# Solving Travelling Salesman Problems by Adapted Genetic Algorithm

Zijing Mo

May 7th, 2017

**Abstract:** This paper presents an adapted Genetic Algorithm to find the solution of Travelling Salesman Problem, a NP-hard problem. Based on normal genetic algorithm, the method in this paper would add the "Cataclysm" thoughts to avoid limitation of local optimal solution, especially for large-quantity-size TSP. This paper also describes two practical applications about travel commuting problems with different cities in China and U.S.
**Keywords:** Travelling Salesman Problem; Genetic Algorithm; "Cataclysm thoughts"; NP-hard Problem

## 1 Introduction

Traveling salesperson problem(TSP below) asks the problems: given a list of distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? The aim for this case is to find the shortest tour to minimization of cost[1]. On the other hand, this problem is known to be NP-hard problem, which mean the solutions have to be verifiable by deterministic computations that can be performed in polynomial time. Moreover, We could think TSP is essentially the problem of permutation and combination. The simplest method to find the solution is brute-force algorithms, yet the complexities of time and space tend to be exponential order as the numbers of cities are increasing. Therefore, we need some other intelligent methods to reduce the cost and space system would pay.

Genetic Algorithms(GA below) could possess some efficient features to solve TSP. In recently, as GA is applied to solve about function optimization problem with continuous variables or discrete variables, it reveals the characteristics of robustness, global optimality, implicit parallelism and self-adaption in mathematics[2]. These features lead GA to be a widespread application methods to solve optimization problems. That is the reason why I am interested at applying GA to solve this NP-hard problem. With the deep study for TSP, it contributes to understand the internal logic of GA. On the other hand, for avoiding the limitations of GA, it was very fascinating to introduce a "Cataclysm thoughts" for improving the performance of algorithm. More details would be expounded later. For verifying the algorithm, this paper brings about three simulations to detect the adapted operational performance of GA with TSP.

To this end, the remainder of the paper is organized along the following lines. In section 2, I first describe the brief background of TSP such as origins and development. This section also introduces some methods or algorithms to solve TSP and explains the reason why some of ways are not the optimal choices. In section 3-5, we would find the descriptions of an adapted GA and three various experiments and related analysis. The last section is about conclusions and future work for this adapted GA.

## 2 Background and Literature Reviews

Travelling Salesman Problem is a puzzling problem since $19th$ century. This problem became a mathematics study in 1930s. In the 1950s and 1960s, this problem is a popular topic in scientific circles. At the beginning , RAND corporation offered some prizes to award people who solved this problems[3]. Since then, this problem is very popular among the experts who are from mathematics, physics, chemistry and computer science. In 1972, the computer scientist, Richard M. Karp show that TSP has some difficulties to find the optimal solutions. He defined those problems are NP-hard problems. With decades research, the scientists found brand-and-bound algorithms could fix those problems at first. As a result, in the next, we would review some algorithms for solving NP-hard problems, especially, find the relations between other algorithms and genetic algorithm.

**Permutations and Brute Force.** There is no doubt that the algorithm of permutations and brute force is most direct and easiest way to get the optimal solution. We just need to order the combinations from the different point sets and search them by brute force. However, the running time for this method is tremendous, which lies in $O(n!)$. It means if n is bigger than 25, our equipment could not sustain the amount of computation. We should find other methods to solve the problem.

**Dynamic Programming.** The second way would be dynamic programming. This algorithms is also named Bellman-Held-Karp algorithm. In this algorithms, the TSP problems were first classified by symmetric TSP, asymmetric TSP and multi-TSP[4]. In this paper, we just consider symmetric TSP. It means the distance from St.Paul to Washington DC is equal to the distance from Washington DC to St.Paul. The algorithm would start with the smallest sub-problem to solve TSP. Whenever computing a solution requires solutions for smaller problems using the above recursive equations, looks up these solutions which are already computed. To compute a minimum distance tour, this algorithm uses the final equation to generate 1st node, and repeat for the other nodes until we could find the optimal solutions. Unfortunately, the running time for this algorithm is O($n^2 2^n$). Unfortunately, the time to solve TSP is still exponential level.

**Branch-and-Bound.** Just as cited above, branch-and-bound algorithm is the first effective method to get the optimal solution[5]. The B&B is by far the most widely used tool for solving the NP-hard combinatorial optimization problems. After developments of few decades, B&B also could solve the Graph Partitioning Problem and Quadratic Assignment Problem. However, the cost of exact algorithms for TSP is large. For example, an instance with 85,900 points was solved using *Concorde TSP Solver*, taking over 136 CPU-years[6]. It means the efficiency is still low. As a result, heuristics and approximation algorithms walked into scientists' horizon. Those algorithms could quickly produce the solutions within reasonable time.

