be categorized as either evenly distributed or clustered. Problems P4, P5, P6, P7, and P9 consist of customer locations which are evenly distributed over the region. [Noon, et al 1991]. Problems P2 and P3 share aspects of both these two categories. [Noon, et al 1991] However, because of the way the genetic algorithm functions, the organization of the customers should have no effect on the results.

Notice that the GA solution to the first three problems, the 29 customer, 32 customer, and 50 customer problems, are all relatively close to the best known solution, with the exception of the 50 customer problem without the time constraint. The poor results for this problem could be due to the high capacity ratio. Beginning with the 75 customer problems, the genetic algorithm performance becomes progressively worse. The genetic algorithm did not even find feasible solutions to the time constrained problems with 75 customers and greater. This is probably due to the increased size of the search space. As the search space size increases, it becomes more and more difficult for the genetic algorithm to converge to the optimal solution. In some instances, the solutions to these problems were continuing to decrease as the genetic algorithm approached its 2000th generation. Therefore, in some cases, the algorithm was allowed to run for 3000 generations in an attempt to allow the algorithm to complete its convergence. However, this did not significantly improve the results. The solutions continued to decrease a small amount for a few more generations

and then converged to a solution not significantly better than the solution at the 2000th generation.

Proposed Improvements

In the article by Whitley, Starkweather, and Fuquay it was observed that:

> The theory behind genetic algorithms is well developed for problems that can be encoded as a binary string with no order in dependencies. However, many potential applications of genetic algorithms involve complex ordering dependencies similar to those found in the Traveling Salesman Problem [Whitley, et al 1989].

Suh and Gucht list three problems to overcome in making this transformation:

1. Representing the problem effectively.
2. Recombination operators are only effective if a heuristic is applied. "Such operators can be found in gradient descent algorithms, hill climbing algorithms, simulated annealing, etc."
3. Premature convergence which is caused by a super individual who overtakes the population or a poor performance by a recombination operator [Suh and Gucht 1987].

According to Grefenstette, in order to apply genetic algorithms to combinatorial optimization problems, some kind of heuristic must be used. He used a heuristic crossover operator which proved to be more effective than the standard genetic algorithm [Suh and Gucht 1987].

Suh and Gucht introduced a method in which two operators were used. The first operator is used to select two parents. A random city is then selected for the beginning of the offspring tour. Subsequent genes are selected one at a time from the parent which will produce the shortest path.

The problem is the paths may still be crossed. This problem is solved using the second operator, the 2-opt operator. This operator randomly selects 2 edges (i_1,j_1) and (i_2,j_2). If $ED(i_1,j_1) + ED(i_2,j_2) > ED(i_1,j_2) + ED(i_2,j_1)$, replace the edges with (i_1,j_2) and (i_2,j_1) (where ED is the Euclidean Distance) [Suh and Gucht 1987]. They were able to produce better results with the 2-opt operator than without it [Suh and Gucht 1987]. "It turned out that the selection of a natural representation and the selection of heuristically motivated recombination operators is critical in the design of robust genetic algorithms for such problems." [Suh and Gucht 1987].

There are several methods which have been proposed to improve the standard genetic algorithm. One method is hybridization of another optimization algorithm with the genetic algorithm. This method can combine the positive features of the other algorithm, such as the encoding technique, with the best features of the genetic algorithm, crossover and mutation [Davis 1991].

Another method is to combine simulated annealing with crossover, mutation, and inversion by using a temperature parameter to control diversity in the population [Sirag and Weisser 1987]. Simulated annealing uses a single individual which is given some amount of energy (high for inefficient solutions, low for efficient ones). When a new solution is generated, it will replace the current solution based on some probability. This probability is assigned according to the amount of energy the new solution has compared to the current one [Sirag and Weisser 1987]. The way this temperature parameter would work with the genetic algorithm would be to select genes from the first parent until the temperature is exceeded, then switch to the second parent until the temperature is

exceeded again. This would work similarly for inversion and mutation. The temperature should start out high and drop fairly rapidly to a medium temperature, then drop slowly to a low temperature [Sirag and Weisser 1987].

One additional possibility for improving the results of the genetic algorithm is to run them in parallel. The idea of Parallel Genetic Algorithms is that instead of having one large population, have several smaller subpopulations reproducing in parallel. At the end of each generation, each subpopulation sends the best individual in its population to the other subpopulations. There are different methods of selecting which individuals are to be replaced by these new members. Among these are replacing randomly, replacing the worst solution, or replacing the solution which is most like the new one. It is undetermined at this time whether selection of the individual based on subpopulation performance rather than population as a whole speeds up or slows down convergence. The advantage of this method is that a large population size is enabled without the unreasonable amount of time that it would take with a sequential genetic algorithm [Pettey, et al 1987].

Conclusion

Based on our research, the genetic algorithm seemed to perform well on problems with 50 or fewer customers. As the number of customers increased, the results became progressively worse and the likelihood of finding a feasible solution also decreased. Also, there was not a version found which would consistently give the best solution for all problems

tested. Following are some thoughts Goldberg and DeJong have expressed involving the inconsistency of genetic algorithms.

The idea of genetics is to be robust over a large domain, not to achieve peak performance. Goldberg says, "When we change a genetic algorithm to work better on a particular problem, we may have some success in jazzing things up on that problem, but when we turn around and try to use those operators elsewhere, we are likely to be disappointed." [Goldberg 1989]. DeJong also believes that evolutionary systems are not meant to be function optimizers and says,

> "... one shouldn't be surprised that: 1) the best individual encountered so far may not even survive into the next generation, 2) that the population itself seldom converges to a global (or even local) optima, or 3) that the ability of GA's to produce a steady stream of offspring that are better than any seen so far can vary from quite impressive to dismal." [DeJong 1985].

Perhaps future research will enable a wider range of vehicle routing problems to be solved closer to optimality with the genetic algorithm. Using the genetic algorithm in combination with some of the ideas presented in the previous section such as a heuristic, simulated annealing, or parallel genetic algorithms could help to improve the results of the GA. One of the major obstacles to overcome seems to be premature convergence of the population. If this problem could be solved, maybe the genetic algorithm would consistently give near optimal results.

References

1. Ackley, David H., "A Connectionist Algorithm for Genetic Search," Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, 1985.

