A GENETIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

By

JIMING WU

Bachelor of Science
Nanjing Agricultural University
Nanjing, People's Republic of China
1982

Master of Science
Nanjing Agricultural University
Nanjing, People's Republic of China
1986

Doctor of Philosophy
Oklahoma State University
Stillwater, Oklahoma
1997

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2000

A GENETIC ALGORITHM FOR THE

TRAVELING SALESMAN

PROBLEM

Thesis Approved:

Jacques E. LaFrance

Thesis Advisor

J Chandler

D. E. Hall

Wayne B. Powell

Dean of the Graduate College

ACKNOWLEDGMENTS

I sincerely appreciate my major advisor, Dr. Jacques E. LaFrance for his excellent supervision, patient guidance and encouragement throughout the course of this study and my graduate training. Sincere appreciation is extended to the members of my advisory committee, Dr. John P. Chandler and Dr. G. E. Hedrick for their guidance, wise comments and suggestions in this study.

Further thanks to Oklahoma State University, Department of Computer Science, for providing the Teaching Assistantship and necessary facilities and resources.

Final thanks to my mother, Jingxian for giving me the unending encouragement and support, and to my wife, Li, my son, Yijun and my older brother and sister for their patience, encouragement, and many sacrifices throughout my graduate studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES

APPENDIX

A Genetic Algorithm for The Traveling Salesman Problem

ABSTRACT

This paper uses a modified Genetic Algorithm (GA) to attack the Traveling Salesman Problem (TSP). Instead of using a traditional GA, Greedy-Crossover, swap mutation and Eshelman's selection method are employed to solve the TSP. The effects of the crossover percentage, mutation rate, population size and number of generations on the final solutions are evaluated. The run time of GA, Bell Labs, and Minimum Spanning Tree (MST) Algorithms are compared. Preliminary results indicate that increasing the crossover percentage can improve the final result while mutation does not show the helpful effect in improving the solution. The results become better as population size enlarges in this system (100-1000). Additionally, the relationship between run time and population size is linear. Re-initialization of the population can prevent the population from convergence but does not improve the solution. The run time of GA is better than that of MST but worse than that of the Bell Labs Algorithm.

Chapter I

Introduction

If a salesman starts from his home and needs to visit every city on his territory list exactly one time and then return home, he could randomly pick a path or tour that covered each city. However, if the salesman wants to save time and energy he will try to find the shortest tour so that he can visit each city once and return home. The shortest path for the trip can be obtained by adding up the distances for each possible path and comparing them. However, as the number of cities to be visited increases, the time required to calculate the shortest path increases as well. The problem of determining the shortest tour has been named the traveling salesman problem (TSP), and is considered one of the classic optimization problems.

The traveling salesman problem has been the subject of mathematicians' interest since 1759 (Lawler et al., 1985). In 1856, Hamilton had developed his "Icosian Calculus" for graph theory and marketed the Icosian Game, where the aim was to finish a Hamiltonian cycle using numbered pegs and a playing board. In 1930, Menger mentioned the "messenger problem" referring to the problem of finding the shortest Hamiltonian path. The term "traveling salesman problem" may have been used for the first time in 1931 or 1932 when A.W. Tucker heard the term from Hassler Whitney of Princeton University (Lawler et al., 1985).

A graph is defined as a finite set of vertices, some pairs of which are connected by edges [i.e., the G=(V, E) notation]. A directed graph (or digraph) is a graph where a

direction is specified for each of the edges in the graph. A cycle in a graph is then defined as a set of vertices of the graph for which it is possible to move from vertex to vertex along edges of the graph. All vertices are encountered exactly once, and the ending vertex is the same as the beginning vertex. If a cycle contains all of the vertices of the graph, it is referred to as Hamiltonian. Thus, the traveling salesman problem may be described as the problem of discovering a Hamiltonian cycle with the shortest length. When calculating length, it is assumed that the weights of the edges used to calculate the length are nonnegative.

The traveling salesman problem is generally classified as a combinatorial optimization problem. The problem size n equates the number of cities to be visited. Since each city is to be visited only one time, the number of possible tours that can be taken is $(n\text{-}1)!$ while the original city is given. Now, if the number of cities to be considered is small, each of the possible paths may be easily checked. However, as the number of city increases, the number of possible solutions becomes very large. For example, if the salesman must visit 6 cities, there are only $5! = 120$ possible tours. However, if the number of cities is increased to 30, the number of possible solutions increases to exceed $8.841 \times 10.^{30}$ Thus, solutions for finding the shortest tour length without checking each and every possible path must be employed if a solution is ever to be found for larger scale traveling salesman problems.

The ultimate goal of TSP research is to find a solution algorithm that gives an optimal solution in a time that has a polynomial variation with the size n of the problem. The best that researchers have been able to achieve, however, is to solve it in a time that

varies exponentially with n. There are several approaches for solving the difficult problems. One approach is to assume that there is no guarantee that an optimal solution will be found in a reasonable amount of time, but there is still an attempt to find the optimal solution and a great deal of time is spent finding it. Another approach is to use an approximation, where time and optimality are traded off, and an approximate solution is accepted. However, when evaluating algorithms for the solution of the TSP, some type of criteria must be developed for comparisons (Lawler et al., 1985)

The problem is said to be polynomial (P) if an algorithm can solve the problem in a time that increases polynomially with the size n of the problem. There also exists a class of problems that can be tested in polynomial time as to whether or not a hypothetical solution to the problem is correct. Problems that can be solved using non-deterministic algorithms are said to be non-deterministic polynomial or NP (Cormen et al., 1990). Finally, there is a NP-complete class of problems. They are considered to be equivalent to each other in the sense so that if a solution to one kind of NP-complete problem were to be found, solutions to all of them could be found. Thus, a problem is called NP-complete if every problem in NP is polynomially reducible to it. However, there are ways of approximating the solutions to these problems with some specifications. Thus, arbitrarily similar solutions may be obtained. The approach is to use approximation algorithms that will find "near-optimal" tours.

In order to evaluate the performance of various algorithms for "solving" the TSP, a common method of comparison is necessary. To compare the results of various heuristics, arbitrarily large TSP's with known optimal solutions may be used. However,

generating these "testable" TSP instances is a very difficult and computationally complex task.

Fractal constructions have been used to generate instances of the traveling salesman problem with known optimal solutions. Since the optimal tours for these TSP instances are known, this will allow the testing, evaluation, and comparison of various heuristics based on their performance for large TSP instances (Moscato and Norman, 1994).

1.1 TSP Formulations

There are several TSP formulations and they are usually symmetric. For any two cities, numbered A and B, the distance from A to B is the same as that from B to A. We need not to distinguish between a tour and its reverse path. The symmetric formulation is defined as given the complete graph $G=(V, E)$ to find a shortest Hamiltonian tour in G. An asymmetric formulation occurs when the length of traveling from city i to city j is not necessarily the same as that of traveling from city j to city i. The asymmetric case can be transformed and solved as a symmetric TSP (Gerhard, 1991).

The other formulations are listed in Table 1. In reality, they are modifications of the general TSP.

Table 1. TSP formulations

	Description	Solution
The Multi-salesmen Problem	Selecting a subset of the salesmen, assigning tours, and ensuring that each city is visited only once with the	Transformed to a symmetric TSP(Gerhard,

	minimum total cost	1991)
Graphical TSP	Seeking the shortest Eulerian cycle in the graph G. Finding a closed path that requires the shortest possible path length for which the salesman visits each city	Transformed to a symmetric TSP (Gerhard, 1991)
The Bottleneck TSP	Finding a tour where the longest edge is as short as possible	Solved as a sequence of TSPs (Gerhard, 1991)
The Prize Collecting TSP	Minimizing a cycle in G containing the node v_0 such that the sum of the edge weights of the cycle minus the sum of the benefits of the nodes of the cycle	
The Vehicle Routing TSP	Finding the minimal number of vehicles needed to visit each of the locations	Similar as multi-salesmen problem
The Euclidean TS Selection Problem	Finding the shortest tour through a subset k of the cities, $k<n$	TSP for a smaller set of cities
Circulant TSP	Finding a minimum weight Hamiltonian cycle in a weighted graph with a circulant distance matrix	Complexity of this problem is not known
Chinese Postman Problem	Finding the shortest closed walk in G containing all edges at least once (i.e., cities may be visited multiple times)	Equivalent to the Graphical TSP formulation
On-line TSP	Minimizing distance traveled or completion time. All of the inputs may not be known in advance	Solutions that are computed from past events
The Angular-Metric TSP	Finding the tour which minimizes the total angle cost for the tour	Related to the cycle-cover problem
The Rural Postman Problem	Finding the shortest closed walk in G containing all edges in F (F, E)	Chinese postman problem if F=E
Maximum Latency TSP	Finding the tour that maximizes the latency l(i). Latency is the distance traveled before first visiting city p_i	

Minimum Latency Problem	Finding the tour that minimizes the latency l(i). Latency is the distance traveled before first visiting city p_i. That is to minimize the average time that a city waits to be visited	
Eulerian Tour Problem	Finding the shortest tour in Eulerian	solved in polynomial time
Hamiltonian Cycle TSP	Deciding if G contains a Hamiltonian cycle. A Hamiltonian cycle exists if and only if the shortest Hamiltonian cycle in K_n has length n	Solving a symmetric TSP(n=|V|)
Bipartite TSP	Finding the optimal bipartite tour starting and ending at a designated blue vertex s and visiting all vertices. A tour is bipartite if every pair of adjacent vertices in it have different colors	
Remote TSP	Finding a subset P of V of cardinality k such that the cost of the minimum TSP tour on G[P] is maximized	
The Geometric Covering Salesman Problem	Finding the shortest tour that intersects all of the neighborhoods(maybe overlap) and returns home	A generalized the Euclidean TSP
The Tree and Tour Cover TSP	Finding a tour for which the travel and purchase costs are minimized	A minimum weight tree and closed walk, NP-hard
K-template TSP	Processing jobs without interruption on a single machine. K unique templates available for use	O(n log n)
Exact TSP	Finding a path of a given exact length	NP-hard

1.2 Applications of the TSP

The traveling salesman problem is an important computational model for a variety of reasons. While the experiment that sends the poor salesman on various routes may or may not be valid, this does not constitute an application that applies to everyday life.