**Heuristic and Approximation Algorithm.** The methods of heuristic and approximation algorithm could find the solutions of TSP in extremely large numbers, like 1 million cities, in the acceptable time with a higher probability away from the optimal solution[7]. There are many bifurcation method to get the solutions of TSP in heuristic idea. One of famous methods is Nearest Fragment, the variation of Nearest Neighbour algorithm, which applies greedy algorithm to find the solution. It could connect a group (fragment) of nearest not-visited cities and find shortest route with continuous iterations[8]. Except the family of constructive heuristics, there are Christofides' algorithm to resolve TSP. The completeness and high efficiency of some heuristic algorithms allow TSP is not cost excruciating issue. Moreover, TSP is a touchstone for many heuristics randomized search methods which combines some optimization thought, such as genetic algorithms, simulated annealing.

**Genetic Algorithm.** In this paper, we would observe the application of genetic algorithm in TSP. As we see above, exact algorithms take huge cost (time and space) to solve TSP. For heuristic algorithms, some of methods might not get optimal solution. However, genetic algorithm is a perfect example to get the optimal solution with low consumption[9]. In some related experiment data, it can solve optimally TSP with up to 442 cities in an acceptable time limited (e.g. the average runtime on a SUN workstation for the TSP (431) is about 30 minutes[10]). By Whitley's discursive analysis[11], each candidate solution has a set of properties which can be mutated and altered; in normal methods, solutions are represented in binary as strings of 0s and 1s. Moreover, for improving the performance of GA, this paper would introduce the concept of "Cataclysm" thoughts, which is for avoiding the imperfection of local optimal problem.

## 3 Approach: An Adapted GA

### 3.1 Mathematical Expression

Assume that we have a set of n cities: $city = [C_1, C_2, ..., C_n]$. $\exists i,\ j, C_i, C_j \in city$. The distance between $C_i$ and $C_j$ is $d_{i,j} \in R_+$. In this part, we also suppose that $d_{ij} = d_{ji}$, which means only consider symmetric TSP. The goal in this problem is to find a unrepeatable permutation $C_1, C_2, ..., C_n$. Let the distance be T:

$$\min T_d = \sum_{n=1}^{\infty} d(C_i, C_{i+1}) + d(C_1, C_n)$$

The $d(C_i, C_{i+1})$ means the distance between city $C_i$ to city $C_{i+1}$. It is very obvious that the computing amount is very huge as n tend to infinite.

**3.2 Individual Coding and Fitness Function**

Individuals are coded by the order of cities traverse. Each element string is like $C_1, C_2, ..., C_n$ and $C_i$ which means the serial number of traversed city. The element of programming is defined as a linear array. Under the legal constraint conditions with initialization of feasible solution, crossover operations and mutation operations in TSP, namely, individual codes of cities are unrepeatable, the fitness function should be:

$$f(n) = \frac{\alpha\sqrt{n}M}{T_d}$$

In this function, $\alpha$ means a constant we preset, n means the number of cities, M means the edge length of minimal square which contains all cities in map, $T_d$ means computing practical path length from the definition of 3.1, or this algorithm calls it mileage.

**3.3 Individual Knowledge Base**

After finding a solution in each iteration, this algorithm would keep the optimal individual of solution. It means when the system meet same problem to solve, the algorithm could check the knowledge base and apply the reserved individual as initial population. It would help improving rate of convergence. In other word, this process embodies the adaptive ability of environment like nature species.

**3.4 The Generating of Initial Population**

In this algorithm, we have two approaches to generate initial population: the first are some parts of population which are produced by knowledge base (we could think the input map from outside of system), the other parts are produced by greedy algorithm; the second is all of initial population is produced by greedy algorithm (or we could consider randomly generating cities in system).