2. Baker, James Edward, "Adaptive Selection Methods for Genetic Algorithms," Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, 1985.

3. Bean, James C., "Genetics and Random Keys for Sequencing and Optimization," Department of Industrial and Operations Engineering, Technical Report No. 92-43, University of Michigan, June 1992.

4. Belew, Richard K., "When Both Individuals and Populations Search: Adding Simple Learning to the Genetic Algorithm," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

5. Bickel, Arthur S. and Riva Wenig Bickel, "Tree Structured Rules in Genetic Algorithms," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

6. Bramlette, Mark F. and Eugene E. Bouchard, "Genetic Algorithms in Parametric Design of Aircraft, " in Handbook of Genetic Algorithms, Lawrence Davis, ed., Van Nostrand Reinhold, 1991.

7. Cleveland, Gary A. and Stephen F. Smith, "Using Genetic Algorithms to Schedule Flow Shop Releases," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

8. Davis, Lawrence, "Adapting Operator Probabilities in Genetic Algorithms," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

9. Davis, Lawrence, Genetic Algorithms and Simulated Annealing, Morgan Kaufmann Publishers, 1987.

10. Davis, Lawrence, Handbook of Genetic Algorithms, Van Nostrand Reinhold, 1991.

11. Davis, Lawrence, "Job Shop Scheduling with Genetic Algorithms," Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, 1985.

12. Deb, Kalyanmoy and David E. Goldberg, "An Investigation of Niche and Species Formation," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

13. DeJong, Kenneth, "Genetic Algorithms: A 10 Year Perspective," Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, 1985.

14. Eshelman, Larry J., Richard A. Caruana and J. David Schaffer, "Biases in the Crossover Landscape," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

15. Eshelman, Larry J. and J. David Schaffer, "Preventing Premature Convergence in Genetic Algorithms by Preventing Incest," Proceedings of the Fourth International Conference on Genetic Algorithms, University of California, San Diego, 1991.

16. Fourman, Michael P., "Compaction of Symbolic Layout Using Genetic Algorithms," Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Carnegie-Mellon University, Pittsburgh, 1985.

17. Gendreau, Michel, Alain Hertz and Gilbert Laporte, "A Tabu Search Heuristic for the Vehicle Routing Problem," Publication CRT-777, Centre de recherche sur les transports Universite de Montreal, June 1991.

18. Goldberg, David E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1988.

19. Goldberg, David E., "Zen and the Art of Genetic Algorithms," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

20. Goldberg, David E. and Jon Richardson, "Genetic Algorithms with Sharing for Multimodal Function Optimization," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

21. Goldberg, David E. and Philip Segrest, "Finite Markov Chain Analysis of Genetic Algorithms," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

22. Golden, B. L. and A. A. Asaad, Vehicle Routing: Methods and Studies, North-Holland, 1988.

23. Grefenstette, John J., "Strategy Acquisition with Genetic Algorithms," in Handbook of Genetic Algorithms, Lawrence Davis, ed., Van Nostrand Reinhold, 1991.

24. Hadj-Alouane, Atidel Ben and James C. Bean, "A Genetic Algorithm for the Multiple-Choice Integer Program," Department of Industrial and Operations Engineering Technical Report No. 92-50, University of Michigan, September 1992.

25. Haimovick, M., A. H. G. Rinnooy Kan, L. Stougie, "Analysis of Heuristics for Vehicle Routing Problems," in Vehicle Routing: Methods and Studies, B. L. Golden and A. A. Asaad, eds., North-Holland, 1988.

26. Holland, John H., Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, University of Michigan Press, 1975.

27. Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, The Traveling Salesman Problem, John Wiley and Sons, 1985.

28. Liepins, G. E., M. R. Hilliard, Mark Palmer, and Michael Morrow, "Greedy Genetics," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

29. Liepins, Gunar E. and W. D. Potter, "A Genetic Algorithm to Multiple-Fault Diagnosis," in Handbook of Genetic Algorithms, Lawrence Davis, ed., Van Nostrand Reinhold, 1991.

30. Noon, Charles E., John Mittenthal and Rekha Pillai, "A TSSP+1 Decomposition Approach for the Capacity - Constrained Vehicle Routing Problem," Management Science Program Technical Report No. 37-91-272, University of Tennessee, June 1991.

31. Nygard, Kendall E., Rhonda K. Ficek and Ramesh Sharda, "Genetic Algorithms: Biologically Inspired Search Method Borrows Mechanisms of Inheritance to Find Solutions," OR/MS Today, 28-34, August 1992.

32. Osman, Ibrahim Hassan, "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," in Annals of Operations Research, 1993.

33. Pettey, Chrisila B., Michael R. Leuze and John J. Grefenstette, "A Parallel Genetic Algorithm," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

34. Richardson, Jon T., Mark R. Palmer, Gunar Leipins and Mike Hilliard, "Some Guidelines for Genetic Algorithms with Penalty Functions," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

35. Schaefer, Craig G., "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

36. Schaffer, David J., Richard A. Caruana, Larry J. Eshelman and Rajarshi Das, "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

37. Sirag and Weisser, "Toward a Unified Thermodynamic Genetic Operator," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

38. Suh, Jung Y. and Dirk Van Gucht, "Incorporating Heuristic Information into Genetic Search," Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms, Massachusetts Institute of Technology, Cambridge, 1987.

39. Syswerda, Gilbert, "Uniform Crossover in Genetic Algorithms," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

40. Wayner, Peter, "Genetic Algorithms: Programming Takes a Valuable Tip from Nature," Byte, 361-368, January, 1991.

41. Whitley, Darrell, Timothy Starkweather and D'Ann Fuquay, "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, San Mateo, 1989.