However, the concept behind the TSP forms a major component of more complicated models, such as routing a fleet of vehicles for pickups and deliveries, or more concretely, the sort of problem that a United Parcel Service (UPS) depot might solve on a daily basis. It can also be used to model a number of problems that do not seem to be related (Wilson, and Pawley,1988).

While the TSP is an interesting problem to address, the amount of research that has been and is currently being done to try to find an optimal (or nearly optimal solution) is motivated by more than mathematical curiosity. As mentioned above, there are a number of real world problems that may be formulated as a type of TSP. The typical applications of the traveling salesman problem include computer wiring, cutting wallpaper, job sequencing and scheduling in a manufacturing plant, and clustering of data arrays et al. (Wilson and Pawley 1988).

1.3 Heuristic Approaches

Heuristics for solving the traveling salesman problem are methods whereby the optimal solution to the TSP is sought. While the use of heuristics may not be guaranteed to find the optimal solution at all, or at least not in real time, a large number of them will find the optimal solution for at least certain cases of the traveling salesman problem

1.3.1 Memetic Algorithms

Memetic Algorithms are a population-based approach for heuristic search in optimization problems. These algorithms have shown that they are orders of magnitude faster than traditional Genetic Algorithms for certain problem domains. Basically, they

combine local search heuristics with crossover operators. For this reason, some researchers have viewed them as Hybrid Genetic Algorithms (a hybrid between a local search and a crossover operator). Since they are most suitable for distributed computing systems (including heterogeneous systems, such as those composed by networks of workstations), and parallel computer systems, they have been referred to as Parallel Genetic Algorithms.

The first use of the term Memetic Algorithms in computing literature appeared in 1989 (Moscato and Norman, 1989). This paper discusses a heuristic which uses Simulated Annealing for a local search with a competitive and cooperative game between agents, interspersed with the use of crossover operators.

This method is gaining wide acceptance, especially in well-known combinatorial optimization problems where many instances have been solved to optimality and where other metaheuristics have failed. An open research issue is whether features of the chosen representation chosen has led to characteristics of the objective functions which are efficiently exploited by a memetic approach (Moscato and Michael, 1992).

1.3.2 Tabu Search

Standard search heuristics start with a tour, and in order to optimize the tour, they exchange a number of edges in the current tour with edges in the "neighborhoods" of those being considered, but not in the current tour. After the edges are exchanged, the operation is evaluated by measuring the decrease in tour length. However, these standard search algorithms have limited obtainable performance, due to the size of the

neighborhood. The tabu search method, as suggested by Zachariasen and Dam marks edges as "tabu" when they have been recently visited or are similar to recently visited solutions, and edges are exchanged for the best recent choice chosen from the tabu space (Hertz et al., 1997).

1.3.3 Simulated Annealing

When simulated annealing algorithms are applied to the traveling salesman problem (Davis, 1987), a sequence of tours is constructed. Each step of the sequence is obtained by moving to a different tour (not necessarily a close one). Martin et al. (1991), however, the simulated annealing and local search heuristics can be combined so that a local optimum will be explored. This heuristic will produce large, global changes, thus overcoming local minima. The results show that the modified simulated annealing is able to solve large TSP instances to optimality (Aarts et al., 1997)

1.3.4 Neural Networks

One possible solution for the traveling salesman problem (TSP) is to apply gradient-type neural networks (Hopfield Nets). However, Hopfield Nets (HN) are generally limited to very small TSP instances, usually around 10 cities (Naphade and Tuzun, 1995). Thus, the general HN formulation is not scaleable to practical sized TSP's. In neural optimization, it is essential that valid solutions be stable fixed points of the dynamics; otherwise, the network will not converge to these solutions and cannot possibly find them. Behzad Kamgar-Parsi and Behrooz Kamgar-Parsi prove that the original Hopfield-Tank (HT) formulation of the Traveling Salesman Problem is flawed.

in that none of the valid tours are stable fixed points in the infinite gain limit (Zurada,1992). When the neuron gain is finite, valid tours become only marginally stable. This helps explain the rather poor performance of the HT formulation in finding valid solutions. Behzad Kamgar-Parsi and Behrooz Kamgar-Parsi also analyze the stability of several modified HT formulations, and show that some are indeed correct and effective. The implication of this work is not that the Hopfield network is an inferior alternative for solving combinatorial optimizations. On the contrary, it shows that dynamical stability analysis is a tool that can help the Hopfield network realize its full potential by identifying flaws in a heuristic formulation (Naphade and Tuzun, 1995).

1.4 Approximation Algorithms

As opposed to heuristics which try to find the optimal solution to the traveling salesman problem, approximation algorithms find solutions that are close enough to the optimal solution. The amount of time required to find these approximate solutions can often be predicted.

1.4.1 Approximation Algorithms

Since it is often impractical or impossible to find the optimal solution to a TSP, it is desirable to find solutions that are close to optimal with algorithms that run in polynomial time.

There are two popular algorithms that approximate the optimal tour within a guaranteed multiple of the optimal. These algorithms only work if the problem satisfies the Triangle Inequality. The Triange Inequality is $D_{ij} <= D_{ik} + D_{kj}$, meaning that the

shortest path between two nodes is directly between them. This may not be true in some instances. For example, a direct flight from Birmingham to Indianapolis may be more expensive than a flight from Birmingham to Chicago plus a flight from Chicago to Indianapolis.

The first algorithm (Kruskal, 1956; Prim,1957) generates the Eulerian tour of the graph formed by having two edges for each edge in the Minimum Spanning Tree (MST). The tour is constructed by removing the nodes from the Eulerian tour after they have already appeared in the tour. This tour is guaranteed to be less than or equal to twice the weight of MST, and the weight of MST is less than the weight of the optimal tour.

Christon's algorithm (Christon, 1976)), the second algorithm, is very similar to this. Instead of doubling the edges of MST, it adds the edges of the least matching of the odd weighted nodes to the edges of MST. It then derives the approximate tour from the Eulerian tour. It has the best known theoretical performance, with a guarantee of being within 3/2 of the optimal tour.

Although these algorithms have guaranteed performances, other algorithms can usually perform better with similar running time. A standard approach to generating good tours is to start with a tour T, and make some changes to produce T', which has a shorter length than T. A heuristic which uses this approach and produces good results is the k-opt. The k-opt replaces k edges with other k-edges so that the resulting tour is shorter. The k edges replaced can be chosen pseudo-randomly, or all possible k-tuples of edges can be tried (Lin, 1956; Lin and Kernighan, 1973; Hochbaum and Goldschmidt, 1997).

1.5 Genetic Algorithms and Evolutionary Programs

The genetic algorithm (Whitley and Mathias, 1992) is a model of machine learning which derives its behavior from a metaphor of the processes of evolution in nature. This is done by creating within a machine of a population of individuals represented by chromosomes; in essence, it is a set of character strings that are analogous to the base-4 chromosomes that seen in human DNA. The individuals in the population then go through a process of evolution.

The processes of evolution in nature seem to boil down to different individuals competing for resources in the environment; some are better competitors than others. Those individuals that are better competitors are more likely to survive and propagate their genetic material. In nature, sexual reproduction allows the creation of genetically different offspring that are still of the same general species. At the molecular level, chromosomes exchange "chunks" of genetic information and then separate. This is the recombination operation, which is generally referred to as crossover because of the way that genetic material crosses over from one chromosome to another (Doolittle, 1987).

The crossover operation happens in an environment where the selection of who gets to mate is a function of the fitness of the individual (i.e., how good the individual is at competing in its environment). Some genetic algorithms use a simple function of the fitness measure to select which individuals undergo genetic operations such as crossover or asexual reproduction (the propagation of genetic material unaltered). Other implementations use a model in which certain randomly selected individuals in a subgroup compete and the fittest is selected. The two processes that contribute to

evolution the most are crossover and fitness based selection-reproduction. As it turns out, there is mathematical proof to indicate that the process of fitness proportionate reproduction is, in fact, nearly optimal in some sense (Whitley and Dzubera, 1994)

Therefore, in practice, this genetic model of computation may be implemented by having arrays of bits or characters represent the chromosomes (Louis,1993). Simple bit manipulation operations allow the implementation of crossover, mutation and other operations. Although a substantial amount of research has been performed on variable-length strings and other structures, the majority of work with genetic algorithms is focused on fixed-length character strings.

When the genetic algorithm is implemented, it is usually done in a manner that involves the following cycle: Evaluate the fitness of all of the individuals in the population (Whitley and Dzubera, 1994). Create a new population by performing operations such as crossover, fitness-proportionate reproduction and mutation on the individuals whose fitness has just been measured. Discard the old population and iterate using the new population. One iteration of this loop is referred to as a generation. The first generation (generation 0) of this process operates on a population of randomly generated individuals. From there on, the genetic operations, in concert with the fitness measure, operate to improve the population.

