Proceedings of the 2023 International Conference on Mathematical Physics and Computational Simulation
DOI: 10.54254/2753-8818/28/20230393

Analysis of various methodologies for the TSP problem

Sheng Xu[1, 5, 6], Xuanhao Hu[2, 7], Weiduo He[3, 8], Kaman Li[4, 9]

[1]Guanghua Cambridge International School, Shanghai, 200125, China,
[2]Shenzhen College of International Education, Shenzhen, 518043, China,
[3]Saint James School, Hagerstown, Maryland, 21740, United States,
[4]Shenzhen College of International Education, Shenzhen, 518043, China,
[5]Corresponding author

[6]18049757883@163.com
[7]h200512190908@qq.com
[8]Heweiduo0929@gmail.com
[9]1843689031@qq.com

Abstract. Linear algebra is an important mathematical method used to solve real-life problems such as optimization. This paper chose the Travelling Salesman Problem as the topic. TSP Problem had a lot of different methods and algorithms. In this paper, the principle of each method is explained, and the comparison with the test data is shown at the end of the paper. Matlab was used to support the research. The source codes of each method were uploaded to Github. According to the test data, the branch-and-bound method performed the best because it takes the least time to run and can handle the most data. TSP Problem is widely used in real life, such as transportation. The faster the code, the better the performance of the program.

Keywords: Travelling Salesman Problem, comparison, source code

1. Introduction

The Traveling Salesman Problem (TSP) is *to determine the shortest path that passes through each node once and ends at the starting node.* TSP has important applications in fields such as logistics, transportation, and circuit board design. TSP can be utilized to optimize the route taken by a delivery company to minimize time and cost, or to minimize the length of the wires connecting different components on a circuit board. The Traveling Salesman Problem (TSP) is categorized as "NP-hard." This means that there is no known efficient algorithm that can solve the problem for all inputs. Karl Menger, a mathematician, and economist first came up with the idea of TSP, and then he published his book, *"Ergebnisse eines Mathematischen Kolloquiums"* about it [1]. The TSP was eventually made popular by the scholars studying it in the 1940s.

There are different algorithms to solve TSP, including exact and inexact ones. For example, as suggested by Damon Cook, "Memetic algorithms" and "Heuristic approaches" would give an approximate solution [2]. heuristic approaches may be able to find near-optimal solutions to a problem quickly, without necessarily guaranteeing finding the optimal solution. The choice of the algorithm used depends on the size and complexity of the problem, as well as the desired solution quality and computational efficiency.

DOI: 10.54254/2753-8818/28/20230393

Sometimes, linear programming (LP) can also be used to find the exact optimal solution for TSP if the computation is small. The problem needs to be converted into a mathematical model with objective functions and constraints. But for some TSPs with large inputs, it is not always feasible to put it in the standard form of LP and get the optimal solution. As a result, developing efficient algorithms for solving TSP is an ongoing and active area of research.

In this paper, we will present different formulations and evaluate and recommend some methods that require computer-aided computation.

2. Formulations

In this section, we will present seven different methods for solving the TSP problem. The section is divided into two parts. One of these two parts is the introduction and examples of the formulations that do not require computer-aided computation. The second part is about the formulations that require computer-aided computation.

2.1. Formulation Without Coding

2.1.1. Held-Karp bound. The Held-Karp bound is a dynamic programming algorithm for TSP. It is relatively more efficient to provide an accountable lower bound for the TSP problem when the input is large. The large-scale TSP with up to 30,000 cities is discussed by Arnold F [3]. Held-Karp algorithm creates a two-dimensional table with each axis corresponding to subsets of cities. At each stage of the algorithm, it uses previously computed entries in the table to calculate the current entry, based on the cost of adding a new city to the cycle at the lowest cost. This involves calculating the cost of the path that includes all cities in the current subset except for the last one, and then adding the distance that connects the last city to the current one.

2.1.2. The Held-Karp lower bound algorithm and how it is used to solve TSP. The Held-Karp algorithm is a dynamic programming approach used to solve the TSP. The algorithm works by breaking down the problem into smaller subproblems and finding optimal solutions one by one to those subproblems. Assuming there are n cities, and we number the cities from 1 to n. In the context of the algorithm, the cities form a set, and the starting point is chosen as city 1.

To implement the algorithm, we create a two-column graph A, which is used to store the minimum path lengths between different subsets of the cities. One column represents the cost to travel through any subsets of the city set that do not contain city 1, and the other column represents the last city we traveled to. Saying that we start at the subset that only contains one city j (not contain city 1), we simply get the cost that we travel from city 1 to j.

If the table can be completed, we only need to find the subset S that contains all cities except city 1. There should be n-1 values here, representing the shortest paths from city 1 to city j by passing through all cities once. Then we just need to add the distance from j to 1 to these values to obtain the lengths of circuit paths. We can then select the minimum value from them.