**3.5 The Selecting Operation**

This is the adapted part for GA. As first, we can think about the Jurassic Period. At that time, primates were just like the tiny toys for those colossal dinosaurs. If these monstrous creatures are still alive today, Australopithecus or Homo Sapiens would still scare for dinosaurs all day long. Human civilization will be a daydream, let alone programming or artificial intelligence. Species extinction had changed every life on the planet and reshaped the history of human being. Following this, we shall step out so-called optimal solution in system. Being trapped in local optimum is one of disadvantages for GA. What we need to do for this system is eliminating all current optimal individuals like a cataclysm. After that the algorithm would be far away from current local optimal and individuals would have sufficient space to crossed and mutate (re-evolution). Then we might find the real optimal solution. Thus, the cataclysm countdown expressions in each iteration below:

$$Cd_{new} = e(Cd_0 - Cd_{optimum})$$

$$Cd_0 \Leftarrow Cd_{new}, \text{until } Cd_0 = 0$$

$$\text{When } Cd_0 = 0, \text{eliminating all optimums and cataclysm is coming}$$

From above, $Cd$ means countdown number which is progressive decreasing in each iteration. Moreover, $Cd_{optimum}$ means the countdown value when the system finds optimal solution. The $e$ is a global variable. This value is a kind of similarity to the factor of influence in Annealing Algorithm. It would affect the generation of new countdown value. If the value is too large, algorithm would take much time in one local optimal situation. However, if the value is too small, the cataclysm always would happen. We cannot find the real optimal solution in this system. On the other hand, the range of countdown depends on the rate of evolution. The slower rate means a larger countdown value.

**3.6 The Crossover Operation**

In this algorithm, the crossover operation is OX-like method based on Khoo and Ng's ideas[12]. The system would randomly choose a matching region between two parent strings (chromosomes), $X$ and $Y$.These two selected cross-over sites denoted by " $\|$ " be expressed as follows:

$$X = [1\ 2 \parallel 3\ 4\ 5\ 6 \parallel 7\ 8\ 9],$$

$$Y = [9\ 8 \parallel 7\ 6\ 5\ 4 \parallel 3\ 2\ 1].$$

Putting selected part of $Y$ before the first part of $X$ and putting selected part of $X$ before the first part of $Y$, we would get:

$$X' = [7\ 6\ 5\ 4 \parallel 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9],$$

$$Y' = [3\ 4\ 5\ 6 \parallel 9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1].$$

In last step, algorithm would delete the same part and keep exact one element in string $X'$. The operation for string $Y'$ is identical:

$$X'' = [7\ 6\ 5\ 4\ 1\ 2\ 3\ 8\ 9],$$

$$Y'' = [3\ 4\ 5\ 6\ 9\ 8\ 7\ 2\ 1].$$

**3.7 The Mutation Operation**

(1). Inversion:

This mutation operation requires the sequence of elements between two randomly selected cross-over sites to be reversed. Following the example from 3.5, M denotes mutant individual:

$$X'' = [7\ 6\ 5 \parallel 4\ 1\ 2\ 3 \parallel 8\ 9], \text{before inverse mutation}$$

$$M = [7\ 6\ 5 \parallel 3\ 2\ 1\ 4 \parallel 8\ 9], \text{after inverse mutation}$$

(2). Heuristic:

This mutation operation bases on neighbourhood idea. For a chromosomes, the times for commutation of neighbourhood is less than $\eta$ genes. It also means we can get a family of chromosomes. From this group, the system could find the optimal as offspring of mutation. Following the example from 3.5, we know $X'' = [7\ 6\ 5\ 4\ 1\ 2\ 3\ 8\ 9]$ and M denotes mutant individuals as well. Suppose $\eta = 3$, the position of element is at 1 3 5:

Before heuristic mutation,

$$X'' = [\underline{7}\ 6\ \underline{5}\ 4\ \underline{1}\ 2\ 3\ 8\ 9]$$

After heuristic mutation:

$$M_1 = [\underline{7}\ 6\ \underline{1}\ 4\ \underline{5}\ 2\ 3\ 8\ 9]$$

$$M_2 = [\underline{1}\ 6\ \underline{5}\ 4\ \underline{7}\ 2\ 3\ 8\ 9]$$

$$M_3 = [\underline{5}\ 6\ \underline{7}\ 4\ \underline{1}\ 2\ 3\ 8\ 9]$$

$$M_4 = [\underline{5}\ 6\ \underline{1}\ 4\ \underline{7}\ 2\ 3\ 8\ 9]$$

$$M_5 = [\underline{1}\ 6\ \underline{7}\ 4\ \underline{5}\ 2\ 3\ 8\ 9]$$

And then, this algorithm would select the optimal solution from this five-member family by heuristic function.

## 4 Experiment Designs and Related Results

For designing this experiment, we would consider two experiments. The fist one is about applying fixed variables for estimating this algorithm performance and related statistical transformations and the other one is related to random variables.

