

Solving the Traveling Salesman Problem: a multithreaded genetic approach

IBSAN ACIS CASTILLO VITAR

School of Sciences and Engineering, Tecnológico de Monterrey, Puebla Campus

A01014779@itesm.mx

LUIS FERNANDO DÁVALOS DOMÍNGUEZ

School of Sciences and Engineering, Tecnológico de Monterrey, Puebla Campus

A01128697@itesm.mx

SALVADOR OROZCO VILLALEVER

School of Sciences and Engineering, Tecnológico de Monterrey, Puebla Campus

A07104218@itesm.mx

November 13th 2017

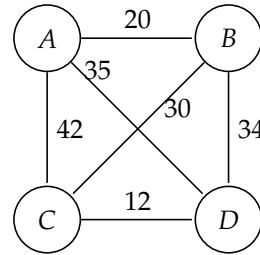
Abstract

The Traveling Salesman Problem is a one of the most common NP-Complete problems. As one of the most complicated problems in the history of Computer Science, there currently are no algorithms that can solve it in polynomial time. The problem's description is simple: given N cities that a salesman has to visit, find the order in which he must visit them to minimize the total distance traveled if he has to visit all cities once and come back to the first city after visiting the last one. For each number of cities N , the number of permutations of cities is $N!$, which makes the problem's size grow very fast. We approach the problem via genetic algorithms.

I. INTRODUCTION

THE goal of this project is to obtain the optimal solution for the Traveling Salesman Problem, a problem that consists of a salesman and a set of cities which have to be visited. The salesman has to visit each city once and return to the same city after the last one has been visited.

Consider the following image. The problem lies in finding the shortest path visiting all vertices once. For example the path 1:A,C,D,B,A and 2:A,B,C,D,A visit all the vertices once, but path 1 has a length of 108 while path 2 has a length of 97.



The number of cities in this problem is 4, and there are $4! = 24$ possible routes, which makes the problem exponentially complex as the number of cities grow.

We approach this problem with genetic

algorithms. We receive the number of cities and its geographic coordinates. We calculate the distance between every city ($n \times n$), and a permutation of the N cities.

II. MATHEMATICAL FORMULATION

For a given $n \times n$ distance matrix $C = (c_{ij})$, find a cyclic permutation π of the set $\{1, 2, \dots, n\}$ that minimizes the function

$$c(\pi) = \sum_{i=1}^n C_{i\pi(i)} \quad (1)$$

where $c(\pi)$ is the length of the permutation π , computed through a distance metric. In our case, the metric is the Haversine function to calculate the distance between two fixed points on Earth given the latitude and longitude of each point.

III. RELATED WORK

The traveling salesman problem was first considered mathematically in the 1930s by Merrill Floyd. In the 1950s and 1960s the problem became increasingly popular in scientific circles in Europe and the USA. Notable contributions were made by George Dantzig, Delbert Ray Fulkerson and Selmer M. Johnson; they expressed the problem as an integer linear problem and developed the cutting plane method.

In the following decades, the problem was studied by many researchers from mathematics, computer science, chemistry, physics and other sciences.

A chemist, V. Černý, created a thermodynamical approach to the Traveling Salesman Problem. He created a Monte Carlo algorithm to find approximate solutions of the TSP. The algorithm generates random permutations, with probability depending on the length of the corresponding route. Reasoning by analogy with statistical thermodynamics, we

use the probability given by the Boltzmann-Gibbs distribution. Using that method they could get very close to the optimal solution.

IV. METHODS

i. Input

The input to the program is a text file containing a line with an integer N , the number of cities the traveling salesman has to visit. The next N lines contain each two floating-point numbers corresponding to the latitude and longitude of the i -th city for $i = 1, 2, \dots, N$.

ii. Output

The output of the program is the best generated chromosome in terms of the distance required to visit all cities and come back to the first one.

iii. Algorithm

The algorithm used to solve this problem is as follows:

iii.1 Genetic Algorithm with Ordered Crossover

Algorithm 1 Genetic Algorithm with ordered crossover

```

1: procedure GENETIC ALGORITHM WITH ORDERED Crossover
2:   Set  $C$ , the total amount of chromosomes per generation
3:   Create  $C$  chromosomes each with a random permutation
4:   Set  $G$ , the total amount of chromosome generations
5:   for  $i \leftarrow 0, G - 1$  do
6:     Sort the array with the  $C$  chromosomes at generation  $i$ 
7:     Let the first  $\frac{C}{4}$  elite chromosomes live to the next generation
8:     Create  $\frac{3C}{4}$  child chromosomes from the  $\frac{C}{4}$  elite chromosomes
9:     Sort the array of chromosomes one last time
10:  return the first chromosome in the array

```

iii.2 Pairing Schema

Use the following pairing schema to generate child chromosomes:

1. The i -th chromosome with the $(i + 1)$ -th chromosome $(1 \leq i \leq \frac{C}{4})$

2. The i -th chromosome with the $(i + 2)$ -th chromosome $\left(1 \leq i \leq \frac{C}{4}\right)$
3. The i -th chromosome with the $(i + 3)$ -th chromosome $\left(1 \leq i \leq \frac{C}{4}\right)$

Note: all indices are taken modulo $\frac{C}{4}$.

iii.3 Ordered Crossover

Perform the ordered crossover in the following fashion:

Algorithm 2 Ordered crossover

```

1: procedure ORDERED_CROSSOVER(permutation1, permutation2)
2:   Select a random range  $[i, j]$  of the first parent's permutation of cities
3:   Place those  $(j - i + 1)$  cities in the same range of the child permutation
4:   Get the remaining cities from the other parent's permutation in the
   same order in which they appear
5:   return the new permutation

```

iv. Multithreading

Algorithm 3 Multithreading

```

1: procedure MULTITHREADING
2:   Set  $P$ , the number of processors to be used.
3:   Create an array of  $P$  threads
4:   for  $i \leftarrow 0, P - 1$  do
5:     Start the  $i$ -th thread
6:     Let the  $i$ -th thread execute  $threadSolution()$ 
7:     Let the  $i$ -th thread call  $solve()$ 
8:     Let the  $i$ -th thread print its winning chromosome

```

v. Complexity Analysis

ID	Operation	Time Complexity
1	Creating C random chromosomes	$\mathcal{O}(C^2N)$
2	Sorting the array of C chromosomes	$\mathcal{O}(C \log C)$
3	Creating $\frac{3C}{4}$ child chromosomes	$\mathcal{O}(CN)$
4	Multithreading	$\mathcal{O}(P)$

Table 1: Complexity Analysis of the Algorithm

We can see that operation 1 has the highest time complexity. Thus, $\mathcal{O}(C^2N)$ is the complexity of our algorithm.

V. RESULTS

The following table shows a comparison across four sets of input ranging from 10 to 25 cities each.

Input size (cities)	CPU Time (s)	
	1 core	4 cores
10	17	8
15	19	10
20	22	14
25	25	13

Table 2: Performance comparison across four sets of input

VI. DISCUSSION

i. Subsection One

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ii. Subsection Two

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris

lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

VII. CONCLUSION

REFERENCES

- [1] V. Černý. (1985). Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. JOURNAL OF OPTIMIZATION THEORY AND APPLICATION, 45, 1.
- [2] University of Waterloo. (2007). History of the TSP. University of Waterloo www.math.uwaterloo.ca/tsp/index.html
- [3] Willi-Hans Steeb. (2005). Genetic Algorithms. The Nonlinear workbook(350.409). London: World Scientific Publishing.