2.2. A survey on the efficiency of Held-Karp lower bound.

The Held-Karp algorithm has a time complexity of O ($n^2 * 2^n$), which is an improvement over the brute force approach with a time complexity of O(n!), where n is the number of cities in the TSP. The algorithm is widely used in the fields of computer science and operations research.

Researchers have proposed various modifications and heuristics to improve the algorithm's performance, such as pruning techniques to reduce the search space and parallelization to take advantage of modern computing architectures. In D. S. Johnson's study, their group used iterative Lagrangean relaxation techniques to improve the HK bound [4]. They proved that it is a feasible method and as the number of cities went up to 3000, the HK bound has the $.00001$ gap under Topology and Planar. AThe95% confidence interval of the HK bound tends to be approximately

DOI: 10.54254/2753-8818/28/20230393

$.7124 \pm .0002$.The study by Pascal Benchimol also considered iterative Lagrangean relaxation and constraint programming to improve the HK bound [5,6].

3. Graph Theory

In mathematics, Graph Theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices (also called nodes or points) which are connected by edges (also called links or lines). A distinction is made between undirected graphs, where edges link two vertices symmetrically, and directed graphs, where edges link two vertices asymmetrically.

r0.25

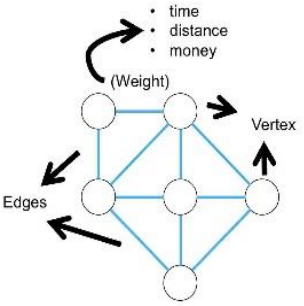

Figure 1. Basic Definitions of Graph Theory

Graphs are one of the principal objects of study in discrete mathematics.

Figure 1 shows the basic definition of Graph Theory. In Graph Theory, we call those edges that have clear direction Direct Edges, and we call those edges that have no clear direction Indirect Edges.

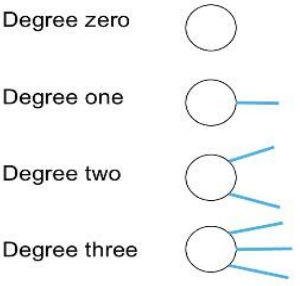

Figure 2. Different Kinds of Vertexes in Graph Theory

Also, there is another important definition for the Graph Theory. For vertexes, we can divide them into different kinds of vertexes depending on how many edges are connected with the vertexes. Figure 2 shows four different kinds of vertex. If the vertex is connected by zero edges, we can call it Degree Zero. If it is connected by one edge, we can call it Degree One. Then and so on.

In Graph Theory, the most important thing is that when people are solving TSP problems, people need to build up a circuit which means they need to find a path that can return to the starting point after pathing all the vertexes. After finding all the circuits, we can find out the optimal circuit for TSP problems which means passing all the vertexes with the least cost. Now, I will give out some specific ways to solve TSP problems by using the Graph Theory.

We can first solve the TSP problem by using the Euler Path and Euler Circuit in graph Theory. To create a Euler path, we need to go through every edge once without repeating, and to create an Euler Circuit, we need to make the Euler Path back to the starting point. However, there is a condition for us to apply this way to solve the TSP problem, we need to make sure that all vertexes have even number

DOI: 10.54254/2753-8818/28/20230393

degrees. Because there must be some cases in real life where we can find out even edges are connected to the vertex, we can use Eulerization to make sure all vertexes have an even number of degrees.

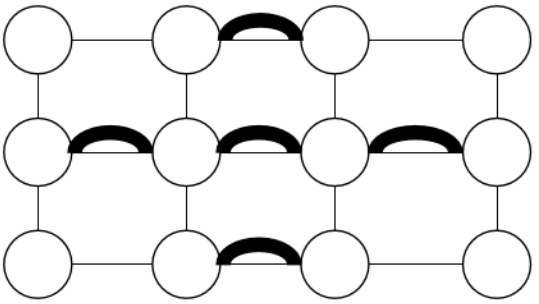

Figure 3. A Sample Situation

In Figure 3, we can find out that 8 vertexes have not had an even number of degrees now, we can just connect those points to make sure that they have an even number of degrees. This is called Eulerization.

However, though we can use this way to find different circuits in a TSP problem, it is hard for us to find the optimal answer.

Then, the second way to solve the TSP problem by Graph Theory is to build up a Hamiltonian Circuit which means we need to build up a circuit that visits every vertex once without repeating and not returning to the starting point.

For example, if we want to solve the TSP problem in this graph by using the Hamiltonian Circuit, we can find three circuits in total which are

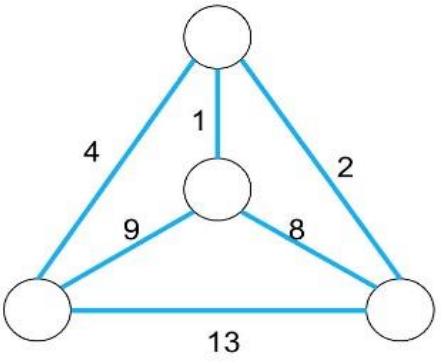

Figure 4. A Case of TSP Problem

Figure 4 shows a Special Case of a TSP Problem with 4 Vertexes.