In contrast, evolutionary programming, originally conceived by Lawrence in 1960 (Moscato and Norman,1989:1992), is a stochastic optimization strategy similar to genetic algorithms that instead, places emphasis on the behavioral linkage between parents and their offspring.

For evolutionary programming there is an underlying assumption that a fitness landscape can be characterized in terms of variables, and that there is an optimum solution (or multiple optima) in terms of those variables. For example, if one were trying to find the shortest path in a traveling salesman problem, each solution would be a path. The length of the path could be expressed as a number, which would serve as the solution's fitness. The fitness landscape for this problem could be characterized as a hypersurface proportional to the path lengths in a space of possible paths. The goal would be to find the globally shortest path in that space, or more practically, to find short tours very quickly (Moscato and Norman, 1989).

The basic evolutionary program method involves three steps which are repeated until a threshold for the number of iterations is exceeded or until an adequate solution is obtained:

1. Choose an initial population of trial solutions at random. The number of solutions in a population is highly relevant to the speed of optimization, but no definite answers are available as to how many solutions are appropriate (other than 1) and how many solutions are just wasteful.

2. Each solution is replicated into a new population. Each of these offspring are mutated according to a distribution of mutation types, ranging from minor to extreme with a continuum of mutation types between. The severity of mutation is judged on the basis of the functional change imposed on the parents.

3. Each offspring solution is assessed by computing its fitness. Typically, a

stochastic tournament is held to determine N solutions to be retained for the population of solutions, although this is occasionally performed deterministically. There is no requirement that the population size be held constant or that only a single offspring be generated from each parent.

The first efforts to find nearly optimal solutions to TSPs by using GA are those of Goldberg using Partial Mapped Crossover (Goldberg, 1985) and Grefenstette using Greedy Crossover (Grefenstette, 1985). Davis, Smith, Suh and Van Gucht also tried to solve TSPs with various crossover operators (Davis, 1985; Smith, 1985; Suh, 1985). In this thesis, I compare the performance of a genetic algorithm with a Minimum Spanning Tree (Prime, 1957) and Bell-Labs (Lin, 1965, Lin and Kernighan, 1973) algorithm using TSPLIB Data (Reinelt,1996) and randomly generated data to initialize tours or individuals.

Chapter II

Genetic Algorithm Analysis and Methods

2.1 Principle of Genetic Algorithms (GA)

The basic principles of GA were first designed by John Holland (Holland, 1975,1992). A genetic algorithm works with a population of individual strings (chromosomes), each representing a possible solution to a given problem. Each chromosome is assigned a fitness value according to the result of the fitness function. Highly fit chromosomes are given more opportunities to reproduce and the offspring share features taken from their parents.

Three operators are employed in GA. They are selection, crossover and mutation. Suppose P(i) is the population of chromosomes at generation i, the procedure of a simple GA is as follows:

```
GA(){
     T=0
     Initialize P(i
     Evaluate P(i)   '/evaluate fitness
     While(termination condition false)
          do{
               select P(i+1) from P(i)
               crossover P(i+1)     //combination
               mutation P(i+1)      // combination
               evaluate P(i+1)      //evaluate fitness
               t=t+1
```

}

}

Evaluation of each chromosome determines which is better, based on a fitness function. Selection is a process in which individual chromosomes are copied according to their fitness function value. The traditional GA uses a selection where the probability of selection is proportional to the fitness value (roulette wheel selection); highly fit individuals have a higher probability of being selected for mating to produce the next generation. Recombination includes two operators: crossover and mutation. The traditional crossover operator randomly chooses a crossover site, cuts chromosomes into two pieces, and swaps the tails to produce two offspring chromosomes as shown in following:

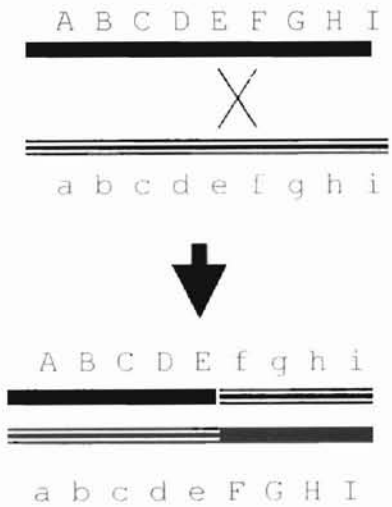
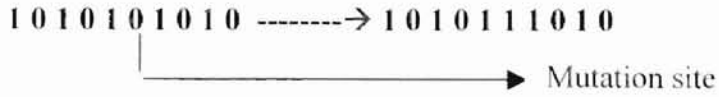


Fig. 1. Traditional Crossover operator

A mutation is a random change of the value of a single (gene) bit. The traditional mutation operator randomly chooses a bit and changes the bit from 1 to 0 or 0 to 1 on the chromosome as presented in the following:



2.2 Modified GA for the TSP

The traditional GA does not fit the TSP since the binary representation can not distinguish the different cities. Furthermore, crossover may produce illegal tour. Modification must be made before implementation.

2.2.1 Fitness Function Definition

The evaluation function for the N cities two-dimensional Euclidean TSP is the sum of the Euclidean distances between every pair of cities in the path. The fitness for the TSP, therefore, is defined as the following:

$$Fitness = \sum_{i=1}^{i=N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

Where, x_i, y_i are the coordinates of city i, and x_0, y_0 equal to x_N, y_N

2.2.2 Representation of Encoding

Some changes must be made to the traditional genetic algorithm to solve TSPs. Binary chromosomes cannot be used to encode the TSP because the TSP is a sequential problem. A path representation where the cities are listed in the order in which they are

visited is used. Assuming there are 6 cities, numbered 0, 1, 2, 3, 4, 5, and if a salesman goes from city 3, through city 0, city 1, city 4, city 2, city 5 and returns back to city 3, the chromosome will be 3, 0, 1, 4, 2 and 5. For the N cities TSP, the representation will initialize the population by randomly assigning 0 to N-1 into N length chromosomes and guaranteeing that each city appears exactly only once on each chromosome. The chromosome must contain unique genes in order to make sure the tour is legal, as will be explained.

2.2.3 Crossover Operator Modification

Because of above representation, the traditional crossover and mutation operators are not suitable for the TSP. The following figure shows how traditional crossover produces illegal offspring. The first child is illegal because city 1 and city 3 appear twice, while city 0 and city 5 do not appear at all. The same problem exits for the second child. City 0 and city 5 appear twice while city 1 and city 3 do not appear.

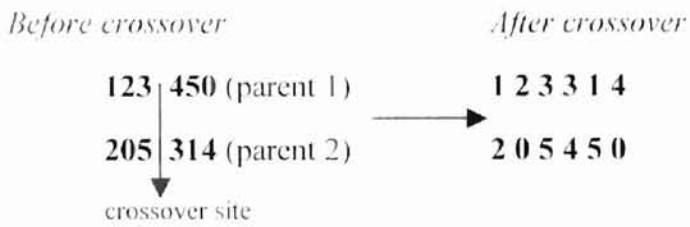


Fig. 2 Production of illegal offspring for TSPs

There are several crossover operators that can be employed for the TSP. In the program implemented for this study, Greedy Crossover is used, which was invented by Grefenstette (Grefenstette, 1985). Greedy crossover selects the first city of one parent,

compares the cities leaving that city in both parents, and chooses the closer one to extend the tour. If one city has already appeared in the tour, we choose the other city. If both cities have already appeared, we randomly select a non-selected city.

For example, if we have two parents numbered 1, 2, 3, 4, 5, 0 and 4, 1, 3, 2, 0, 5. To generate a child using the second parent as the template, we select city 4 (the first city of our template) as the first city of the child-- 4 * * * *. Then we find the edges after city 4 in both parents: (4, 5) and (4, 1) and compare the lengths of these two edges. If the length between city 4 and city 1 is shorter, we select city 1 as the next city of child—4, 1 * * *. Then we find the edges after city 1, (1, 2) and (1, 3). If the length between city 1 and city 2 is shorter, we select city 2 as the next city, resulting in 4, 1, 2 * * *. Then we find the edges after city 2, (2, 3) and (2, 0). If the length between city 2 and city 0 is shorter, we select city 0 as next city 4, 1, 2, 0 * *. The edges after city 0 are (0, 1) and (0, 5). Since city 1 already exists in the child, we select city 5 as the next city, resulting 4, 1, 2, 0, 5 *. Then the edges after city 5 are (5, 0) and (5, 4). We cannot select either one because both city 4 and city 0 already exist in the child. We select a non-selected city, which is city 3. Finally, this procedure produces a legal child numbered 4, 1, 2, 0, 5 and 3. The same approach can be applied to the other child, resulting in the child numbered 1, 2, 0, 5, 4 and 3. After crossover, both offspring contain the legal chromosomes.

2.2.4 Mutation Operator Modification

For the same reason, the traditional Mutation operator cannot be used for the TSP and needs to be modified. For example, if a legal chromosome before mutation is 1, 2, 3, 4, 5 and 0, a mutation occurs at site 4 and the gene has been changed form 4 to 5 which

generates a new linear chromosome numbered 1, 2, 3, 5, 5 and 0. The new chromosome is illegal because city 5 appears twice while city 4 disappears. Instead of using the traditional mutation operator, two genes in one chromosome are selected and *swap* their values. Thus, we still have legal chromosomes after mutation. The following figure shows how the swap works:

1 2 3 4 5 0 ----→1 5 3 4 2 0

Mutation operator (Swap 2 to 5 mutation)

2.2.5 Selection Specification

In the traditional roulette wheel selection, the best individual has the highest probability of survival but does not necessarily survive. In this program, Eshelman's method is employed to guarantee that the best individual will always survive in the next generation (Eshelman, 1991). In this method, if the population size is N, N children are produced by using the roulette wheel selection and greedy-crossover. Then the N parents are combined with the N children, these $2N$ individuals are compared to the averaged fitness value, and the best N individuals are chosen to propagate to the next generation.