### 4.1 The Fixed Variables

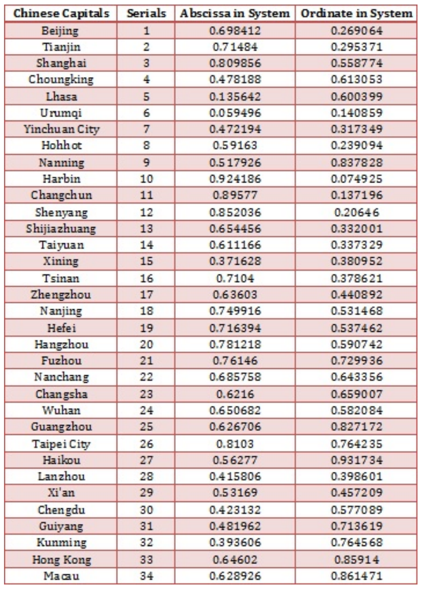
At first, we can assume that an American, who is very interested at Chinese scenes and culture, has a plan to travel through China. He knows the biggest and most prosperous cities in China is each capital of province or autonomous region. It is not like the state capitals in U.S. In China, each provincial capital can offer you authentic local food, regional historic museums or characteristic theatres. Moreover, some Chinese famous cultural or natural scenes locate at those cities such as the Great Wall, the Forbidden city or West Lake. Unfortunately, the money and time for this person is limited. As a result, he requires a path in map for visiting each provincial capital only once and returns the city of departure in the end. This TSP would be the first practical experiment we would handle.

From real geographical coordinates in satellite map, I create a corresponding coordinate system to adapt experimental map (Figure 1(a)). The form of each element (gene) in array (chromosomes) is like $E[x - value, y - value, serials]$ (Figure 2(a)). The latitude of cities is mapped to GA system's map as the x-value. The longitude of cities is mapped to this system as the y-value. The serials in this case is sorted by codes of Chinese administrative division (based on a report from Nation Bureau of Statistics of the People's Republic of China). As the capital of China, Beijing was chosen as start and end point. The input orders satisfy one of our assumptions in this case: only vising each city once and returning the departure city at last, however, it is not able to ensure the cost of path is the lowest. Thus, we need to apply this adapted GA to find a optimal path. This problem could be mathematicization:

$$G(V,E) \Leftarrow G(34,34)$$

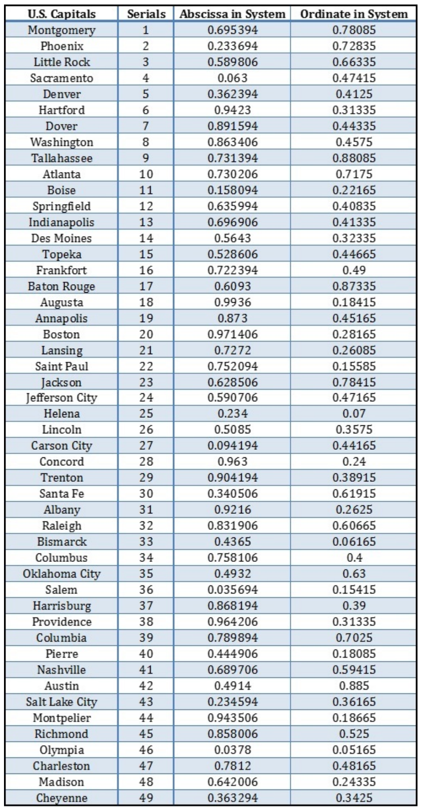
$$\text{Find } P_{optimum} \text{ Satisfies } C_P = \min \sum_{n=1}^{\infty} W_P$$

In this formula, $G$ means Chinese map, $V$ means provincial cities, $E$ means amount of paths, $C$ means total cost and $W$ means distance of two lands.