1. ABCDA: /(4+13+8+1=26/)
2. ABDCA: /(4+9+8+2=23/) (optimal)
3. ACBDA: /(2+13+9+1=25/)

So, it is clear to see that the second circuit is the optimal answer. However, this way, if we assume that the total vertex number is N then, we will have $\frac{(N-1)!}{2}$ circuits, which means in real life, if we want to solve a TSP problem with 20 cities in a country, then the total unique circuits we will have is 60822550204416000 which is too large for even computer to calculate.

So, overall, though this way can find out the optimal answer, it is not an efficient way.

The third way to solve the TSP problem by using Graph Theory is the Neighbor Algorithm, for this way to solve the TSP problem, there are two steps in total. Firstly, we need to choose a starting point,

DOI: 10.54254/2753-8818/28/20230393

then we need to choose the least-cost edge every time. Finally, we need to repeat this process to find the answer.

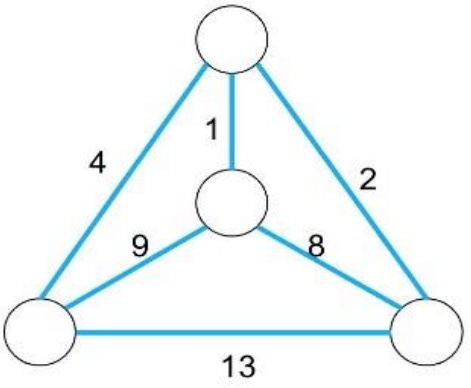

Figure 5. A Case of TSP Problem

Figure 5 shows a Special Case of a TSP Problem with 4 Vertexes

However, though this way can easily solve the TSP problem quickly, we can hardly find the most optimal solution. Also, we can use the same diagram to explain this. As we know, for this case, the optimal solution is the route ABDCA, but if we use the Neighbor Algorithm to solve this TSP problem and choose vertex A as a starting point, we will get ACDA finally which is not an Euler circuit. So, this is the problem for this way to solve the TSP problem.

Lastly, there is both an optimal and efficient way to solve the TSP problem is Kruskal's Algorithm which can also be called Spaning tree. We can adapt this way to real cases by just connecting all the cheapest edges and we can get a path in the shape of a tree, though it is not a tree, it doesn't matter.

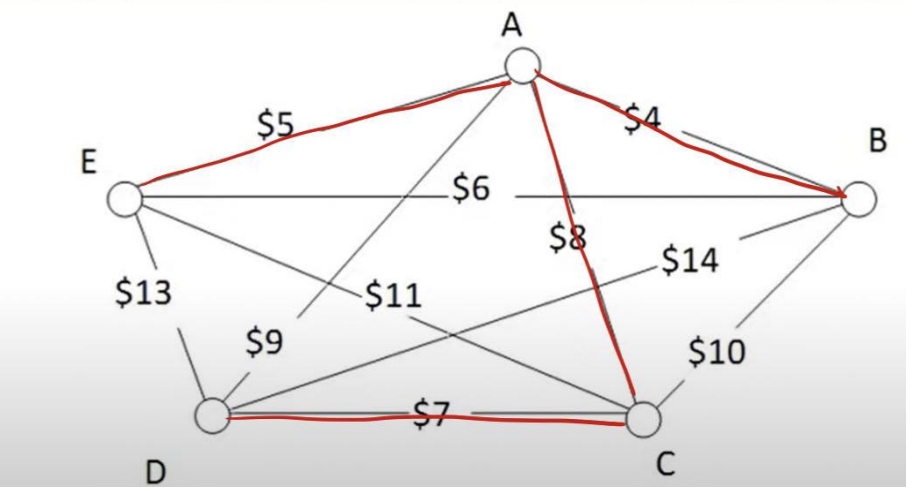

Figure 6. Another Case of TSP Problem

As Figure 6 shows, by using a Spaning Tree to solve a TSP problem, we only need to connect AE, AB, AC, and DC, then we can get an optimal answer efficiently. However, the only problem is that we need to go through AC twice.

Now, I am going to introduce a real-world case—-the *Chinese Postman Problem*. In graph theory, a branch of mathematics and computer science, Guan's route problem, the Chinese postman problem, postman tour, or route inspection problem is to find the shortest closed path or circuit that visits every edge of a (connected) undirected graph. When the graph has an Eulerian circuit (a closed walk that

DOI: 10.54254/2753-8818/28/20230393

covers every edge once), that circuit is an optimal solution. Otherwise, the optimization problem is to find the smallest number of graph edges to duplicate (or the subset of edges with the minimum possible total weight) so that the resulting multigraph does have an Eulerian circuit. It can be solved in polynomial time.