APPENDICES

APPENDIX 1

Appendix 1 consists of the final genetic algorithm which was used for comparison of the genetic algorithm results to the best known solution presented in the literature. The genetic algorithm is written in C programming language.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define maxload 201          /* max number of loads */
#define maxveh 20            /* max number of vehicles */
#define maxpop 200           /* poplulation size */
#define maxcount 2000                /* number of generations to run */

int nrep = 20, nmut = 2, do_out = 0, pf = 10;
int seed,clevel = 50;
float target = -12090.0;
int nload,stop,nveh,maxweight,stoptime,timelimit,vick;
int ii[1000];
void hsort();
float ff[1000];
struct ind {                    /* chromosome array */
        float gene[maxload];
        float val;
     float realval;
     int  infeas;
     int  weight[maxveh];
     float time[maxveh];
        };
struct dist
        {                       /* distance array */
        float node[maxload];
        };
struct ind *pptr,*opptr;
struct dist *dptr;
int xcoor[maxload+1],ycoor[maxload+1],custweight[maxload+1];
int custno[maxload+1];
int count,feascnt;

FILE *outfile;
FILE *test;
FILE *infile;
FILE *testout;
FILE *gengraph;
FILE *geninfs;
int time_passed;
long timestore, time_now;
float bestinfs,worstfs;
int worstchr;
```

```
main()
{
void eval(),repro();
int pick();
float urand();
void readinb(), setpop();
int i,j,stop,k,l,feasfnd,numinfeas;
float avg,z;
pptr = (struct ind *) calloc (sizeof(struct ind),302);
opptr = (struct ind *) calloc (sizeof(struct ind),302);
dptr = (struct dist *) calloc (sizeof(struct dist),302);


readinb();



outfile = fopen("output","w");
testout = fopen("coortest","w");
gengraph = fopen("graph","w");
geninfs = fopen("feasfile","w");
for(seed=2;seed<=2;seed++) {
time_now= clock(&timestore);
srand(seed);

feascnt = 200;
setpop();
if (do_out)
  {
  avg = 0.0;
  for (i=1;i<=maxpop;i++)
        {
        for (j=1;j<=nload;j++)
                printf("%.3d ",(*(pptr+i)).gene[j]);
        printf("%f\n",(*(pptr+i)).val);
        avg += (*(pptr+i)).val;
        }
  printf("%f\n",avg/maxpop);
  }

stop = 0;
count = 0;
while (1-stop)
   {
   count++;
   /* change the number of chromosomes mutated each generation */
   /*if (count == 500)
        {
        nmut = 5;
        }
      if (count == 1000)
```

```
        {
        nmut = 8;
        }*/
      /*if (count == 1500)
        {
        nrep = 14;
        nmut = 8;
        }*/
      feasfnd = 0;
      numinfeas = 0;
      bestinfs = 1000000;
      worstfs = 0;
      worstchr = 0;
      if(count >= maxcount)
         stop = 1;
      for (i=1;i<=maxpop;i++)
        if ((*(pptr+i)).infeas >= 1)
         numinfeas++;
      feascnt = numinfeas;
      fprintf(gengraph,"%d %.4f %d\n",count,(*(pptr+1)).val,numinfeas);
      fprintf(geninfs,"%d %d\n",count,numinfeas);
         if(count == pf*(count/pf))
           {
           /*printf("time = %d\n",clock(&timestore)-time_now);*/
           printf("%d generations, best value found is %.0f",count,
              (*(pptr+1)).val);
           printf("   Number infeasible is %d",numinfeas);
           printf("\n");
           }
         repro();
         if((*(pptr+1)).val <= target) stop = 1;
         if (do_out)
           {
           avg = 0.0;
           for (i=1;i<=maxpop;i++)
             {
             for (j=1;j<=nload;j++)
                printf("%d ",(*(pptr+i)).gene[j]);
             printf("%f\n",(*(pptr+i)).val);
             avg += (*(pptr+i)).val;
             }
           printf("%f\n",avg/maxpop);
           }
         }

time_passed = clock(&timestore)-time_now;
printf("time = %f seconds\n",time_passed/1e6);
eval(1);
i = 1;
printf("Best solution found is %.0f\n ",(*(pptr+1)).val);
printf("Best solution distance is %.0f\n ",(*(pptr+1)).realval);
printf("Vehicle weights and times:\n");
```

```
for (k=1;k<=nveh;k++)
  printf("    Weight[%d] = %d    Time[%d] = %f\n",
k,(*(pptr+1)).weight[k],k,(*(pptr+1)).time[k]);
printf("Infeasibility = %d\n",(*(pptr+1)).infeas);
numinfeas = 0;
for (k=1;k<=maxpop;k++)
  {
  if ((*(pptr+k)).infeas >= 1)
    numinfeas++;
  }
printf("Number infeasible = %d\n",numinfeas);
printf("Generations = %d\n",count);

printf("\n");
for (j=1;j<=nload;j++)
  {
  printf("%.3f ",(*(pptr+i)).gene[j]);
  if (j == 10*(j/10))
    printf("\n");
  }
printf("\n");
printf("%f\n",(*(pptr+i)).val);
i = 1;
fprintf(testout,"Best solution found is %.0f\n ",(*(pptr+1)).val);
fprintf(testout,"Best solution distance is %.0f\n ",(*(pptr+1)).realval);
fprintf(testout,"Vehicle weights and times:\n");
for (k=1;k<=nveh;k++)
  fprintf(testout,"    Weight[%d] = %d    Time[%d] = %f\n",
k,(*(pptr+1)).weight[k],k,(*(pptr+1)).time[k]);
fprintf(testout,"Infeasibility = %d\n",(*(pptr+1)).infeas);
numinfeas = 0;
for (k=1;k<=maxpop;k++)
  {
  if ((*(pptr+k)).infeas >= 1)
    numinfeas++;
  }
fprintf(testout,"Number infeasible = %d\n",numinfeas);
fprintf(testout,"Generations = %d\n",count);

for (j=1;j<=nload;j++)
  {
  fprintf(testout,"%.3f ",(*(pptr+i)).gene[j]);
  if (j==10*(j/10)) fprintf(testout,"\n");
  }
fprintf(testout,"\ntime = %f seconds\n",time_passed/1e6);
fprintf(testout,"\n");
fprintf(testout,"%f\n",(*(pptr+i)).val);
fprintf(testout,"%d %d\n",xcoor[nload+1],ycoor[nload+1]);
for (j=1;j<=nload;j++)
  {
  k = ff[j];
  l = ff[j+1];
```