| Chinese Capitals | Serials | Abscissa in System | Ordinate in System |
|---|---|---|---|
| Beijing | 1 | 0.698412 | 0.269064 |
| Tianjin | 2 | 0.71484 | 0.295371 |
| Shanghai | 3 | 0.809856 | 0.558774 |
| Choungking | 4 | 0.478188 | 0.613053 |
| Lhasa | 5 | 0.135642 | 0.600399 |
| Urumqi | 6 | 0.059496 | 0.140859 |
| Yinchuan City | 7 | 0.472194 | 0.317349 |
| Hohhot | 8 | 0.59163 | 0.239094 |
| Nanning | 9 | 0.517926 | 0.837828 |
| Harbin | 10 | 0.924186 | 0.074925 |
| Changchun | 11 | 0.89577 | 0.137196 |
| Shenyang | 12 | 0.852036 | 0.20646 |
| Shijiazhuang | 13 | 0.654456 | 0.332001 |
| Taiyuan | 14 | 0.611166 | 0.337329 |
| Xining | 15 | 0.371628 | 0.380952 |
| Tsinan | 16 | 0.7104 | 0.378621 |
| Zhengzhou | 17 | 0.63603 | 0.440892 |
| Nanjing | 18 | 0.749916 | 0.531468 |
| Hefei | 19 | 0.716394 | 0.537462 |
| Hangzhou | 20 | 0.781218 | 0.590742 |
| Fuzhou | 21 | 0.76146 | 0.729936 |
| Nanchang | 22 | 0.685758 | 0.643356 |
| Changsha | 23 | 0.6216 | 0.659007 |
| Wuhan | 24 | 0.650682 | 0.582084 |
| Guangzhou | 25 | 0.626706 | 0.827172 |
| Taipei City | 26 | 0.8103 | 0.764235 |
| Haikou | 27 | 0.56277 | 0.931734 |
| Lanzhou | 28 | 0.415806 | 0.398601 |
| Xi'an | 29 | 0.53169 | 0.457209 |
| Chengdu | 30 | 0.423132 | 0.577089 |
| Guiyang | 31 | 0.481962 | 0.713619 |
| Kunming | 32 | 0.393606 | 0.764568 |
| Hong Kong | 33 | 0.64602 | 0.85914 |
| Macau | 34 | 0.628926 | 0.861471 |

(a) Coordinate points of Chinese provincial capitals



| U.S. Capitals | Serials | Abscissa in System | Ordinate in System |
|---|---|---|---|
| Montgomery | 1 | 0.695394 | 0.78085 |
| Phoenix | 2 | 0.233694 | 0.72835 |
| Little Rock | 3 | 0.589806 | 0.66335 |
| Sacramento | 4 | 0.063 | 0.47415 |
| Denver | 5 | 0.362394 | 0.4125 |
| Hartford | 6 | 0.9423 | 0.31335 |
| Dover | 7 | 0.891594 | 0.44335 |
| Washington | 8 | 0.863406 | 0.4575 |
| Tallahassee | 9 | 0.731394 | 0.88085 |
| Atlanta | 10 | 0.730206 | 0.7175 |
| Boise | 11 | 0.158094 | 0.22165 |
| Springfield | 12 | 0.635994 | 0.40835 |
| Indianapolis | 13 | 0.696906 | 0.41335 |
| Des Moines | 14 | 0.5643 | 0.32335 |
| Topeka | 15 | 0.528606 | 0.44665 |
| Frankfort | 16 | 0.722394 | 0.49 |
| Baton Rouge | 17 | 0.6093 | 0.87335 |
| Augusta | 18 | 0.9936 | 0.18415 |
| Annapolis | 19 | 0.873 | 0.45165 |
| Boston | 20 | 0.971406 | 0.28165 |
| Lansing | 21 | 0.7272 | 0.26085 |
| Saint Paul | 22 | 0.752094 | 0.15585 |
| Jackson | 23 | 0.628506 | 0.78415 |
| Jefferson City | 24 | 0.590706 | 0.47165 |
| Helena | 25 | 0.234 | 0.07 |
| Lincoln | 26 | 0.5085 | 0.3575 |
| Carson City | 27 | 0.094194 | 0.44165 |
| Concord | 28 | 0.963 | 0.24 |
| Trenton | 29 | 0.904194 | 0.38915 |
| Santa Fe | 30 | 0.340506 | 0.61915 |
| Albany | 31 | 0.9216 | 0.2625 |
| Raleigh | 32 | 0.831906 | 0.60665 |
| Bismarck | 33 | 0.4365 | 0.06165 |
| Columbus | 34 | 0.758106 | 0.4 |
| Oklahoma City | 35 | 0.4932 | 0.63 |
| Salem | 36 | 0.035694 | 0.15415 |
| Harrisburg | 37 | 0.868194 | 0.39 |
| Providence | 38 | 0.964206 | 0.31335 |
| Columbia | 39 | 0.789894 | 0.7025 |
| Pierre | 40 | 0.444906 | 0.18085 |
| Nashville | 41 | 0.689706 | 0.59415 |
| Austin | 42 | 0.4914 | 0.885 |
| Salt Lake City | 43 | 0.234594 | 0.36165 |
| Montpelier | 44 | 0.943506 | 0.18665 |
| Richmond | 45 | 0.858006 | 0.525 |
| Olympia | 46 | 0.0378 | 0.05165 |
| Charleston | 47 | 0.7812 | 0.48165 |
| Madison | 48 | 0.642006 | 0.24335 |
| Cheyenne | 49 | 0.363294 | 0.3425 |