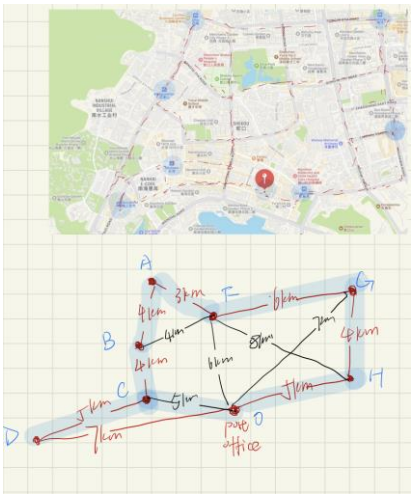

Figure 7. A Real Case with Map

3.1. A real-world case—Chinese Postman Problem

As you can see in Figure 7, this is a part of the map of Shenzhen China, in this case, a postman needs to send a post to 8 different places. To solve this problem, firstly, we need to convert the map into a Graph as the diagram shows, then we can just use the Spanning Tree. We just need to connect all the least cost edges and we can get the optimal answer efficiently.

4. Formulation With Coding

This part is about how to solve the TSP problem by using the program. The computer can use a special program to solve a difficult task, so many complicated calculations can be done by the computer. All coding in this section is done using MATLAB. In this part, it is assumed that the problem TSP is a **Symmetric Traveling Salesman Problem**.

4.1. Symmetric Traveling Salesman Problem

The symmetric traveling salesman problem (TSP) is *the problem of finding the shortest Hamiltonian cycle (or tour) in a weighted finite undirected graph without loops*.

4.2. Canonical Form

To solve a realistic problem on the computer, the abstract realistic problem must be converted into a mathematical form, such as the canonical form.

In the case of the TSP problem, the problem is defined o a grah $D = (N, A)$, which N is the number of the nodes, and A is set of the indexes of the nodes. The distance between node i and node j is defined as $arc(i,j)$ for $\forall(i,j) \in A$, and $arc(i,j)$ is equal to $arc(j,i)$ due to the symmetric condition.

According to the basic definition of TSP, the objective function is used to determine the distance of the shortest path that passes through each node, such as

$$min\sum d_{i,j} * X_{i,j} \ (1)$$

$X_{i,j}$ is a binary value that can take only two values: 1 and 2. If $X_{i,j}$ is equal to 1, $arc(i,j)$ is selected, and the distance of $X_{i,j}$ is computed to the optimal solution. Otherwise, $arc(i,j)$ is not included in the

DOI: 10.54254/2753-8818/28/20230393

optimal solution, and $X_{i,j}$ is not calculated into the optimal the objective function [equation1] is the sum of the product of $d_{i,j}$ and $X_{i,j}$.

According to the basic definition of TSP, each node must be traversed and can be traversed only once. Consequently, there is only one arc entering a node and only one arc leaving a node. In addition, the total number of arcs included in the optimal solution is equal to the total number of nodes. These three conditions can form a set of constraints.

$$\sum X_{i,j} = n \ (2)$$

$$\sum X_{i,k} = 1, \forall i \in V, k \neq i \ (3)$$

$$\sum X_{k,j} = 1, \forall j \in V, k \neq j \ (4)$$

$$X_{i,i} = 0, \forall i \in V \ (5)$$

1. The constraint (2) set the number of $arc(i,j)$.
2. The constraint (3) limit the number of the arc leaving the each of the nodes.
3. The constraint (4) limit the number of the arc entering the each of the nodes.
4. The constraint (5) set the value of $X_{i,i}$ to be zero.

Therefore, the canonical form can be obtained by combining the constraints and the objective together.

$$min \sum d_{i,j} * X_{i,j} \quad (1)$$

$$\sum X_{i,j} = n \quad (2)$$

$$\sum X_{i,k} = 1, \forall i \in V, k \neq i \quad (3)$$

$$\sum X_{k,j} = 1, \forall j \in V, k \neq j \quad (4)$$

$$X_{i,i} = 0, \forall i \in V \quad (5)$$

4.3. DFJ Formulation

The Danzig-Fullerton-Johnson formulation (also known as the DFJ model) is a widely used mathematical model for freight transportation. It was developed by Bernard Danzig, Ralph W. Fullerton, and Marvin L. Johnson in the mid-1960s, and has since been refined and adapted by many other researchers.

The DFJ model is based on linear programming, which is a method for optimizing a mathematical model subject to constraints. The model considers a range of factors that affect the movement of goods, including transportation costs, production costs, inventory costs, and demand for goods.