```
  if (k==l)
    fprintf(testout,"%d %d\n",xcoor[ii[j]],ycoor[ii[j]]);
  else
    {
    fprintf(testout,"%d %d\n",xcoor[ii[j]],ycoor[ii[j]]);
    fprintf(testout,"%d %d\n\n",xcoor[nload+1],ycoor[nload+1]);
    if (l<=nveh)
      {
      fprintf(testout,"%d %d\n",xcoor[nload+1],ycoor[nload+1]);
      }
    }
  }
fprintf(outfile,"time = %f seconds\n",(time_passed/1e6));
fprintf(outfile,"clevel, seed, count, value  %d %d %d %f\n",clevel, seed,
count,(*(pptr+1)).val);
}/*end seed loop*/
fclose(outfile);
fclose(gengraph);
fclose(geninfs);

}

/*****************************************************************
 *          This procedure randomly generates the initial population          *
 *****************************************************************/

void setpop()

{
int i,j;
void eval();
float urand();
int pick();

for (i=1;i<=maxpop;i++)
  {
  for (j=1;j<=nload;j++)
    (*(pptr+i)).gene[j] =  pick(nveh) + urand();
  eval(i);
  }

}

/*****************************************************************
 *          This procedure assigns a value to each solution based on the      *
 *          distance traveled and the penalty                                 *
 *****************************************************************/

void eval(m)
int m;

{
```

```
int i,j,k,l,w[maxveh+1];
float t[maxveh+1],tottime;
float f,overweight,overtime;
tottime = 0;
for (i=1;i<= nload;i++)
   {
   ii[i] = i;
   ff[i] = (*(pptr+m)).gene[i];
   }

hsort(nload);

overweight = 0.0;
overtime = 0.0;
f = 0.0;
for (k=1;k<=nveh;k++)
   w[k] = 0;
for (k=1;k<=nveh;k++)
   t[k] = 0.0;

for(i=1;i<=nload;i++)
   {
   k = ff[i];
   if (i<nload)
      l = ff[i+1];
   else
      l = nveh + 1;
   if(i==1)  /* if leaving depot */
      {
      f = (*(dptr+nload+1)).node[ii[i]];
      t[k] += (*(dptr+nload+1)).node[ii[i]] + stoptime;
      }
   if (k == l)  /* if using same vehicle for next stop */
      {
      f += (*(dptr+ii[i])).node[ii[i+1]];
      t[k] += (*(dptr+ii[i])).node[ii[i+1]] + stoptime;
      }
   else  /* using different vehicle for next stop */
      {
      f += (*(dptr+ii[i])).node[nload+1];
      t[k] += (*(dptr+ii[i])).node[nload+1];
      if (l <= nveh)  /* end of tour */
         {
         f += (*(dptr+nload+1)).node[ii[i+1]];
         t[l] += (*(dptr+nload+1)).node[ii[i+1]] + stoptime;
         }
      }
   w[k]+=custweight[ii[i]];
 }
(*(pptr+m)).infeas = 0;
for (k=1;k<=nveh;k++)
   {
```

```
  if (w[k] > maxweight)
    {
    (*(pptr+m)).infeas ++;
    overweight = overweight + (w[k] - maxweight);
    }
  if (t[k] > timelimit)
    {
    (*(pptr+m)).infeas++;
    overtime = overtime + (t[k] - timelimit);
    }
  }
(*(pptr+m)).val = f + (pow(overweight,2.) + pow(overtime,2.)) *
(*(pptr+m)).infeas;
/*(*(pptr+m)).val = f + (overweight*.25) + (overtime*.25);*/
(*(pptr+m)).realval = f;
if ((*(pptr+m)).infeas > 0)
 if ((*(pptr+m)).val < bestinfs)
   bestinfs = (*(pptr+m)).val;
if ((*(pptr+m)).infeas == 0)
 if ((*(pptr+m)).val > worstfs)
   {
   worstfs = (*(pptr+m)).val;
   worstchr = m;
   }
for (i=1;i<=nveh;i++)
  {
  (*(pptr+m)).weight[i] = w[i];
  (*(pptr+m)).time[i] = t[i];
  }

}

/*****************************************************************/
/*          heapsorts arrays ff[n] and ii[n] in increasing order of ff          */
/*****************************************************************/
void hsort(n)
int n;
{
int l,i,j,ir,rri,stop;
float rrf;
l = n/2 + 1;
ir = n;
stop = 0;
while (1-stop) { /*printf("%d %d %d %d\n",l,ir,i,j);
        for (i=1;i<= nrep;i++) printf("%.0f ",ff[i]);
          printf("\n");*/
        if (l>1)
          {
          l--;
          rrf = ff[l]; rri = ii[l];
          }
        else
```

```
            {
            rrf = ff[ir]; rri = ii[ir];
            ff[ir] = ff[1]; ii[ir] = ii[1];
            ir--;
            if (ir==1)
              {
              ff[1] = rrf; ii[1] = rri;
              stop = 1;
              }
           };

     if (1-stop)
       {
       i = l;
       j = l+l;
      while (j<=ir)
          {
          if (j<ir)
          if (ff[j] < ff[j+1]) j++;
          if (rrf < ff[j])
            {
            ff[i] = ff[j]; ii[i] = ii[j];
            i = j;
            j = j + j;
            }
         else j = ir + 1;
           ff[i] = rrf; ii[i] = rri;
          } /* while */
       }
  }
}

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

int pick(n)
int n;

{
float p;
int p1;

p = rand();
p = p*n/2147483647.0;
p1 = p + 1;
if (p1<1)
   p1 = 1;
if (p1>n)
  p1 = n;
return p1;
}

