Scientific Researches in Academia
Vol. 2, Issue 2, July 2024, pp: 87-94
ISSN: 3021-9108
DOI: https://doi.org/10.3126/sra.v2i2.74286

TRAVELING SALESMAN PROBLEM: A COMPREHENSIVE REVIEW AND COMPUTATIONAL IMPLEMENTATION IN PYTHON

Saishab Bhattarai
Department of Computational Mathematics,
Kathmandu University, Dhulikhel, Kavre
saishab.bhattarai.75@gmail.com

ABSTRACT

The Traveling Salesman Problem (TSP) is a well-known problem in combinatorial optimization, in which a salesperson who wishes to sell goods in a specific set of cities is seeking for the shortest possible course that would take him or her to all the cities before returning to the original point where he or she is based. Looking at what we have come up with, it becomes evident that we want to find the shortest distance possible. Even though Algorithm TSP is a well-researched problem, the fact that it is NP-resistant has encouraged a lot of work in various solutions. As such, the aim of this paper is to present a brief discussion of TSP, its history, and various approaches developed throughout the years. Thus, when considering the brute force approach implemented in Python we examine its flaws in terms of the methodological approach and possible computational performance. Using the notion of visualization, we also offer a means for an intuitive understanding of the solution space with particular focus placed on the interdependence between data, computations, and graphics.

Keywords: *NP-Hard Problems, Brute Force, City Routing, Computational Optimization, Python.*

Introduction

The traveling salesman problem (TSP) represents a fundamental challenge in combinatorial optimization and has attracted the attention of many researchers and scientists from different fields. The basics of TSP is to determine the most efficient route across a given group of cities, ensuring that each city is visited only once and ends at the starting location. This may sound simple, but being classified as an NP-hard problem highlights the inherent complexity and nuances involved. [1]

The impact of TSP is enormous and goes beyond mere academic curiosity. For example, in the logistics sector, the right solution can lead to significant operational efficiency gains and cost savings. Additionally, industries such as manufacturing and telecommunications regularly suffer from similar challenges to their TSP:

optimizing processes and network routes to improve performance and reduce effort. [2]

The constant interaction between the theoretical complexity of TSP and its practical applications has generated a wealth of research. Over the years, a variety of algorithms, techniques, and heuristics have emerged, each attempting to address this puzzling problem from a different angle, and have proven their continued relevance in both academia and industry. [3] The main context of this paper are as follows, Section II provides Literature Review of previous works done in the field of Route Optimization, Section III Methodology describes the mathematical use of Brute Force algorithm and its implementation in Python Programming Language Section IV Results and discussion gives the summary of implementation of the algorithm and with its advantages and limitations, Section V Conclusion summarizes and concludes the project with the future implications.

Literature Review

The traveling salesman problem (TSP), a central subject in combinatorial optimization, has been the subject of extensive scientific research for decades. The purpose of this literature review is to capture key moments, methods, and transformative contributions that have shaped our understanding of TSP.

2.1 Historical Context

Although informal references to the TSP-like problems can be found as early as the 19th century, it wasn't until the 1930s that Menger began formal academic discussions and analyses on the subject [4]. In 1934, Whitney introduced the concept of a "graph," a mathematical structure used to model pairwise relations between objects. This laid the groundwork for a more systematic approach to many combinatorial problems, including the TSP. Specifically, he explored the properties of what we now call "Hamiltonian cycles" in a graph [5].

2.2 Exact Solution Approaches

While in early analysis of TSP, emphasis was made on exact solutions techniques. The goal set was to work toward a deterministic response that provided for maximum fitness. The 1960s is the starting year of the dynamic programming approach talked about by Bellman. It provided a clear, methodical framework of how the path to the TSP could be drawn, which not only was valuable for this particular problem but also served as a general framework for future research in combinatorial optimization. The principles of dynamic programming put forward by Bellman have since then been applied in a plethora of domains way beyond TSP, which is a perfect testament to the far-reaching importance of his work [6]. Cessenak and Galan 318 Parallel, linear and integer programming methods

received an opposition, following the introduction of “Simplex method” by Dantzig, an efficient technique in linear programming. The basic concept of linear programming is to find values for one or more variables to make an algebraic expression known as the linear objective function either a maximum or minimum value subject to constraints that are also linear. Although Dantzig's early aim was not to solve the TSP problem directly, he provided a way of doing so in different ways [7].

2.3 Approximation Techniques and Heuristics

As researchers struggle with the intricacies of TSP for larger datasets, approximation and heuristic methods gained momentum. Christofides' algorithm, introduced in the 1970s, provides an efficient approximation solution to the TSP. It builds on a combination of techniques: constructing a Minimum Spanning Tree, identifying odd-degree vertices, finding a Minimum Weight Perfect Matching, creating an Eulerian circuit, and converting it to a Hamiltonian circuit. Notably, its solution is guaranteed to be within 1.5 times the optimal length for metric TSP instances, making it a benchmark in approximation algorithms for the problem [8]. Heuristic approaches, rooted in intuitive principles, gained traction for solving the TSP. Key strategies like

The nearest neighbor and diverse insertion methods emerged, providing solutions that were both efficient and computationally lighter compared to exact methods [9].