2.3 Other Considerations

In order to test the effect of convergence on the result, the program saves the best 10 individuals and reinitializes the rest of the population in 20 and 50 generations. Algorithm performance is observed by testing on randomly produced data sets.

The Minimum Spanning Tree (Prim, 1957) and Bell Labs (Lin, 1965; Lin and Kernighan,1973) algorithms are employed to compare the run time among the three

algorithms using randomly produced data sets. These programs use standard TSP benchmarks whose optimal solutions are known (Reinelt, 1996) to compare the three algorithms. For all problems, the program uses the same crossover and mutation probabilities of 0.95 and 0.01 respectively and the number of generations is 500 except as explained in the figures. All results presented in this paper are averaged over10 runs.

2.4 Coding and Run Environment

The Program including four classes is coded using C++. The parent class TSP produces a tour and holds the tour coordinate structure and distance matrix. Bell_Lab, CGA and MST are children classes that implement the Bell Labs, Genetic, and Minimum Spanning Tree algorithms, respectively. The program was run on a Pentium II-300 with 96MB RAM and 150MB virtual memory under Windows NT.

Chapter III

Results and Analysis

3.1 Genetic Algorithm Performance

The Genetic Algorithm performance is affected by four parameters besides problem size (number of cities). They are the crossover percentage, mutation percentage, number of generations, and population size.

3.1.1 Crossover Percentage

Figure 3 compares the average performance for different crossover percentages. The travel path lengths are improved as the crossover percentage increases. The travel length becomes constant at generation 174 when the crossover percentage is 85 in this test. However, the travel length approaches the lowest point at the generation 143 while the crossover percentage is 95. The improvement percentages (result length/original length *100%) are 15.9% and 15.3% for the crossover percentages of 85 and 95, respectively. The improvement percentage is 15.7% for the crossover percentage of 90.

3.1.2 Mutation Percentage

While crossover can improve the solutions of Genetic Algorithm, mutation seems to prevent the convergence of the population although it does not improve the solution (Figure 4). A mutation is a permanent, inherited change in the structure of a gene (i.e., a change in the sequence of travel path). Mutation genes frequently have deleterious effects on fitness (Doolittle, 1987). The program's results indicate a similar trend in

natural mutation in the biological populations. Improvement percentages are 15.2%, 15.4% and 15.8% for the mutation rate of 1%, 2% and 3%, respectively. The travel lengths reach a constant at generations 123, 129, 137 and 149 for mutation rates of 0%, 1%, 2% and 3%, respectively, in this program. The results indicate that mutation increases the search time and delays the convergence of population. Mutation affects the equilibrium of a population, and thereafter, the fitness. This is a common phenomenon in the natural population.

3.1.3 Population Size

Population size affects Genetic Algorithm performance on both improvement and generations needed at which the population reaches the best fitness (Figure 5,6). The solution is improved and the generations required reach the final result increases as the population size becomes larger. Figure 5 indicates that the result is not changed after 93 generations when population size is 100 in this program, and its improvement percentage is 23.3% (Figure 6). While population size goes up to 1000, 143 generations are needed to get the final result. The improvement percentage is 16.0% at this point. No more change occurs after 143 generations up to 500 generations.

The relationship between run time and population size looks like linear (Figure 7), and it is statistically significant ($R^2 = 0.9614$, $P<0.05$). The slope is 0.4788 while generation is fixed on 500.

3.1.4 Generation

The generations needed to approach the final results are about 150 while population size, crossover percent and mutation rate are 1000, 95% and 1%, respectively (Figures 3-5). The figures show that the population is likely to converge and individuals

in the population become similar so that no more improvement can be achieved. In order to test this kind of case, the population is re-initialized as described in **2.3**. Figure 8 and figure 9 indicate that re-initialization of the population does prevent the population from converging. The convergence of the population that is re-initialized very 20 generations is postponed 83 generations as compared to the population that has not been re-initialized. However, the final results are not improved when compared to the population that has not been re-initialized in this program (500 generations). On the contrary, the final results from re-initialized population are worse than that of the non-re-initialized populations.

3.1.5 Number of Cities

Genetic Algorithm performance on different numbers of cities are shown in Figure 10 and 11. As Figure 10 shows, population converges very quickly when the problem size is small. The population begins to converge at generation 10 when the number of cities equals 20, and this results in an improvement percentage of 50.6% (Figure 11). When the number of cities is 500, the population converges at generation 143. The percentage of improvement reaches 16.0%. Results indicate that the travel path length is improved as the number of cities increases.

3.2 Comparison of the Run Time among Genetic, Bell Lab, and MST Algorithms

Figure 12 shows the run time of the three algorithms. The relationship between the run time and the number of cities is linear for Bell Lab algorithm (Slope=1.0369, R^2=0.9993). However, the relationships for Genetic and MST algorithms are not linear. Figure 12 indicates that the Genetic and MST algorithms need a shorter amount of time to get results when the number of cities is approximately 400. In contrast, the run time

increases very quickly when number of cities grows for MST. The run time of the Genetic algorithm is less than that of MST when number of cities greater than 400.

The log-log plot shown in Figure 13 indicates that the run time of MST is an approximate value of $N^{2.5467}$, and run time of the Genetic Algorithm is about $N^{1.0843}$.

The improvement percentages of different algorithms show that MST achieved the best results using randomly produced data sets (8.5%), see Figure 14.

3.3 Comparison of Three Algorithms using TSPLIB Data

Although MST achieves the best results when the data sets are randomly produced, solutions are not the best when the data sets are not random (Table 2). The Bell Lab algorithm produces the best solutions for all three data sets. For travel path length, GA results in percentages of 8%, 36% and 31% longer paths than the best result when the problem sizes are 52,150 and 318, respectively. MST does not approach the best result partially due to the different starting points. The starting point is so important for MST that final path lengths will vary dramatically if the starting point is not desirable. In random data sets, well-distributed points are guaranteed in a two-dimension plane. The starting point is not as important as it is in non-randomly fixed data sets.

Table 2. Comparison of solutions among the three algorithms using TSPLIB data

Problem Size	Best solution	GA		Bell Lab		MST	
	Distance	Distance	%*	Distance	%	Distance	%
52	7542	8168	108	7625	101	9490	125
150	26130	35637	136	32853	126	35952	137
318	42029	55116	131	49521	117	61446	146

*Test solution/Best solution × 100%.

Figure 15 shows the optimal solution path of the 52 cities problem. The solution for the Bell Lab Algorithm is similar to the optimal solution except for the area indicated by the arrow (Figure 17). One big path crossing occurs in the GA's solution (Figure 16). Several path crossings can be observed in the MST solution (Figure 18). The path crossing is main source of additional length for traveling salesman problem (Pagberg and Rinaldi, 1987). This also indicates the MST Algorithm gains additional length because of path crossings.

Chapter IV

Conclusions and Future Work

This paper demonstrates that the Genetic Algorithm can be modified to solve the Travel Salesman Problem. Increasing the crossover percentage can improve the final result while mutation is not helpful in improving the result in this test. The solutions become better as the population size enlarges in this system (100-1000). In addition, the relationship between the run time and the population size is linear. Re-initialization of the population can prevent the population from converging, but it does not improve the solution in limited population size and generations. The run time of the genetic algorithm is better than that of the MST, but worse than that of the Bell Lab algorithm.

Although this approach seems feasible, much work remains to be done. The optimal crossover percentage and the effect of re-initializing population on solution in the different number of cities still remain issues. Furthermore, the efficient schemes to prevent population from convergence to a local minimum need to be considered.