The DFJ model is particularly useful for optimizing the movement of goods over complex transportation networks, such as those involving multiple modes of transportation (e.g., truck, rail, and ship) or multiple origins and destinations. By modeling the transportation system as a set of interdependent flows and constraints, the DFJ model can help decision-makers optimize freight transportation and logistics in a cost-effective and efficient manner.

DFJ is a relatively primitive approach to integer linear programming (LP) formulation. It has $2^n + 2n - 2$ constraints, it is evident that due to limitations on ordinary computers' hash rate, normal computer configuration cannot compute the great number of vertices. Consequently, this restricts the performance of DFJ formulation to find the solution for a great number of vertices. Thus, DFJ formulation is prone to having sub-tour issues.

$$\sum \sum y_{i,j} \leq |Q| - 1 \quad i \in Q, j \in Q \quad Q \subset 1,2,\dots,n \quad 2 < |Q| \leq n - 1 \quad (6)$$

The fundamental of DFJ formulation (The code can be found in the "https://github.com/vdggtw/Travelling-Salesman-Problem) is simply examining the visited nodes and identifying the weight of the distance between two nodes.

DOI: 10.54254/2753-8818/28/20230393

1. $Y_{i,j} \in 0,1$ represents for nodes already visited
2. There must be greater than two nodes to have a tour
3. The number of visited nodes must not exceed the number of graphs $|Q| - 1$

4.4. MTZ Formulation

MTZ formulation (The code can be found in the "https://github.com/vdggtw/Travelling-Salesman-Problem) was proposed by Miller, Tucker, Zemlin. A variable that belongs to integers t_i for $i \in [2,3,\ldots,n]$is added to record the times at which node i is visited. It is cclearthat if an $arc(i,j)$ is selected, the time of node i at which i is visited must larger than that of node j.

$$t_j > t_i - B * \left(1 - X_{i,j}\right) \tag{7}$$

In this constraint (7),B is a arbitrarily large value. When $X_{i,i}$ is equal to 1, which means $B * \left(1 - X_{i,j}\right)$ is equal to zero. Therefore, this constraint is changed to $t_j > t_i$. There is an example that explains how to eliminate subtours by using the MTZ method.

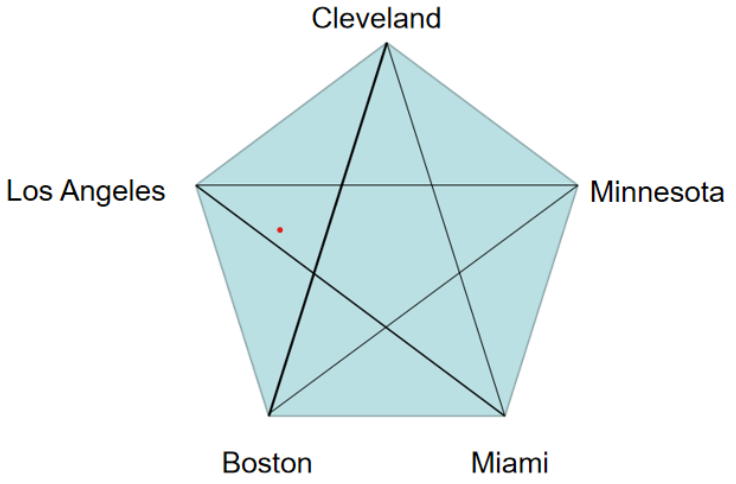

Figure 8. Special case of TSP

4.5. An Example of a TSP Problem

Figure 8 consists of five American cities, which are Los Angeles, Cleveland, Minnesota, Miami, and Boston. Let's Assume that one of the subtours includes Cleveland, Boston, and Los Angeles, and we can get $s = Cleveland, Boston, LosAngeles$. Another subtour is $s' = Miami, Minnesota$. So, $X_{Cleveland,Boston} = 1$, $X_{Boston,LosAngeles} = 1$, $X_{LosAngeles,Cleveland} = 1$ Now as per constraint (7)

$$t_{Boston} > t_{Cleveland}$$

$$t_{LosAnegles} > t_{Boston}$$

$$t_{Cleveland} > t_{LosAnegles}$$

It is obvious that it is impossible to find a solution for this set of conditions. The proof of contradiction can be used to prove this. There are $following$ inequalities

$$t_2 > t_1$$

$$t_3 > t_2$$

$$t_4 > t_3$$

$$\vdots$$

DOI: 10.54254/2753-8818/28/20230393

$$t_n > t_{n-1}$$

$$t_1 > t_n$$

Let us assume that these inequalities are correct, and we can add them together to get an inequality

$$t_2 + t_3 + \ldots\ldots + t_n + t_1 > t_1 + t_2 + \ldots\ldots + t_{n-1} + t_n$$

The left side is equal to the right side. This equality is false, and the assumption is false. Therefore, it is proven.