(b) Coordinate points of U.S. state capitals

Figure 1: The tables of empirical coordinate points

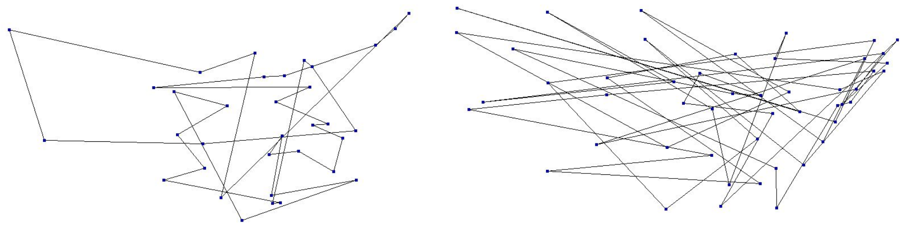

(a) Chinese visiting cities map (b) U.S. visiting cities map

Figure 2: The initial empirical maps

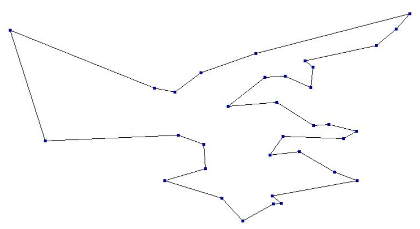
We can consider a similar problem with U.S. case. In this time, we would need to change the subject of the problem at first. Since the historic and institutional factors, for most American states the capitals are not the most flourishing cities. We can suppose that a political delegations from a developing country investigates and studies the democratic system of U.S. Their goals is understanding how the American governments operate. As a result, they would visit the capital of each state.

The acquiring methods for data of x-values and y-values is same to Chinese case. The serials in this time is a little different. I sorted these elements by state abbreviations from Montgomery (AL) to Cheyenne (WY). This data set includes the location of Washington DC and excludes oversea state capitals (Juneau and Honolulu). The related figures are Figure 1(b) and Figure 2(b). The mathematical expression in this case is:

$$G(V, E) \Leftarrow G(49, 49)$$

$$\text{Find } P_{optimum} \text{ Satisfies } C_P = \min \sum_{n=1}^{\infty} W_P$$

The system draws the optimal paths map for these two cases (figure 3 below). In the process of experiment, there are three main results: the debut of optimums, mileage and times of “catastrophe”. As system obtain the optimal solution, it would record the amount of generation which the system have went through. The product of mileage and map scale could be interpreted lowest distance cost in real life. The times of catastrophe record optimizing situation in adapted GA. To avoid system trapping in local optimal, the algorithm could eliminate all solutions and computing again. Since the number of cities is not large, this algorithm could get the first optimal solution within the millisecond unit time. The table 1 and 2 (below) is related to the variation of data in five different experiments.


(a) Chinese visiting cities map

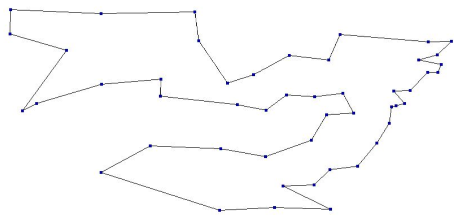

(b) U.S. visiting cities map

Figure 3: The Maps after Systems Routing

| **Adapted GA in TSP with Chinese 34 Cities** | | | |
|---|---|---|---|
| **Test Sequence** | **The Debut of Optimums** | **Mileage** | **Catastrophe** |
| 1 | 1,136,184 | 4.1888890 | Increasing |
| 2 | 10,134 | 4.1888890 | Increasing |
| 3 | 12,4870 | 4.1888890 | Increasing |
| 4 | 285 | 4.1888890 | Increasing |
| 5 | 19,819 | 4.1888890 | Increasing |