2.4 Advanced Metaheuristics and Modern Approaches

Towards the latter part of the 20th century and into the 21st, metaheuristic methodologies began to emerge as powerful contenders for solving TSP. Notable among these are Genetic Algorithms, Simulated Annealing, and Ant Colony Optimization. These techniques, inspired by natural processes, showcased significant promise, especially for complex TSP scenarios [10]. Furthermore, the literature reveals a growing interest in hybrid methodologies, which combines features from multiple algorithms to push the boundaries of TSP solution quality and efficiency [11].

3. METHODOLOGY

In addressing the Travelling Salesman Problem (TSP), a brute-force approach is adopted to carefully examine every possible route, ensuring an exact solution. While this method guarantees precision, it's computationally demanding, especially with an increase in dataset size. For the purpose of our implementation, we have selected a dataset comprising 15 major cities of Nepal, namely Kathmandu, Pokhara, Lalitpur, Bhaktapur, Biratnagar, and so on. The north and east

coordinates of these cities are utilized to determine the distances, thereby assisting in identifying the optimal path.

The following outlines the methodology and the key components used in the provided Python code:

3.1Data Collection

City data is collected interactively from the internet. The code captures names of the cities and their corresponding x, y coordinates. The cities and their coordinates are stored in a dictionary named cities.

Table 1
North and South Coordinates of Major Cities of Nepal

City Name	North Coordinates	East Coordinates
Kathmandu	27.7172	85.324
Pokhara	28.2096	83.9856
Butwal	27.673	83.4822
Bhaktapur	27.671	85.4298
Biratnagar	26.4394	87.2861
Birgunj	27.0123	84.8777
Dharan	26.8124	87.2861
Janakpur	26.7288	85.9266
Hetauda	27.4167	85.0335

3.2 Distance Calculation

The pairwise Euclidean distance between every pair of cities is calculated. The Euclidean distance between two points' p and q in Euclidean n space is calculated using the formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad \text{---------- (i)}$$

Where,

- p and q are two points in Euclidean n-space.
- p_i and q_i are the coordinates of points p and q in the n-space.
- n is the dimension of the space.

For geographic coordinates (latitude and longitude), the Euclidean distance between two cities A and B with coordinates (p_1,p_2) and (q_1,q_2) respectively is:

$$d\ (A,B) = \sqrt{(q_1 - p_1)^2 - (q_2 - p_2)^2} \quad \text{----------(ii)}$$

Where,

- p_1 and p_2 are the coordinates of city A
- q_1 and q_2 are the coordinates of city B

3.3 Brute-Force Solution

Brute Force Algorithm is a straightforward way of addressing problems that involves methodically attempting each potential solution in order to identify the right one. It is frequently employed when there is enough computing power to fully examine every possibility and the problem has a narrow search space. A brute-force algorithm's primary concept is to produce and test every potential answer until the right one is identified. A brute-force approach, as used in computer science and algorithms, thoroughly searches the solution space, assessing each potential solution to see if it meets the requirements of the problem. Although brute-force methods are straightforward conceptually and straight forward to apply, they may not be feasible for problems with significant solution spaces since the number of possible solutions increases exponentially with the size of the input.

3.3.1 Mathematical Foundations of Brute Force Algorithm

3.3.1.1 Permutations and Solution Space:

The brute force algorithm involves evaluating all possible permutations of the problem's elements. For a problem with n elements, the total number of permutations is:

$$n!=n\times(n-1)\times(n-2)\times\cdots\times2\times1 \quad \text{---------- (iii)}$$

3.3.1.2 Time Complexity:

The time complexity of a brute force algorithm is typically $(n!)$. This exponential time complexity makes brute force impractical for large n because the number of evaluations needed grows factorial with the number of elements.

3.3.1.3 Example in Route Optimization:

For a route optimization problem like the Traveling Salesman Problem (TSP), where a salesman needs to visit n cities, the brute force approach would involve evaluating each of the $n!$ possible routes. The total distance for each route is computed, and the shortest one is selected as the optimal route.

Given a permutation $\sigma = (\sigma_1,\sigma_2,\ldots, \sigma_n)$, the total distance D is:

$$D(\sigma) = d(\sigma_n, \sigma_i) + \sum_{i=1}^{n-1} d(\sigma i, \sigma i + 1) \quad \text{---------- (iv)}$$

Where d (i,j) represents the distance between city i and city j.

To illustrate the computational growth:

- For n = 5, there are 5! = 120 permutations.
- For n = 10, there are 10! = 3,628,800 permutations
- For n = 20, there are 20! = $2.43*10^{43}$ permutations [12]

3.3.2 Function: *total_distance*

This function calculates the total distance of a given order of cities using the distances dictionary. It iterates through the given order, sums up the distances for the entire route, and includes the distance from the last city back to the starting city.

3.3.3 Function: *tsp_bruteforce*

The core of the brute-force approach, this function computes the distance for every possible permutation of cities using Python's itertools permutations. It maintains a record of the shortest route found. When the function has checked all permutations, it returns the optimal route.