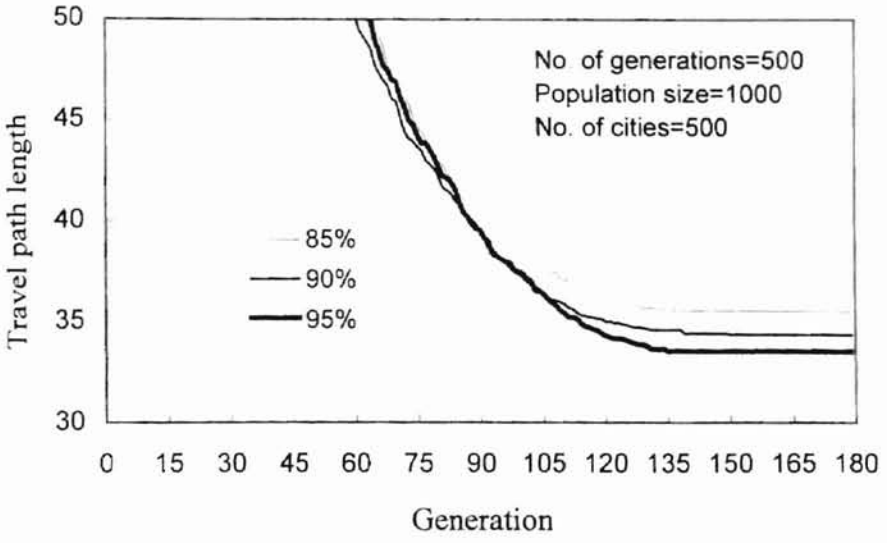


Fig. 3 Genetic Algorithm performance on different crossover percentage

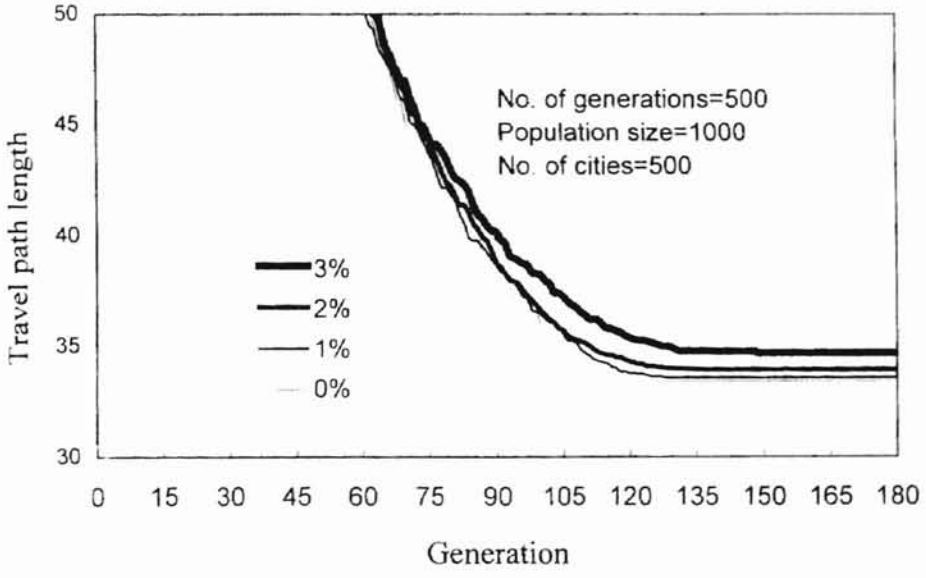


Fig. 4 Genetic Algorithm performance on different mutation percentage

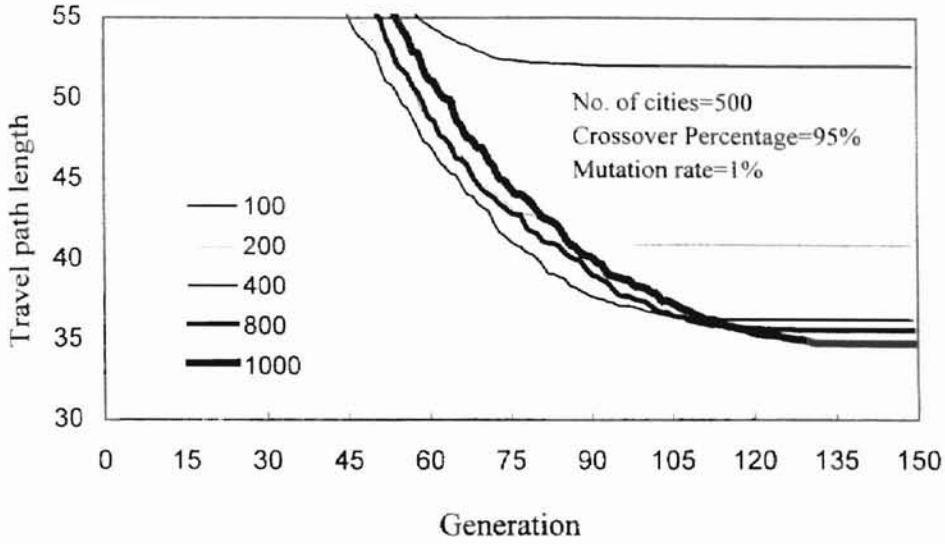


Fig. 5 Genetic Algorithm performance on different population size

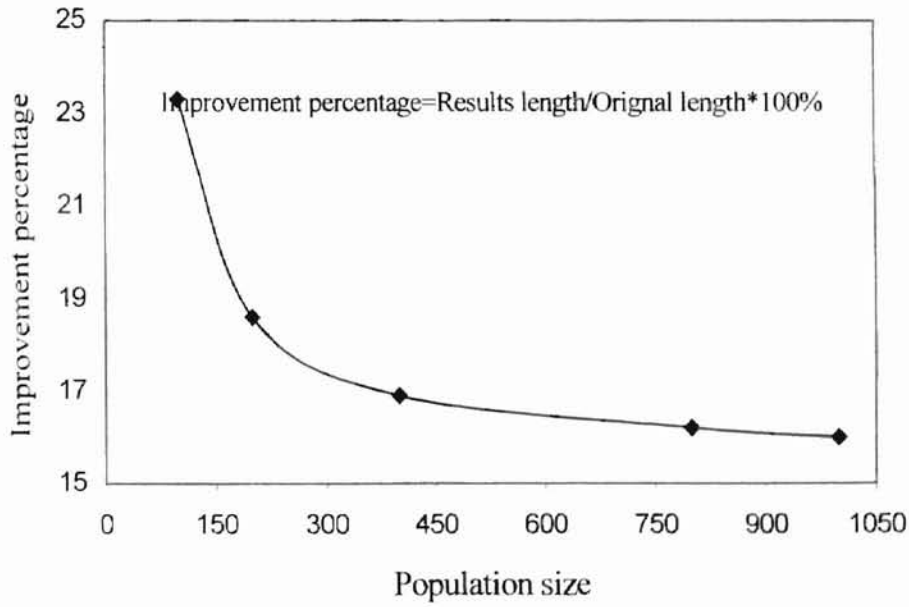


Fig. 6 Genetic Algorithm computing improvement on different population size

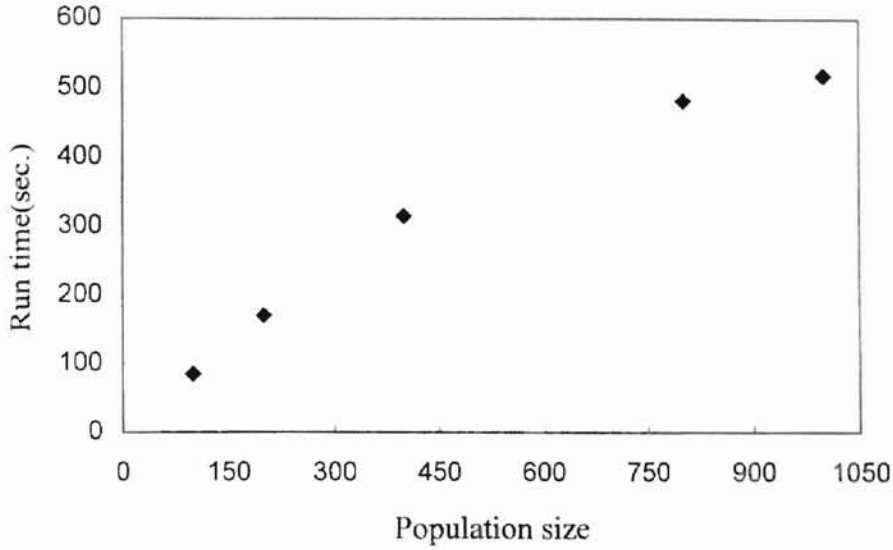


Fig. 7 Genetic Algorithm performance on different population size

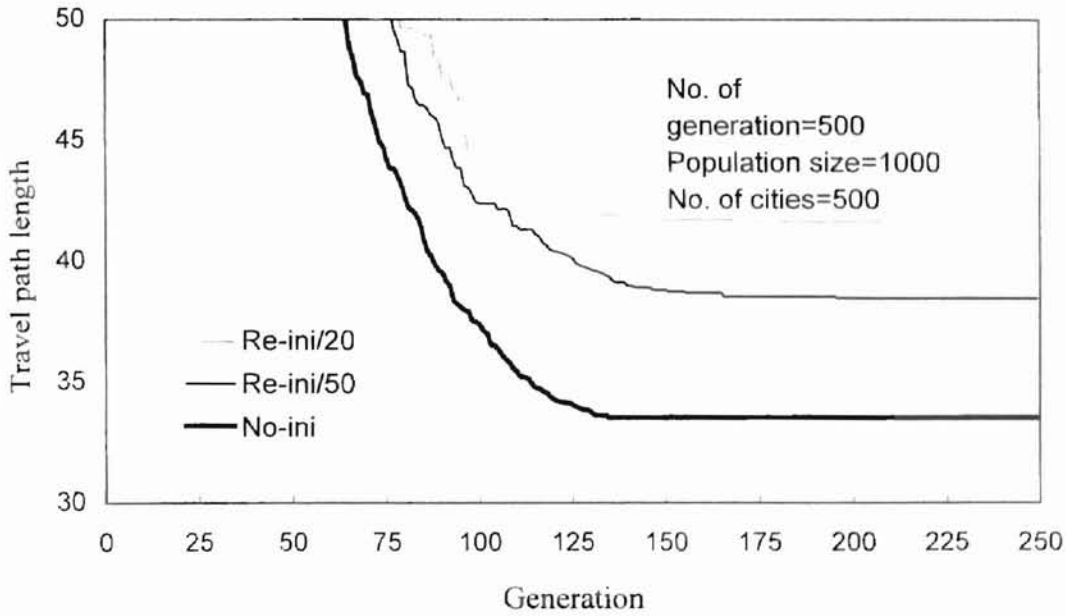


Fig. 8 Travel path length comparison with and without re-initialization of population