Therefore, adding the constraint[equation6] to the canonical form can delete the subtours. The combining LP model can be written as

$$min \sum d_{i,j} * X_{i,j} \ (1)$$

$$\sum X_{i,j} = n \ (2)$$

$$\sum X_{i,k} = 1, \forall i \in V, k \neq i \ (3)$$

$$\sum X_{k,j} = 1, \forall j \in V, k \neq j \ (4)$$

$$X_{i,i} = 0, \forall i \in V \ (5)$$

$$t_j > t_i - B * \left(1 - X_{i,j}\right) \ (7)$$

4.6. GG Formulation

The Gavish-Graves formulation (also widely known as the GG formulation, the code can be found in the "https://github.com/vdggtw/Travelling-Salesman-Problem) is known as the prototypical formulation for commodity flow formulations, where the additional variables represent commodity flows through the arcs and satisfy additional flow conservation constraints

4.6.1. Additional flow conservation constraints. Algebraically, the constraint is that the sum of the arc flows directed toward a node plus the supply of the node (if any) is equal to the sum of the arc flows directed away from that node.

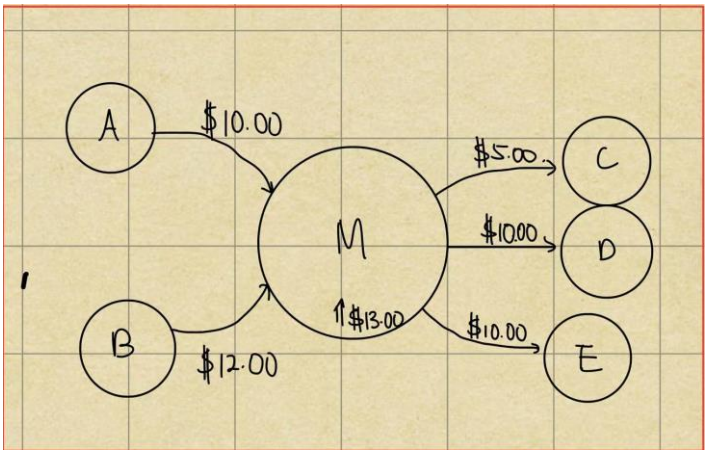

Figure 9. Single Commodity Flow

Figure 9 shows a special case of Single Commodity Flow.

The GG formulation has numerous advantages over other formulations:

1.GG formulation has fewer constraints compare to DFJ; it has "n(n-2)-1" number of constraints. It allows the formulation to find a solution for a considerably great number of vertices.

DOI: 10.54254/2753-8818/28/20230393

2. GG formulation is, in fact, a highly adaptive and widely applicable formulation, designed to solve complex ATPS where the weight of the routine may depend on the "traveler". The GG formulation supports multi-commodity flow, just like what has been previously discussed about merchandise passing customs. Based on the commodity type, the percentage of taxation expense may differ diversely.

3. GG utilizes an additional set of variables to determine of presequence arcs between each two points. This is GG formulation's elimination method for solving sub-tour issue.

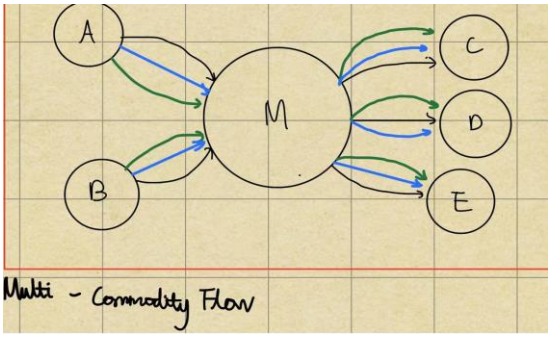

Figure 10. Multiple Commodity Flow

Figure 10 shows a case of Multiple Commodity Flow.

$$\sum_{j=1} Z_{i,j} - \sum_{j\neq 1} Z_{i,j} \quad i = 2,\ldots,n \ (8)$$

$$Z_{i,j} \leq (n-1) * y_{i,j} \quad i = 2,\ldots,n \quad j = 1,\ldots,n \ (9)$$

$$Z_{i,j} \geq 0 \quad \forall i,j \ (10)$$

Constraint [equation 8]: Check if flow variable exists between node s with only one unit.

Constraint [equation 9]: Ensures flow is possible when nodes are connected.

Constraint [equation 10]: $Z_{i,j}$ describes a single commodity's flow vertex $x_{i,j}$ from every other vertex.

4.6.2. Branch and Bound Formulation. Unlike the other three formulations, Branch and Bound formulation (The code can be found in the "https://github.com/vdggtw/Travelling-Salesman-Problem) eliminates the subtours by separating some small problems from the main problem rather than increasing the number of constraints. First, we use a simplex loop to obtain a feasible solution with or without subtours. If the feasible solution contains no subtours, the optimal solution is found. Otherwise, we should choose a subtour with the least number of nodes as the main subtour. At this point, we separate the subproblem from the main problem, and let one of the $X_{i,j}$ in the main subtour be 0. To force $X_{i,j}$ to be zero, we can let the distance from i to j be arbitrarily large, such as 10^8.