Table 1: Three Main Data for Chinese Cities TSP

| **Adapted GA in TSP with U.S. 49 Cities** | | | |
|---|---|---|---|
| **Test Sequence** | **The Debut of Optimums** | **Mileage** | **Catastrophe** |
| 1 | 89,232 | 5.034639 | Increasing |
| 2 | 22,318 | 5.034639 | Increasing |
| 3 | 156,312 | 5.034639 | Increasing |
| 4 | 9,985 | 5.034639 | Increasing |
| 5 | 1,758 | 5.034639 | Increasing |

Table 2: Three Main Data for U.S. Cities TSP

## 4.2 The Random Variables

In this case, we would consider about a map which produced 500 cities randomly. From above, We have known the consumption of space and time for solution is increasing by exponential order when the N in TSP is increasing. At first, We could assume there are five hundred cities in the map. The location of cities and the start point in map are stochastic. And then, following the order, we could create an initial map (Figure 4). The mathematical expression for this

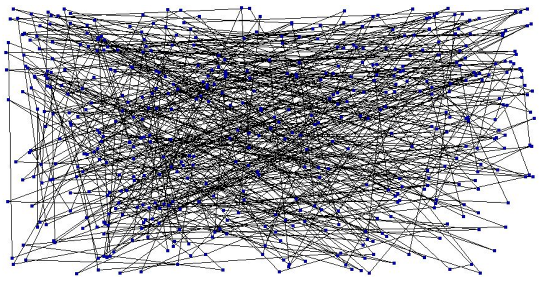


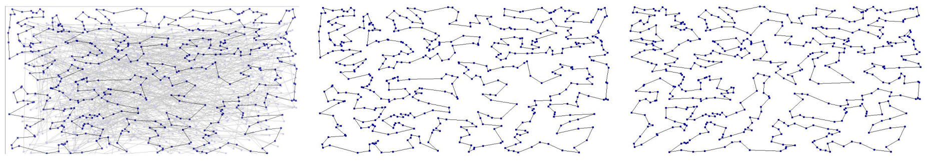
Figure 4: Random 500 Cities Map

500 cities case is similar to the fixed variable case.

$$G(V,E) \Leftarrow G(500,500)$$

$$\text{Find } P_{optimum} \text{ Satisfies } C_P = \min \sum_{n=1}^{\infty} W_P$$

However, the variation of map is tremendously surprising. The difference between those two fixed variable case is: the optimal path in graph is always changing over whole experiment time. We could see some these direct variation below (Figure 5). In this large-size TSP, those three main results for small-size TSP could not be recorded efficiently. For developing research further, I recorded some data when this adapted algorithm was terminated (Table 3).



(a) Before 1*st* Catastrophe (b) At 1*st* Catastrophe (c) At 35*th* Minute of GA Running

Figure 5: The Variations after Systems Routing

| **Empirical Data for Random 500-cities** | | | |
|---|---|---|---|
| **Mileage at LO** | 16.319 | **Mileage at FO** | 16.640 |
| **Total Generations** | 151,453,235 | **Total Running Time** | 154 min |
| **Total Catastrophe** | 278 | **Generations at FC** | 11,032 |
| **Generations at LO** | 77,201,514 | **Generations at FO** | 10,343 |

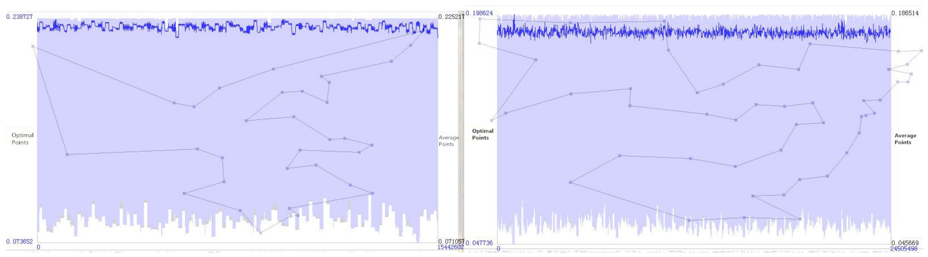
LO: Last Optimal, FO: First Optimal, FC: First Catastrophe

Table 3: Eight Main Data for Random 500-cities TSP

From the table, we might know in the short time find the convergence of large-size TSP optimal solutions is impossible. As a result, we could apply statistical tools for analyzing the trend of those data.