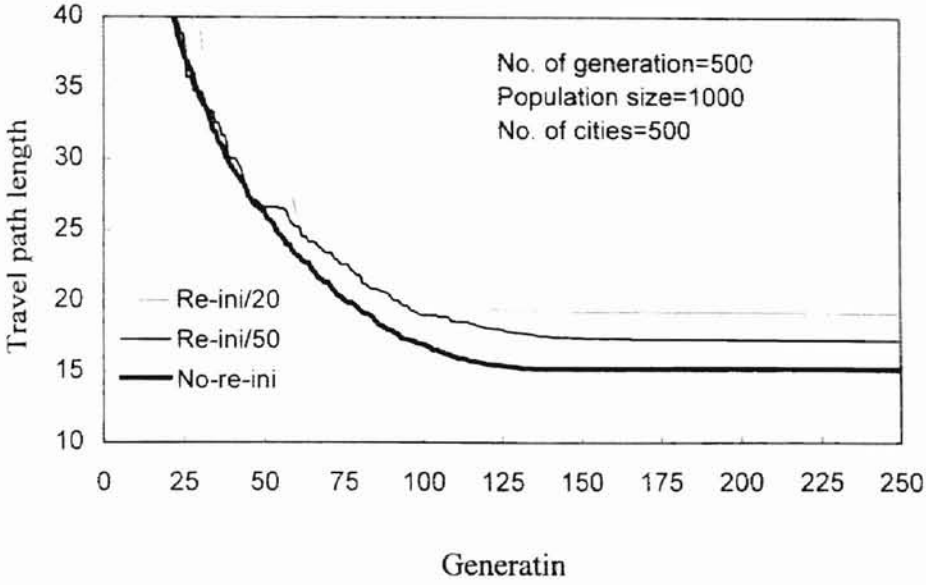


Fig. 9 Improvement comparison with and without re-initialization of population

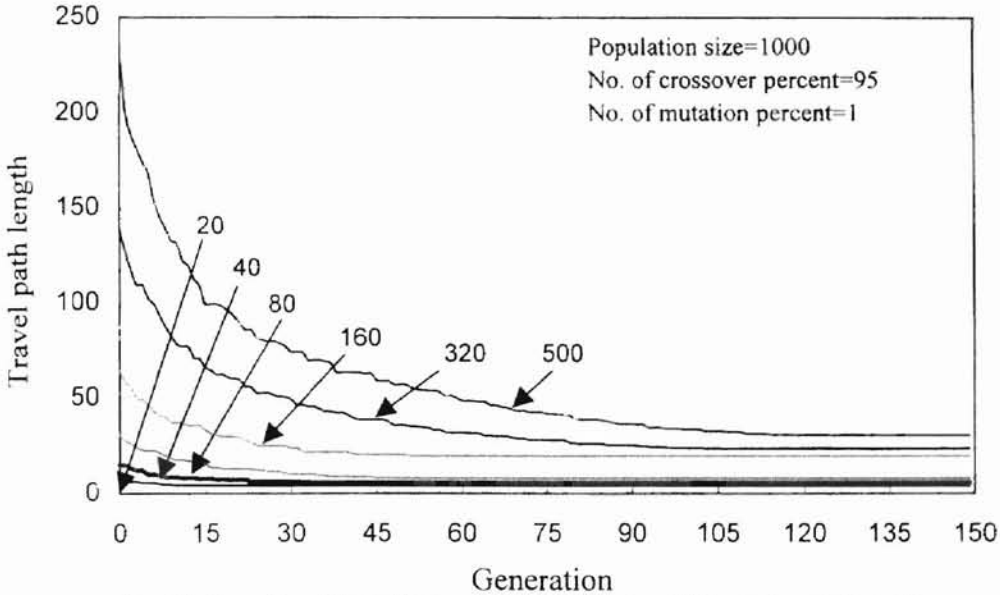


Fig.10 Genetic Algorithm performance on different number of cities

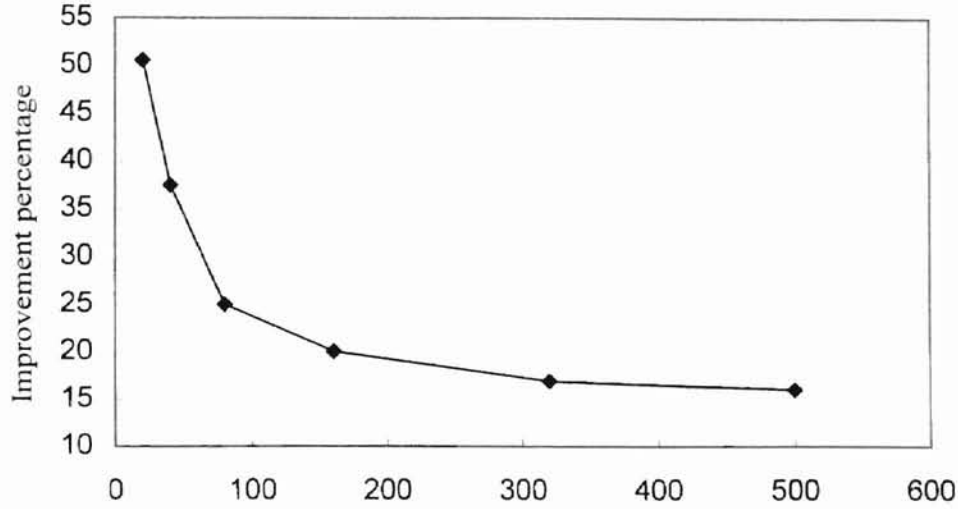


Fig. 11 Genetic Algorithm computing improvement on different number of cities

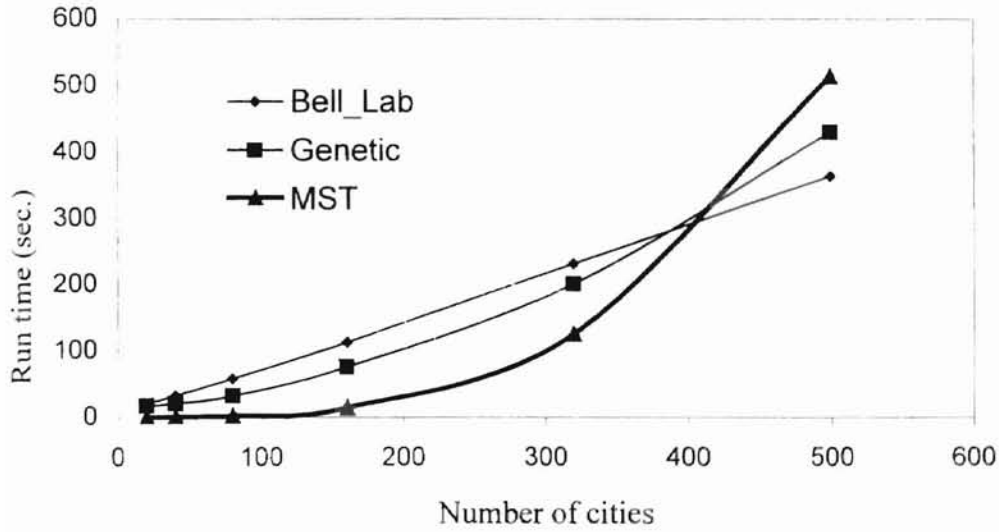


Fig.12 Bell_Lab,Genetic and Minimum spanning tree algorithms performance on different number of cities