DOI: 10.54254/2753-8818/28/20230393

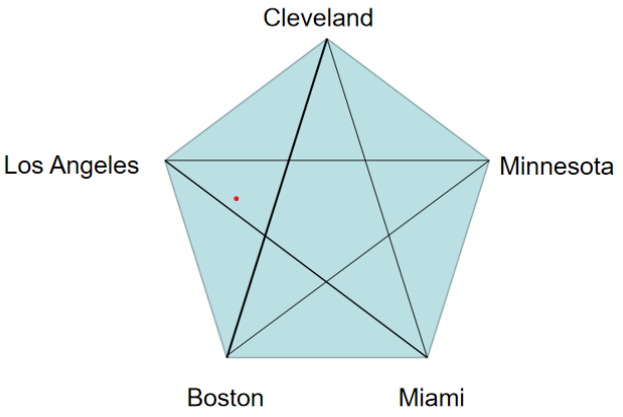

Figure 11. Special case of TSP

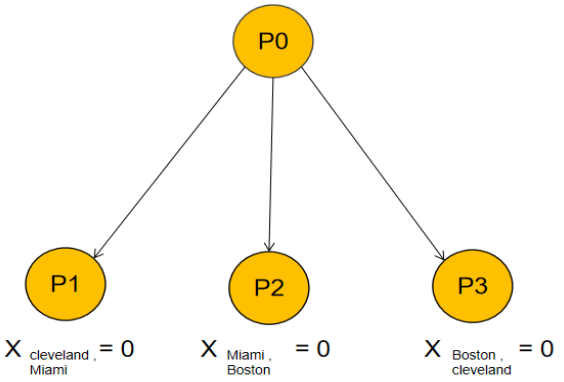

Figure 12. Subtours of the big path

Figure 11 and 12 shows a special case of TSP and its subtours. In this case, we choose the subtler with the largest number of nodes to explain more clearly. So the main subtour is Cleveland → Boston → Los Angeles. In problem 1(P1), $X_{Cleveland,Boston}$ is chosen as zeros, so the $d_{Cleveland,Boston}$ in the distance matrix are changed to 10^8.

$$\mathbf{X} = \begin{pmatrix} NuLL & 49.0 & 47.5 & 39.7 & 38.0 \\ 49.0 & NULL & 37.4 & 52.4 & 24.3 \\ 10^8 & 37.4 & NULL & 40.8 & 31.6 \\ 39.7 & 52.4 & 40.8 & NULL & 28.5 \\ 38.0 & 24.3 & 31.6 & 28.5 & NULL \end{pmatrix}$$

In the same way, we can create problem 2(P2) and problem 3(P3).

Note that P1 and P2 partially overlap. In this example, the Hamiltonian cycle Cleveland → Boston → Miami → Los Angeles → Minnesota is present in both P1 and P2. The efficiency of the computations may be affected by the overlaps, so we need to delete the overlaps without losing any solution. To eliminate the overlaps, we can set $X_{Cleveland,Boston}$ to 1 in P2. The method to set $X_{Cleveland,Boston}$ to 1 is like setting $X_{Cleveland,Boston}$ to 0. We can simply set the distance from other points to Miami and the distance from Cleveland to other points to 10^8.

DOI: 10.54254/2753-8818/28/20230393

$$\mathbf{X} = \begin{pmatrix} NuLL & 49.0 & 47.5 & 39.7 & 38.0 \\ 10^8 & NULL & 37.4 & 52.4 & 24.3 \\ 47.5 & 10^8 & NULL & 10^8 & 10^8 \\ 10^8 & 52.4 & 40.8 & NULL & 28.5 \\ 10^8 & 24.3 & 10^8 & 28.5 & NULL \end{pmatrix}$$

There is no solution lost since we have already let the $X_c leveland, miami$ to be 0 in the P1. In the same way, The P3 can be changed to a more efficient form. Figure 13 shows the subtours after improvement.