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

```
float urand()

{
float p;

p = rand();
p = p/2147483647.0;
return p;
}

/*****************************************************************
 *  Reproduction procedure                                        *
 *****************************************************************/

void repro()

{
int i,j,k,l,split,m,stop,n,ncr;
int v,genecnt,bcnt,acnt;
int numrep;
float z;
float addconst,bestgene,bestval;

/* initialize */
for (i=1;i<=maxpop;i++)
   {
   for(j=1;j<=nload;j++)
      (*(opptr+i)).gene[j] = (*(pptr+i)).gene[j];
   (*(opptr+i)).val = (*(pptr+i)).val;
   (*(opptr+i)).realval = (*(pptr+i)).realval;
   (*(opptr+i)).infeas = (*(pptr+i)).infeas;
   for (l=1;l<=nveh;l++)
      {
      (*(opptr+i)).weight[l] = (*(pptr+i)).weight[l];
      (*(opptr+i)).time[l] = (*(pptr+i)).time[l];
      }
   ii[i] = i;
   ff[i] = (*(opptr+i)).val;
   }

hsort(maxpop);

/* replicate top nrep solutions */
for(i=1;i<=nrep;i++)
   {
   for (j=1;j<=nload;j++)
      (*(pptr+i)).gene[j] = (*(opptr+ii[i])).gene[j];
   (*(pptr+i)).val = (*(opptr+ii[i])).val;
   (*(pptr+i)).realval = (*(opptr+ii[i])).realval;
   (*(pptr+i)).infeas = (*(opptr+ii[i])).infeas;
   for (l=1;l<=nveh;l++)
```

```
        {
        (*(pptr+i)).weight[l] = (*(opptr+ii[i])).weight[l];
        (*(pptr+i)).time[l] = (*(opptr+ii[i])).time[l];
        }
    if ((*(pptr+i)).infeas > 0)
      if ((*(pptr+i)).val < bestinfs)
        bestinfs = (*(pptr+i)).val;
    if ((*(pptr+i)).infeas == 0)
      if ((*(pptr+i)).val > worstfs)
        {
         worstfs = (*(pptr+i)).val;
         worstchr = i;
        }
    }
if(count == pf*(count/pf)) {  /* check for duplicate genes */
  for (i=1;i<=nload;i++)
    {
    genecnt = 1;
    bestgene = (*(opptr+ii[1])).gene[i];
    for (l=2;l<=maxpop;l++)
      {
      if ((*(opptr+ii[l])).gene[i] == bestgene)
        {
        genecnt ++;
        }
      }
    if (genecnt > 70)
      {
      for (j=1;j<=20;j++)
        {
        l = pick(maxpop-nrep);
        (*(opptr+ii[l+nrep])).gene[i] = pick(nveh) + urand();
        }
      }
    }
}
vick = 0;
/* mate maxpop-nrep random pairs */
i = nrep;
stop = maxpop - nmut;
while (i < stop)
    {
    i++;
    j = pick(maxpop);
    k = pick(maxpop);
    for(m=1;m<=nload;m++)
      {
      n = pick(100);/*printf("clevel = %d %d\n",clevel,n);*/
      if (n<=clevel)
        {
        (*(pptr+i)).gene[m]=(*(opptr+j)).gene[m];
        (*(pptr+i+1)).gene[m]=(*(opptr+k)).gene[m];
```

```
        }
      else
        {
        (*(pptr+i)).gene[m]=(*(opptr+k)).gene[m];
        (*(pptr+i+1)).gene[m]=(*(opptr+j)).gene[m];
        }
    }
eval(i);
 eval(i+1);
if ((*(pptr+i+1)).val < (*(pptr+i)).val)
   {
   for (m=1;m<=nload;m++)
        (*(pptr+i)).gene[m] = (*(pptr+i+1)).gene[m];
   (*(pptr+i)).val = (*(pptr+i+1)).val;
   (*(pptr+i)).realval = (*(pptr+i+1)).realval;
   (*(pptr+i)).infeas = (*(pptr+i+1)).infeas;
   for (l=1;l<=nveh;l++)
        {
        (*(pptr+i)).weight[l] = (*(pptr+i+1)).weight[l];
        (*(pptr+i)).time[l] = (*(pptr+i+1)).time[l];
        }
    }
}


/* create mutations */
for (i=1;i<=nmut;i++)
   {
   for (j=1;j<=nload;j++)
        (*(pptr+i+maxpop-nmut)).gene[j] = pick(nveh) + urand();
   }


/* evaluate new values and move forward 1 generation */
for(i=maxpop-nmut+1;i<=maxpop;i++)
   eval(i);

if (bestinfs < worstfs)
  {
  vick = 1;
  addconst = worstfs - bestinfs + 1;
  for (i=1;i<=maxpop;i++)
    {
    if ((*(pptr+i)).infeas > 0)
      {
      (*(pptr+i)).val += addconst;
      (*(opptr+i)).val = (*(pptr+i)).val;
      }
    }
  }
}
```

```
/*************************************************************
 * Read in the Data Set                                      *
 *************************************************************/

void readinb()

{
int i,j,k,l;
float diffx;
float diffy;
infile = fopen("vrp.dat","r");
test = fopen("test.tst","w");

fscanf(infile,"%d %d %d %d %d\n",&nload,&maxweight,&nveh,&stoptime,&timelimit);

nload--;
for (i=1;i<=nload;i++)
  fscanf(infile,"%d %d %d %d\n",&xcoor[i],&ycoor[i],&custweight[i],&custno[i]);
fscanf(infile,"%d %d",&xcoor[nload+1],&ycoor[nload+1]);

fclose(infile);

/*    ************************************************************
      ***                 Calculate distance matrix                ***
      ************************************************************ */
 for (i=1;i<=nload+1;i++)
   for (j=1;j<=nload+1;j++)
      {
      diffx = xcoor[i]-xcoor[j];
      diffy = ycoor[i]-ycoor[j];
       (*(dptr+i)).node[j] = sqrt(pow(diffx,2.)+pow(diffy,2.));
      }
fprintf(test,"Loads = %d, Wt Limit = %d, Veh = %d, St Time = %d, Time Limit = %d\n",
  nload,maxweight,nveh,stoptime,timelimit);
for (i=1;i<=nload;i++)
  fprintf(test,"%d %d %d %d\n",xcoor[i],ycoor[i],custweight[i],custno[i]);
fprintf(test,"%d %d\n",xcoor[nload+1],ycoor[nload+1]);
fprintf(test,"\n\n");
for (i = 1;i<=nload+1;i++)
  {
  for (j=1;j<=nload+1;j++)
    fprintf(test,"%.1f",(*(dptr+i)).node[j]);
  fprintf(test,"\n");
  }
fclose(test);
}
```

APPENDIX 2

Appendix 2 displays the vehicle routes of the best solutions produced by the genetic algorithm. These results are displayed for the data sets which were presented in Table 3.1.