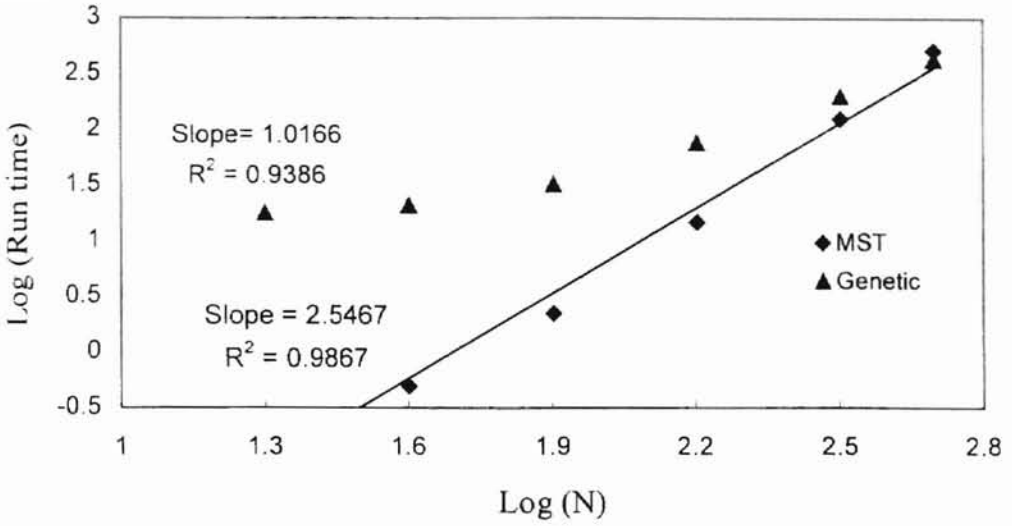


Fig. 13 Relationship between the number of cities (N) and Run time

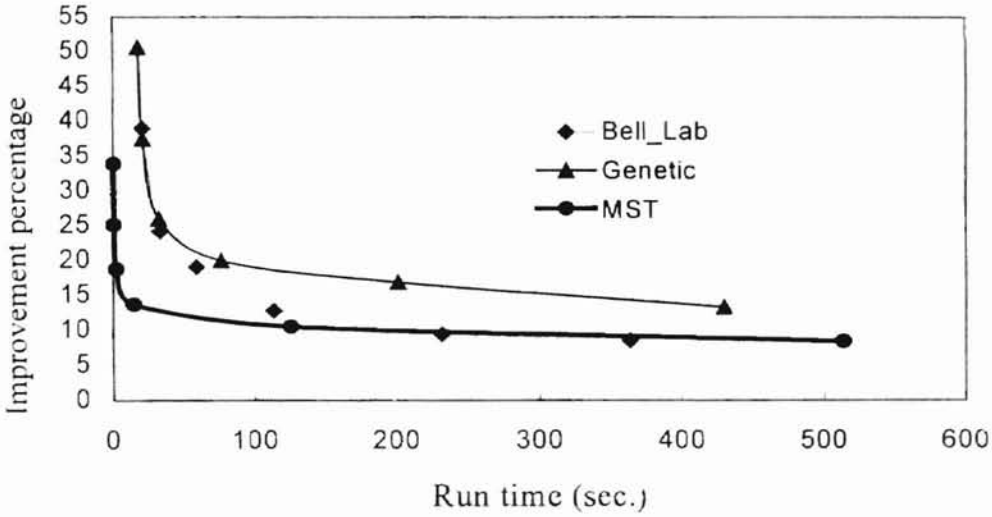


Fig.14 Comparison of improvement for Bell_Lab ,Genetic Algorithm and MST

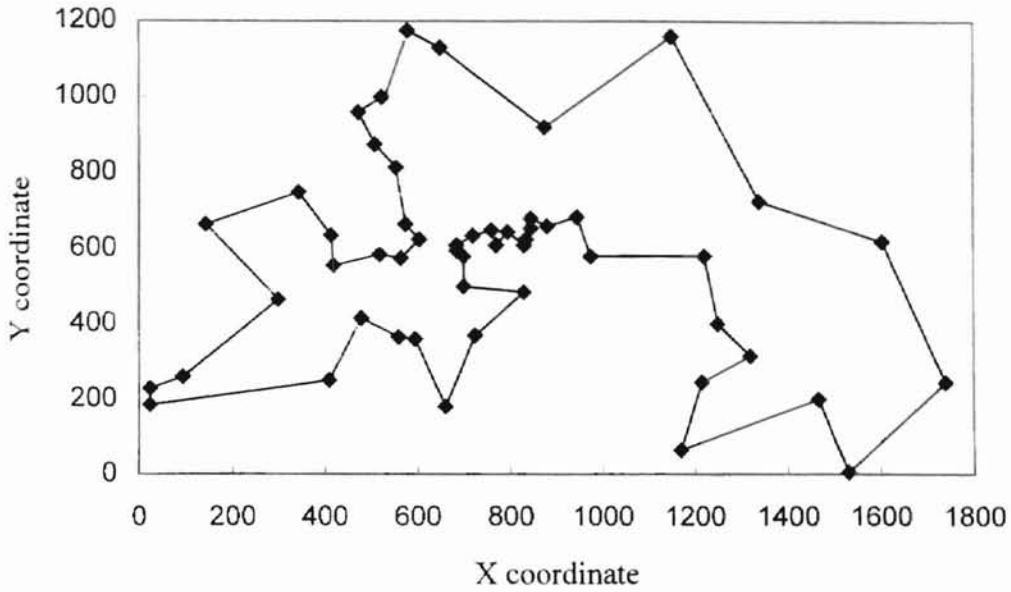


Fig.15 Optimal solution on 52 cities problem (7542)

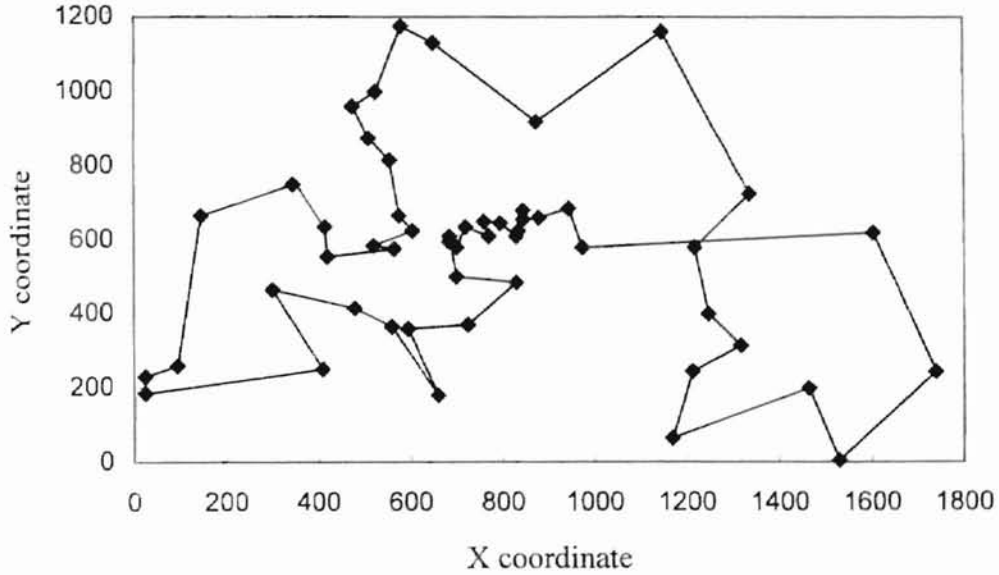


Fig.16 Solution of 52 cities problem with Genetic Algorithm (8168)

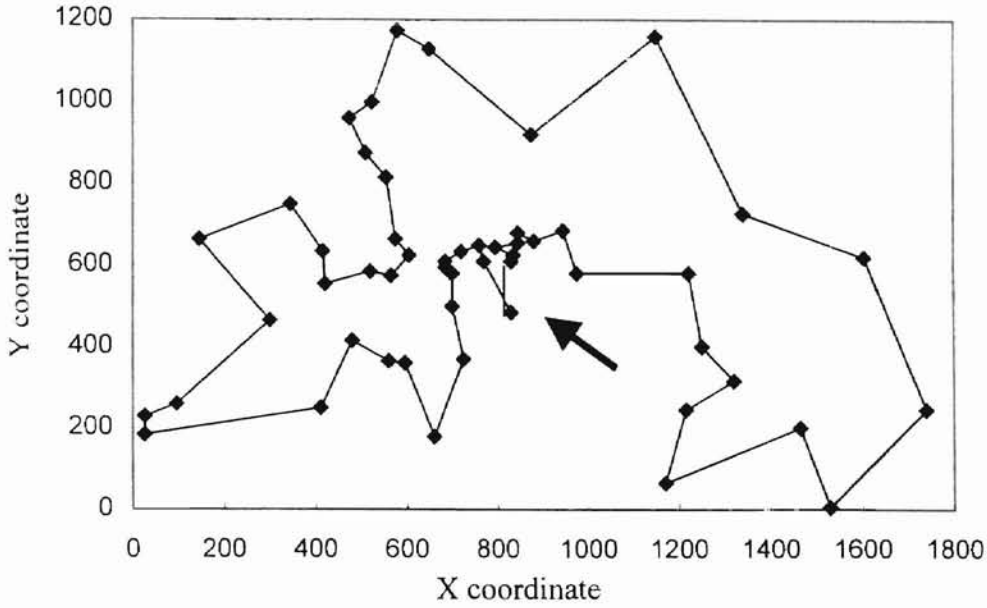


Fig. 17 Solution of 52 cities problem with Bell Lab Algorithm (7625)

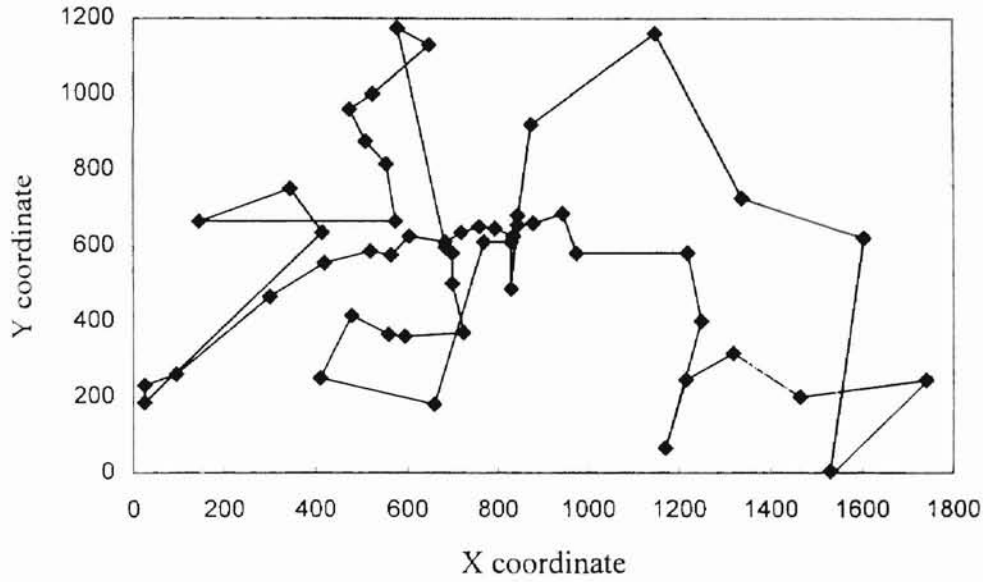


Fig.18 Solution of 52 cities problem with MST Algorithm (9490)

References

1. Aarts, E.H.L., Korst, J.H.M. and Laarhoen, P.J.M. "Simulated annealing". In *Local Search in Combinatorial Optimization.* Aarts, E. and Lenstra, J.K., Editors, John Wiley & Sons Ltd, New York, pp. 91-120, 1997.