## 5 Analysis of the Results

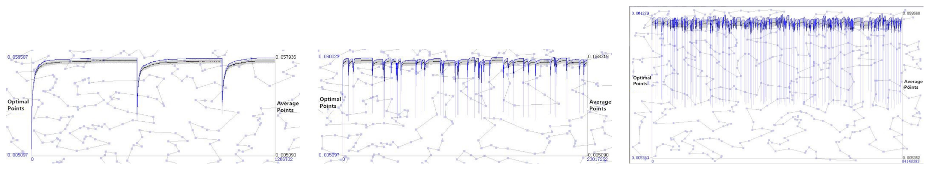
In this case, the data is not able to confirm the optimal solutions from adapted GA follow normal or other distribution. Thus tendency charts would be an efficient tool to analysis the data. The figure 6 manifests the variation of fixed variables tendency.



(a) The Variation of Chinese Data (b) The Variation of U.S. Data

Figure 6: The Tendency Graph for Fixed Variables (small-size)

From above, the optimal solutions after catastrophes tended to a fixed value, in other words, optimal values in catastrophes intervals moves around the axis of mean of optimal solution. The real optimal solution is located at the wave crest of data variation. Since the size of these two TSP is not large, the adapted GA is able to find the optimal path at a rapid rate under the dynamic equilibrium surroundings. The points of solution is much stronger than average points of population. However, the situation for large-size TSP is different (Figure 7). Comparing with small-size TSP, the algorithm in this case spends nearly 2 hours



(a) At 2*nd* Minute (b) At 35*th* Minute (c) At 154*th* Minute

Figure 7: The Tendency Graph for Random Variables (large-size)

to get similar graph above. By observation, the average points of population would reach a related large value and then drop rapidly, then again rise relatively slowly until next catastrophe's arrival. It interprets TSP has a great deal of local extreme value. The points of solutions, which is closed to extreme value, are far below the average points of population. The main reason is the most of descendants, which are produced by the mutation of comparatively optimum solution, are abnormal (very low points). Since GA continues to extend these descendants sequentially, the average points of abnormal population are inferior to the normal mean. During catastrophe, the algorithm would search solutions in all state space. The population would re-evolve. As evolution proceeds, the population would be caught in an interval of specific local extreme values. Until the population is controlled by comparatively optimal solution. The average points of population would ascend once again.

## 6 Conclusion

GA is fairly appropriate for solving TSP. In an extremely short time, the algorithm can locate a relatively optimal solution. Especially, in small-size TSP, GA could keep the optimum on consistent and efficient space. With the addition of "catastrophe" thought, we also can explore the operation pattern of GA in large-size TSP such as variation of comparatively optimal solutions and abnormal descendants. We also briefly apply this adapted algorithms in real life. On account of the limited space, I would not list the specific path planning for Chinese and U.S. cases. For random 500-city case, this algorithm is still not able to find a real optimal solution or converge around average points of population in short time. However, the analysis results in the end, this algorithm is efficient and complement. For improving this algorithm in future, I would attempt to add some latest thoughts in mutation operation. A more efficient propagation method could bring better descendants and save space and time of system running.

## References

[1] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Pearson Education, 2010. ISBN 9780136042594.

[2] Y. Wu Y. Huang and H. Liu. Based on improved genetic algorithm for tsp. *Computer Engineering and Design*, 28(24):5909–5911, 2007.

[3] E. L. Lawler. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Wiley, 1985. ISBN 0471904139.

[4] R. Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

[5] J. Clausen. *Branch and Bound Algorithms - Principles and Examples.* University of Copenhagen, March 1999.

[6] V. Chvátal D. Applegate, R. Bixby and W. Cook. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press, 2007. ISBN 9780691129938.

[7] F. Glover C. Rego, D. Gamboa and C. Osterman. *Traveling salesman problem heuristics: Leading methods, implementations and latest advances.* June 2011.

[8] S. Bandyopadhyay S. Ray and S. Pay. *Genetic operators for combinatorial optimization in TSP and microarray gene ordering.* Applied Intelligence, November 2006. doi: 10.1007/s10489-006-0018-y.

[9] J. Potvin. *Genetic algorithms for the traveling salesman problem.* Annals of Operations Research, June 1996. doi: 10.1007/BF02125403.

[10] H. Braun. *On solving travelling salesman problems by genetic algorithms.* Parallel Problem Solving from Nature, 1990. doi: 10.1007/BFb0029743.

[11] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994. ISSN 1573-1375. doi: 10.1007/BF00175354. URL `http://dx.doi.org/10.1007/BF00175354`.

[12] L.P. Khoo and T.K. Ng. A genetic algorithm-based planning system for pcb component placement. *International Journal of Production Economics*, 54 (3):321–332, 1998. ISSN 0925-5273.