Table A2.1. 22 customers, 3 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 18 19 20 22 17 14 15 16 3 2 1 6 0	2730	404.0
2	0 12 11 9 8 5 4 21 7 0	3100	297.8
3	0 10 13 0	4300	87.9

Table A2.2. 29 customers, 3 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 19 26 29 24 25 27 28 15 18 0	3950	326.7
2	0 23 10 11 12 16 13 7 17 9 8 14 21 0	4425	275.3
3	0 22 2 5 4 1 6 3 20 0	4375	236.6

Table A2.3. 32 customers, 3 vehicles, with time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 6 23 30 31 18 29 28 27 19 0	34577	972.5
2	0 22 26 21 14 15 25 16 20 24 9 11 0	37358	895.4
3	0 17 2 4 3 1 5 32 7 8 13 10 12 0	26630	902.6

Table A2.4. 32 customers, 3 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 6 21 14 15 25 16 20 24 26 22 9 11 0	37805	1076.2
2	0 7 8 19 27 28 29 18 31 30 23 0	37414	744.7
3	0 17 2 4 3 1 5 32 13 10 12 0	23346	828.5

Table A2.5. 50 customers, 6 vehicles, with time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 5 49 10 39 33 45 15 44 37 12 0	155	199.1
2	0 2 29 20 35 36 3 32 0	115	168.2
3	0 27 48 8 26 31 28 22 1 0	102	160.4
4	0 11 16 50 21 34 30 9 38 46 0	128	172.9
5	0 18 13 41 40 19 42 17 4 47 0	157	199.1
6	0 6 23 7 43 24 25 14 0	120	161.6

Table A2.6. 50 customers, 5 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 11 2 20 36 35 29 21 30 10 15 37 0	157	251.9
2	0 5 9 34 50 16 32 8 48 27 0	150	195.7
3	0 12 47 18 25 14 6 0	159	121.4
4	0 7 43 24 4 17 44 45 33 39 49 38 46 0	153	287.3
5	0 42 19 40 41 13 23 26 31 28 3 22 1 0	158	300.4

Table A2.7. 75 customers, 11 vehicles, with time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 67 45 48 28 22 62 68 0	140	151.6
2	0 12 31 10 38 65 66 0	137	166.4
3	0 5 47 36 69 21 74 30 0	120	155.6
4	0 33 73 1 63 3 44 40 17 0	143	158.6
5	0 39 9 50 18 55 25 32 0	129	171.4
6	0 15 20 70 60 71 37 0	71	155.6
7	0 53 8 46 34 52 27 29 75 0	153	158.3
8	0 51 16 49 24 56 23 6 0	114	161.8
9	0 26 72 58 11 59 14 0	132	155.6
10	0 2 61 64 42 41 43 0	113	165.2
11	0 7 35 19 54 13 57 4 0	112	150.0

Table A2.8. 75 customers, 10 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 31 55 18 24 2 47 48 0	137	234.5
2	0 74 61 41 56 23 3 32 9 0	140	215.4
3	0 30 71 60 46 59 38 26 0	131	235.1
4	0 68 21 29 27 13 66 65 0	126	205.3
5	0 67 19 54 15 37 45 12 17 0	140	198.1
6	0 10 11 7 57 70 69 28 0	140	233.5
7	0 4 8 35 14 58 72 39 25 0	139	195.2
8	0 75 5 36 20 34 40 51 0	139	193.4
9	0 44 50 49 63 73 22 1 33 6 0	137	206.2
10	0 53 52 62 64 42 43 16 0	135	214.1

Table A2.9. 75 customers, 14 vehicles, without time constraint

Vehicle #	Route of Vehicle	Wt	Time
1	0 30 52 59 8 34 0	100	152.4
2	0 22 61 5 27 67 0	95	148.4
3	0 11 14 46 0	95	99.6
4	0 6 32 53 4 0	99	135.7
5	0 26 12 3 24 23 43 0	96	160.9
6	0 75 2 54 66 0	99	157.9
7	0 33 42 64 74 29 15 0	97	185.4
8	0 25 18 49 28 36 13 19 0	100	247.2
9	0 37 70 71 69 41 51 40 0	96	221.1
10	0 62 9 55 10 65 35 0	99	220.7
11	0 72 31 16 63 73 20 57 0	98	245.1
12	0 7 58 39 50 56 0	100	173.3
13	0 48 47 60 21 1 0	98	165.9
14	0 17 44 38 45 68 0	92	158.1

Table A2.10. 100 customers, 9 vehicles, with time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	54	4	15	43	14	38	44	42	13	0		134	229.7
2	0	58	2	57	87	92	98	85	61	83	46		172	228.1
		48	0											
3	0	52	84	17	45	8	82	18	60	5	93		150	215.6
		59	0											
4	0	31	10	1	50	68	80	55	25	39	56	0	153	226.3
5	0	88	19	49	64	63	90	32	30	70	69		149	227.5
		27	0											
6	0	7	47	36	11	62	20	66	71	65	35		175	300.1
		33	12	0										
7	0	89	6	96	99	16	86	91	100	37	97		193	221.2
		95	94	28	0									
8	0	76	77	3	79	78	34	9	51	81	29		144	223.7
		24	0											
9	0	53	40	21	73	72	74	22	41	75	23		188	228.2
		67	26	0										

Table A2.11. 100 customers, 8 vehicles, without time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	28	76	77	68	80	54	12	26	0			139	142.4
2	0	13	87	37	93	85	16	61	5	60	89	0	196	174.4
3	0	69	70	30	32	20	51	3	29	24	55		196	320.3
		39	67	25	4	53	0							
4	0	21	56	23	75	74	22	41	57	15	43		197	297.5
		38	14	42	97	94	0							
5	0	58	40	72	73	2	95	59	99	96	6	0	139	172.8
6	0	50	79	78	34	35	65	66	71	9	81		200	258.2
		33	1	27	0									
7	0	31	10	90	63	64	11	7	82	46	45		200	341.6
		17	86	44	91	100	98	92	0					
8	0	52	88	62	19	49	36	47	48	8	84		191	241.3
		83	18	0										