2. Cheriton, D. "Finding minimum spanning trees". *SIAM J. Computing* 5(4):724-742,1976.

3. Cormen, T.H., Leiserson, C.E. and Rivest, R.L. *Introduction to Algorithms.* McGraw-Hill Book Company, New York, pp. 916-973, 1990.

4. Davis, L. "Job shop scheduling with genetic algorithms". In *Proceedings of the Second International Conference on Genetic Algorithms.* Lawrence Eribaum Associates, Mahwah, NJ, 1985.

5. Davis, L. *Genetic Algorithms and Simulated Annealing.* Pittman Press, London, 1987

6. Doolittle, D. P. *Population Genetics Basic Principles.* Springer Verlag, New York, 1987.

7. Eshelman, L.J. "The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination". In *Foundations of Genetic Algorithms-1.* Rawlins, G. J. E., Editor, Morgan Kauffman, pp. 265-283,1991.

8. Gerhard, R. *The Traveling Salesman Computational Solutions for TSP Applications.* Springer Verlag, New York, 1991.

9. Goldberg, D. E. and Lingle, R. "Alleles, loci and the traveling salesman problem". In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ, 1985.

10. Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

11. Grefenstette, J., Gopal, R., Rosmaita, R. and Gucht, D. "Genetic algorithms for the traveling salesman problem". In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ, 1985.

12. Hertz, A., Taillard, E. and Werra, D. "Tabu Search". In *Local Search in Combinatorial Optimization*. Aarts, E. and Lenstra, J.K., Editors, John Wiley & Sons Ltd, New York, pp. 121-137, 1997.

13. Hochbaum, D.S. and Goldschmidt, O. "K-edge subgraph problems". *Discrete Applied Math*, 74(2): 159-169,1997.

14. Holland, J.H. *Adaptation In Natural and Artificial Systems*. The University of Michigan Press, Ann Arbour, 1975.

15. Holland, J. H. *Adaptation In Natural and Artificial Systems*. MIT Press, Combridge, 1992.

16. Krushal, J.B.Jr. "On the shortest spanning subtree of a graph and the traveling salesman problem". *Proc. AMS* 7:1, pp. 48-58, 1956.

17. Lawler, E.L., Lenstra, J.K., Rinnooy A.H.G. and Shmoys, D.B. Editors, *The*

Traveling Salesman Problem. John Wiley & Sons, New York, pp. 1-31,1985.

18. Lin, S. "Computer solutions of the traveling salesman problem". *Bell, Syst. Tech. J.* 44:2245, 1965.

19. Lin, S. and Kernighan, B. "An effective heuristic algorithm for the travelling-salesman problem". *Operations Research*, 21(2):498-516, 1973.

20. Louis, S.J. "Genetic algorithms as a computational tool for design". Ph.D. thesis, Indiana University, Indiana University, 1993.

21. Martin, O., Otto, S. and Felten, E. "Large-step mark of chains for the traveling salesman problem". *Complex Systems*, 5(3): 299-326, 1991.

22. Moscato, P. and Norman M.N., "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". *Caltech Concurrent Computation Program*, C3P Report, 1989.

23. Moscato, P. and Norman M.N., "A 'memetic' approach for the traveling salesman problem-Implementation of a computational ecology for combinatorial optimization on message-passing systems". In *Parallel Computing and Transporter Applications*. Valero, M. Onate, E. Jane, M. Larriba and Suarez, B., Editors, IOS Press, Amsterdam,1992.

24. Naphade, K.S. and Tuzun, D. "Initializing the Hopfield-Tank networks for TSP using a convex hull: A computational study". In *Proceedings of the Artificial Neural Networks in Engineering Conference*. V.5, pp399-404, St. Louis, 1995

25. Padberg, M. and Rinaldi, C. "Optimization of a 532-city symmetric traveling

salesman problem by branch and cut". *Operations Research Letters*, 6(1):1-7, 1987.

26. Prim, R.C. "Shortest connection networks and some generalization". *Bell System Technical J.*, 36:1389-1401, 1957

27. Reinelt, G. *TSPLIB*. University of Heidelberg, http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html, 1996.

28. Smith, D. "Bin packing with adaptive search". In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ, 1985.

29. Suh, J. and Gucht, D.Van. "Incorporating heuristic information into genetic search". In *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Eribaum Associates, Mahwah, NJ, 1985.

30. Whitley, D. and Mathias K. "Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem". In *Parallel Problem Solving from Nature-PPSN II*. Manner, R. and Manderick, B., Editors, North Holland-Elsevier, pp. 219-228, 1992.

31. Whitley, D. and Dzubera, J. "Advance Correlation Analysis of Operators for the Traveling Salesman Problems". In *Parallel Problem Solving from Nature-PPSN III*. Davidor, Y., Schwefel, H.P. and Manner, R., Editors, pp. 68-77. Springer-Verlag,1994

32. Wilson, G.V. and Pawley, G.S. "On the stability of the traveling salesman problem algorithm of Hopfiled and Tank". *Biological Cybernetics*, 58:63-70, 1988

33. Zurada, J.M. *Introduction to artificial neural systems*. West Publishing Company,

New York, 1992

APPENDIX

Crossover Function Code List

```
/*************************************************************************/
/* Greedy crossover was suggested by Grefenstte(1985).In "proceedings
/* of the Second International Conference on Genetic Algorithm".
/* Lawrence Eribaum Associates,Mahwah, NJ.
/* This function is created by Jiming Wu,July 10,1999.
/* parameters:    int n--Number of genes on one chromosome
/*                int* AQ and int * BQ--A pair of integer chromosome,
/*                                      each contains N genes
/*                float *DISTANCE--The distance matrix among genes
/* return a pointer of array (after crossover)
/* The function is Free to use.
/*************************************************************************/

int* CGA::greedy_crossover(int n,int *AQ,int *BQ,float *DISTANCE)
{
      int i,k,l=0,m,loci=0;
      int anext;  /* next gene in A chromosome */
      int bnext;  /* next gene in B chromosome */
      float avalue,bvalue;/* Distances of current gene and next gene */
                        /* two chromosome pieces.*/

      int* check=new int[n];  /*check if genes exist in array*/
      int* CQ=new int[n];     /*resulting chromosome for return*/
      for(i=0;i<n;i++)/*initialize all genes not exist in chromosome */
            check[i]=-1;/*if gene exists in chromosome, then value=0 */

      CQ[0]=AQ[0];      /*initialize first gene*/
      check[AQ[0]]=0;   /*use parent AQ as template*/

      do{/* crossover*/
            for(k=0;k<n;k++)/*find the same gene in parent B*/
            {
                  if (CQ[loci]==BQ[k])
                  break;
            }

            for(m=0;m<n;m++)/*find the same gene in parent A*/
            {
```

```
      if (CQ[loci]==AQ[m])
      break;
}

if (k==n-1)/*if it's the last gene,the next is first gene*/
      bnext=BQ[0];
else if (k<(n-1))
      bnext=BQ[k+1];

if (m==n-1)
      anext=AQ[0];
else if (m<(n-1))
      anext=AQ[m+1];

if ((check[bnext]==0) && (check[anext]==0))
{/*if anext and bnext both already exist in CQ */
      /*select one doesn't exist in CQ*/
      while(check[l]==0)
      {
            l++;
      }
      loci++;
      CQ[loci]=l;
      check[l]=0;
}
else if(check[bnext]==0)/*if bnext exists in array*/
{                       /*the next gene is anext*/
      loci++;
      CQ[loci]=anext;
      check[anext]=0;
}
else if(check[anext]==0)/*if anext exists in array*/
{                       /*the next gene is bnext*/
      loci++;
      CQ[loci]=bnext;
      check[bnext]=0;
}
else/*if both not exist in array*/
{     /*select shorter one */
      avalue=DISTANCE[AQ[m]][anext];
      bvalue=DISTANCE[BQ[k]][bnext];
      if (avalue>bvalue)
```

```
                {
                        loci++;
                        CQ[loci]=bnext;
                        check[bnext]=0;
                }
                else
                {
                        loci++;
                        CQ[loci]=anext;
                        check[anext]=0;
                }
        }
    }while(loci<n-1);
    delete check;
    return CQ;
}
```

VITA

JIMING WU

Candidate for the Degree of

Master of Science

Thesis: A GENETIC ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Wuxi, Jiangsu, People's Republic of China, February, 17, 1960, the son of Xiji Wu and Jingxian.

Education: Graduated from Xian High School, Wuxi, Jiangsu, People's Republic of China, July, 1976; received Bachelor of Agronomy Degree from Nanjing Agricultural University, Nanjing, People's Republic of China in July, 1982; received Master of Science Degree in Plant Breeding and Genetics from Nanjing Agricultural University, Nanjing, People's Republic of China in January, 1986; received Doctor of Philosophy Degree in Crop Science from Oklahoma State University, Stillwater, Oklahoma in December, 1997. Completed requirements for the Master of Computer Science at Oklahoma State University, Stillwater, Oklahoma, in May, 2000.

Professional Experience: Assistant professor, Department of Agronomy, Nanjing Agricultural University, Nanjing, People's Republic of China, January, 1986. Lecturer, Department of Agronomy, Nanjing Agricultural University, Nanjing, People's Republic of China, June, 1987. Assistant Director and Researcher, Wheat Breeding Institute, Nanjing Agricultural University, Nanjing, People's Republic of China, January, 1990. Research assistant, Department of Agronomy, Oklahoma State University, August, 1994 to September, 1997. Teaching Assistant, Department of Computer Science, Oklahoma State University, 1999.