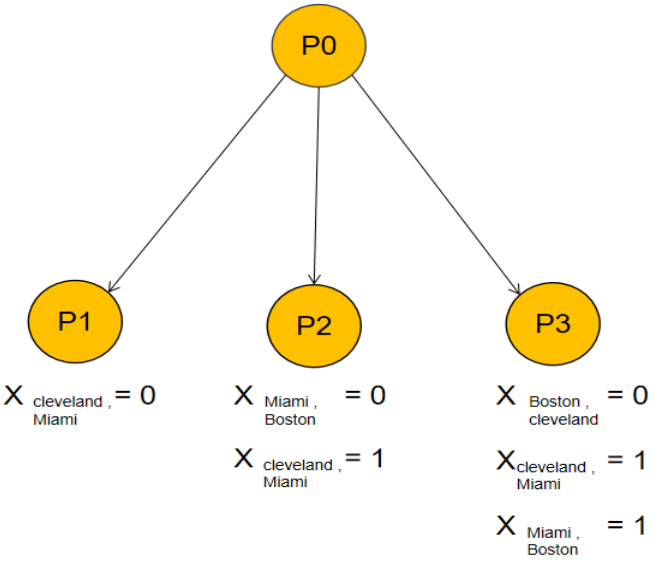

Figure 13. More efficient sub-tours

After that, we get three subproblems. Then we use the simplex loop again to solve these three problems and get the three feasible solutions. If there is only one solution without subtours, that solution will be the final answer. If there is more than one solution without subtours, the smallest solution will be selected as the final answer. If there is no solution without subtours, the smallest solution will be selected as the next main problem, and the separation will be performed again. Finally, we can repeat the whole procedure until we get a feasible solution without subtours. The following 10 steps are the summary of the whole procedure. Step 1: Solve the original problem to get a solution with or without subtours Step 2 If no subtour in the solution, go to Step 10 Step 3:Choose a subtour with the least number of nodes from the original problem as the main problem Step 4:Separate the main problem into some subproblems Step 5:Solve the subproblems, and get some feasible solution Step 6:If only one solution without subtours, go to Step 10Step 7:If more than one solutions that have no subtour, go to Step 9 Step 8:Choose the smallest solution as the original problem, and go to Step 3 Step 9:Choose the smallest solution from the solutions without subtours Step 10:The optimal solution is found The Branch and Bound formulation is implemented in MATLAB as shown below.

Table 1. Comparison and Evaluation

The number of nodes	4 nodes	6 nodes	8 nodes	20 nodes	50 nodes	100 nodes
DFJ(running time)	0.301s	0.453s	87.674s	NULL	NULL	NULL
MTZ(running time)	0.613s	0.576s	10.264s	3566.39s	7214.217s	NULL
GG(running time)	0.514s	0.618s	23.419s	4028.539s	8739.624s	NULL
BAB(running time)	1.333s	1.561s	1.229s	8.045s	79.949s	260.601s

DOI: 10.54254/2753-8818/28/20230393

Recommendation:

1. $NumberofNodes \leq 30$: DFJ Formulation
2. $30 \leq NumberofNodesleq100$: MTZ or GG Formulation
3. $100 \leq NumberofNodes$: Branch and Bound Formulation

Table 1 shows the running times of each algorithm with different numbers of nodes. According to the data, the branch-and-bound formulation covers the largest range of the node set and has a relatively short running time with a large number of nodes. However, DFJ, MTZ, and GG can be perfectly executed with a small number of nodes. Thus, BAB is the best choice for a large number of nodes, and DFJ, MTZ, and GG are good choices for a limited number of nodes.

5. Conclusion

The algorithm of TSP can be used in many areas, such as automatic transportation systems, which is an important part of mankind's path to full automation. The runtime determines the performance and response speed of the system. Therefore, better algorithms are needed as systems in the future will have to deal with huge amounts of data, such as millions of nodes. The shorter the runtime, the better the performance.

References

[1] Nigel Cummings (2000) The OR Society, "A Brief History of the Travelling Salesman Problem", https://www.theorsociety.com/about-or/or-methods/heuristics/a-brief-history-of-the-travelling-salesman-problem/

[2] Damon Cook, "Evolved and Timed Ants: Optimizing the Parameters of a Time-Based Ant System Approach to the Traveling Salesman Problem Using a Genetic," accessed March 2, 2023, https://www.cs.nmsu.edu/ dcook/thesis/paper2.html.

[3] Florian Arnold, Michel Gendreau, Kenneth S¨orensen,"Efficiently solving very large-scale routing problems," Computers & Operations Research, Volume 107, 2019, Pages 32-42, ISSN 0305-0548, https://doi.org/10.1016/j.cor.2019.03.006.

[4] D. S. Johnson, L. A. McGeoch, E. E. Rothberg, "Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound," Proceedings of the 7th Annual ACMSIAM Symposium on Discrete Algorithms (1996), 341-350, http://dimacs.rutgers.edu/ archive/Challenges/TSP/papers/HKsoda.pdf

[5] Pascal Benchimol, Jean-Charles R´egin, Louis-Martin Rousseau, Michel Rueher, Willem-Jan Hoeve, integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 2010, Volume 6140 ISBN: 978-3-642-13519- 4, https://citations.springernature.com/item?doi=10.1007/978-3-642- 13520-0 6

[6] Luís Machado, Simes M D G , Souza R R ,et al.cincia da informao e web semntica: uma relao efetiva ou apcrifa? information science and semantic web: an effective or apocryphal relationship?[J]. 2019.DOI:10.1007/978-3-642-28977-4_12.