Table A2.12. 100 customers, 11 vehicles, with time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	57	55	53	56	58	60	59	40	43	0		180	913.9
2	0	4	2	6	9	12	14	16	11	10	0		160	905.6
3	0	20	22	25	26	8	7	3	5	75	0		160	871.9
4	0	46	51	31	35	32	33	36	34	29	24	0	190	997.7
5	0	41	42	44	45	48	50	49	27	28	21	0	130	971.9
6	0	90	88	98	96	97	100	17	18	30	0		170	952.2
7	0	91	87	86	77	71	70	79	74	65	67	0	180	1033.5
8	0	1	99	95	94	92	93	23	0				130	741.1
9	0	63	81	78	76	73	80	61	64	62	0		170	939.3
10	0	66	69	68	54	72	82	83	84	85	89	0	170	1039.1
11	0	13	15	19	38	39	37	52	47	0			170	860.9

Table A2.13. 100 customers, 10 vehicles, without time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	20	25	26	28	18	17	19	16	10	0		180	908.8
2	0	46	45	44	42	43	23	13	11	9	8	0	160	1008.9
3	0	90	89	85	82	77	78	81	63	0			180	835.7
4	0	67	65	66	59	60	58	56	53	54	55		200	1096.1
		69	0											
5	0	62	74	72	61	41	47	27	24	22	21	0	170	987.0
6	0	49	48	51	50	52	29	30	15	14	12		200	1304.5
		6	2	1	0									
7	0	40	57	68	64	80	79	73	70	71	76	0	160	1068.2
8	0	7	4	96	94	92	93	97	99	75	5	0	180	999.7
9	0	37	38	35	31	91	84	88	83	86	87	0	180	1065.9
10	0	39	34	36	33	32	3	100	95	98	0		200	979.2

Table A2.14. 100 customers, 14 vehicles, without time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	2	41	22	74	77	33	81	51	90	63	0	112	247.0
2	0	95	59	38	43	42	87	0					102	156.5
3	0	18	82	7	24	29	80	12	26	0			87	198.8
4	0	53	54	39	56	1	62	0					98	192.5
5	0	52	36	49	27	68	28	0					112	193.3
6	0	40	21	72	4	50	20	30	0				107	180.5
7	0	31	11	64	32	71	9	0					102	191.3
8	0	6	96	99	93	5	94	0					98	110.6
9	0	97	44	85	70	66	69	0					107	203.9
10	0	92	37	14	16	17	46	47	19	88	0		105	213.9
11	0	58	67	15	57	100	91	61	45	0			105	239.4
12	0	76	3	79	78	34	35	65	10	0			110	200.6
13	0	13	98	86	84	83	89	0					101	140.8
14	0	55	25	23	75	73	60	8	48	0			112	229.0

Table A2.15. 120 customers, 11 vehicles, with time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	6	3	9	8	12	29	34	33	27	24	0	91	711.4
2	0	55	56	60	61	65	45	43	40	59	57		191	1025.6
		62	64	54	68	72	0							
3	0	99	100	98	97	108	5	4	10	15	13		132	684.9
		117	0											
4	0	111	2	1	7	11	14	19	35	26	20	0	117	698.3
5	0	82	84	113	83	90	73	79	71	74	69		117	713.9
		120	0											
6	0	115	21	23	36	31	30	25	22	16	109	0	110	684.7
7	0	70	76	78	77	66	63	58	53	52	0		125	671.7
8	0	17	28	32	44	46	49	47	48	42	95	0	95	714.1
9	0	67	75	80	51	50	41	37	38	39	0		151	674.8
10	0	88	87	96	110	116	104	107	106	102	92		125	718.8
		85	112	86	0									
11	0	105	103	101	93	91	18	118	114	94	89		121	675.8
		81	119	0										

Table A2.16. 120 customers, 7 vehicles, without time constraint

Vehicle #	Route of Vehicle											Wt	Time
1	0	89	46	50	51	63	80	70	111	81	1	196	1222.0
		5	6	17	90	117	88	0					
2	0	95	68	72	98	100	96	93	120	86	18	198	1425.5
		11	53	58	62	47	28	22	31	2	0		
3	0	85	92	43	45	19	8	94	102	106	107	196	1368.0
		104	103	99	114	20	33	36	29	108	0		
4	0	119	23	26	35	32	34	27	3	10	112	199	1462.2
		105	74	69	101	113	118	59	65	110	0		
5	0	84	83	4	9	25	37	38	115	52	56	193	1188.2
		61	60	77	71	116	82	0					
6	0	91	109	30	24	16	21	44	40	64	66	194	1239.7
		76	73	67	97	13	14	7	0				
7	0	48	49	41	42	39	79	75	78	55	54	199	1054.4
		57	12	15	87	0							

Table A2.17. 150 customers, 14 vehicles, with time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	110	25	95	14	55	134	67	13	41	40		191	260.4
		64	87	56	0									
2	0	133	132	98	23	69	114	99	43	86	61		197	260.4
		7	27	0										
3	0	32	59	2	100	126	50	130	30	9	38	0	145	198.7
4	0	11	127	129	29	28	22	120	48	138	0		115	191.6
5	0	60	8	26	113	140	112	57	97	24	96		144	253.2
		58	102	46	0									
6	0	148	88	66	135	143	4	149	68	6	0		153	199.8
7	0	81	1	83	131	128	84	21	79	74	34		185	261.7
		104	39	54	0									
8	0	17	93	19	94	136	111	141	150	109	0		131	192.8
9	0	77	18	142	147	15	52	63	144	103	76	0	169	197.9
10	0	90	105	75	89	117	73	10	49	5	0		146	198.6
11	0	71	122	91	65	42	45	124	106	125	33		172	254.4
		72	123	108	0									
12	0	78	139	47	146	145	137	44	107	92	37		179	194.5
		12	0											
13	0	62	118	16	101	3	82	31	80	51	0		161	196.1
14	0	53	20	35	85	36	115	121	116	70	119	0	147	217.4