4. Result and Discussion

3.4.1 Plotting the Route

Using python programming language libraries such as matplotlib. pyplot, the optimal route is visualized on a 2D plane. The cities are represented as red dots, and the optimal route is illustrated with blue lines connecting the cities in the sequence of the best route.

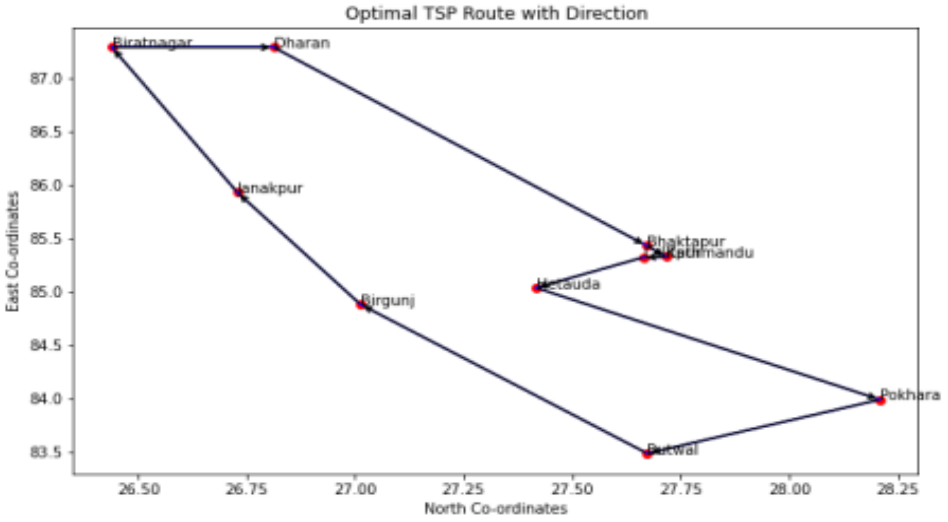

Figure 1: Graphical Representation of Route Optimization

3.4.2 Code Execution

The code starts by collecting city information. It then calculates pairwise distances, solves the TSP using the brute-force method, and finally visualizes the optimal route as below:

Optimal route: ***Butwal -> Birgunj -> Janakpur -> Biratnagar -> Dharan -> Bhaktapur -> Kathmandu -> Lalitpur -> Hetauda -> Pokhara***

3.6 Advantages and Limitations

The strength of this approach lies in its exactness – it guarantees the discovery of the optimal route. However, its computational complexity grows factorial with the number of cities. As we have seen in computational growth example for a larger number of cities, it becomes infeasible due to excessive computational time.

In-corporating the methodology described above, researchers can understand the logic behind the provided Python code for the TSP. This code serves as an example of how brute-force

methodologies can be effectively employed for small datasets but may require alternative optimization techniques as the data size increases.

5. Conclusion

The travelling salesman problem is fundamental to the study of combinatorial optimization because of its historicity and complexities. This paper also underscored the practical and theoretical relevance of TSP, which could be evidenced by the brute force algorithm used in this paper. The brute force approach, where all possible routes are considered until the best solution is found, is efficient in this context because it will always identify the optimal solution, but computationally very expensive that it cannot be used for large instances. This is because the amount of computational capability that is needed to solve the problem increases at a much faster rate with the number of cities as is seen in the brute force method which lead to the discovery that there is need to develop more scalable methods. As mentioned earlier, our research offered practical knowledge about the brute force algorithm emphasizing its significance in terms of proposing elementary techniques for use in optimization algorithms. Though the method described here is quite straightforward, learning from this exercise helped in reveling a few significant aspects of coding and computation. Further ahead, growth in the application of algorithms and advancements in quantum computing as well as machine learning offer prospects of improved solving of TSP and other kinds of optimization problems that may prove to be challenging. Therefore, brute force methods remain a useful starting point for developing efficient algorithms, TSP is a more complex problem that will undoubtedly further evolve in the future as researchers focus on creating new, improved algorithms with even higher performance in terms of solution quality and computer time.

References

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (2007). The Traveling Salesman Problem. Princeton University Press.

Bellman, R. (1962). Dynamic Programming Treatment of the Travelling Salesman Problem. Journal of the ACM (JACM), 9(1), 61-63.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms" (3rd ed.). MIT Press.

Dantzig, G. B., Fulkerson, D. R., & Johnson, S. M. (1954). Solution of a large-scale traveling-salesman problem. Journal of the operations research society of America, 2(4), 393- 410.

Dorigo, M., & Gambardella, L.M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1), 53-66.

Garey, M.R., & Johnson, D.S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman.

Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. European Journal of Operational Research, 126(1), 106-130.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(2), 231-247.

Menger, K. (1932). Das Botenproblem. Mathematische Zeitschrift, 37(1), 669-684.

Rosenkrantz, D.J., Stearns, R.E., & Lewis, P.M. (1977). An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing, 6(3), 563-581.

Whitney, H. (1934). On the Abstract Properties of Linear Dependence. American Journal of Mathematics, 57(3), 509-533.