Table A2.18. 150 customers, 12 vehicles, without time constraint

Vehicle #	Route of Vehicle											Wt	Time
1	0	5	90	15	107	92	41	88	64	38	104	194	319.6
		30	50	130	83	0							
2	0	49	45	106	10	124	65	93	44	89	39	199	347.1
		71	142	135	143	0							
3	0	51	80	101	128	84	115	2	48	60	81	188	285.2
		27	46	0									
4	0	144	19	13	67	138	20	35	59	146	87	163	324.6
		148	56	0									
5	0	126	100	119	14	25	133	4	149	109	145	161	215.4
		17	0										
6	0	7	43	24	112	131	53	127	129	16	98	192	354.5
		86	96	0									
7	0	1	120	113	140	22	9	117	125	123	103	186	333.9
		108	137	37	63	0							
8	0	68	134	136	139	57	69	8	31	29	21	194	341.7
		79	74	34	76	0							
9	0	42	150	147	102	82	114	99	23	78	91	191	357.8
		72	33	0									
10	0	47	54	3	121	70	28	116	36	85	118	192	294.6
		62	0										
11	0	141	40	94	66	111	18	110	55	52	122	197	319.4
		105	75	73	0								
12	0	11	26	61	132	97	58	95	6	32	77	178	262.5
		12	0										

Table A2.19. 199 customers, 18 vehicles, with time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	188	16	73	147	181	116	23	194	61	0		153	190.7
2	0	157	156	94	121	138	37	88	140	22	190		318	415.5
		185	143	89	137	62	160	183	1	198	0			
3	0	139	176	102	78	177	19	11	180	7	0		138	196.6
4	0	187	32	57	109	39	40	50	129	71	126	0	177	191.1
5	0	173	83	123	178	84	14	167	179	99	58		189	257.3
		26	0											
6	0	159	192	186	141	142	42	158	66	193	0		132	193.5
7	0	168	100	130	151	117	44	106	144	74	0		130	192.4
8	0	86	93	2	120	155	36	21	64	28	101	0	154	196.5
9	0	184	199	113	43	191	195	104	3	81	0		158	199.0
10	0	60	175	46	34	45	59	98	79	13	152	0	167	196.0
11	0	171	166	124	154	8	51	10	75	31	25		264	353.6
		18	146	56	9	0								
12	0	55	145	148	92	135	163	162	164	133	70		194	348.3
		128	27	4	87	0								
13	0	127	125	153	5	48	174	47	82	172	30	0	183	191.7
14	0	33	105	182	49	107	24	63	95	54	112	0	152	195.6
15	0	114	91	68	115	197	136	196	53	67	0		129	195.2
16	0	17	97	131	80	119	52	38	170	165	77		199	257.7
		150	65	0										
17	0	96	6	111	15	20	122	103	29	0			154	197.9
18	0	76	12	169	161	72	118	110	189	85	134		195	301.7
		108	69	132	35	149	0							

Table A2.20. 199 customers, 16 vehicles, without time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	29	21	114	122	47	79	132	98	196	105		200	416.7
		32	163	0										
2	0	156	91	68	101	27	170	48	73	125	153		200	509.3
		19	72	147	0									
3	0	40	49	184	90	16	74	25	38	168	0		197	279.4
4	0	88	102	8	10	100	182	183	199	62	65		200	432.1
		46	169	78	35	0								
5	0	157	154	167	180	187	24	80	189	126	95		200	378.6
		69	60	0										
6	0	185	141	59	121	179	83	81	193	66	139		199	416.2
		87	134	51	0									
7	0	188	127	7	111	110	75	181	118	194	166		199	392.1
		120	2	0										
8	0	144	159	106	198	20	108	50	26	6	162		199	551.3
		18	192	99	9	0								
9	0	130	11	131	186	104	67	173	36	0			200	293.1
10	0	76	71	119	41	64	137	146	135	39	109		201	522.4
		178	58	150	164	0								
11	0	37	124	155	174	94	175	23	136	77	57		197	476.6
		61	128	123	70	4	0							
12	0	34	1	115	143	86	161	165	14	44	63	0	196	353.3
13	0	133	117	55	195	158	5	171	17	56	31		203	423.0
		160	0											
14	0	152	172	82	151	97	33	112	142	22	149		196	422.6
		176	15	191	43	0								
15	0	45	13	197	148	107	96	103	42	53	93		200	483.9
		54	84	0										
16	0	12	85	52	138	28	89	140	30	129	116		199	587.4
		113	190	92	145	3	0							

Table A2.21. 199 customers, 17 vehicles, without time constraint

Vehicle #	Route of Vehicle												Wt	Time
1	0	17	6	114	198	185	147	72	118	134	19		196	338.9
		175	0											
2	0	125	27	171	68	109	97	55	144	74	152	0	174	295.5
3	0	111	197	90	183	104	137	199	116	182	130		168	304.1
		168	0											
4	0	24	73	192	127	87	35	69	8	124	172		200	345.9
		67	0											
5	0	187	106	32	25	92	146	50	180	133	14		187	267.9
		7	0											
6	0	162	132	170	177	98	155	36	138	20	5		197	333.3
		54	33	0										
7	0	66	62	41	42	141	1	158	156	22	136		193	345.4
		57	100	193	105	0								
8	0	93	173	154	167	176	21	43	23	195	142		192	391.5
		91	191	188	12									
9	0	166	48	112	126	150	164	85	84	179	60	0	162	313.4
10	0	38	80	131	65	99	123	13	153	79	37		198	340.7
		190	115	53	194	0								
11	0	51	46	83	157	102	169	16	186	4	0		194	283.4
12	0	110	39	75	165	52	86	101	140	121	94		199	364.5
		30	29	64	28	0								
13	0	81	149	71	119	129	77	178	174	139	89		186	359.3
		184	120	0										
14	0	47	122	82	148	135	145	107	63	15	88		183	364.0
		103	59	0										
15	0	117	159	44	9	3	95	61	113	143	160		195	281.0
		196	2	0										
16	0	76	40	151	96	161	108	128	78	70	26		179	327.7
		181	49	0										
17	0	58	45	34	11	10	31	189	163	56	18	0	183	270.8

VITA

Vickie Dawn Wester was born in Maryville, TN on April 7, 1969. She attended elementary school in Maryville and graduated from Heritage High School in June 1987. She entered Maryville College in August of 1987 and received her Bachelor of Arts degree in Mathematics/ Computer Science with a minor in Biology in May 1991. The following August she entered the University of Tennessee and received her Master of Science degree in Management Science with a minor in Statistics in December 1993.

Vickie is currently employed at Maryville College as a computer programmer. She is also currently maintaining a computer system which she developed at Montgomery Associates in Alcoa, TN.