Subject Area:

Mathematics, Group 5

Research Question:

To what extent mathematical optimization methods can be applied to improve algorithmic efficiency in application to the traveling salesmen problem?

Session:

May 2024

Word Count: 3897

Table of Contents

O. Introduction to Key Terms

Introduction to Big O Notation and Polynomial Time:

Big O notation is a mathematical concept used in computer science to analyze and describe the efficiency of algorithms and the growth rate of their resource usage. It provides a method to express the worst-case scenario of an algorithm's performance in terms of input size.

Polynomial time refers to the computational efficiency of an algorithm, specifically in relation to the size of the input. An algorithm is said to run in polynomial time if its execution time is proportional to a polynomial function of the input size.

The general form of a polynomial function is expressed as $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$, where n is the input size, a_k, a_{k-1}, …, a_1, a_0 are constants, and k is the highest degree of the polynomial. The exponent k indicates the degree of the polynomial and represents the dominant factor influencing the algorithm's runtime.

In Big O notation, polynomial time complexities are often expressed as $O(n^k)$, where k represents the degree of the polynomial. This notation provides a concise way to describe the upper bound of an algorithm's time complexity, allowing for a quick comparison of different algorithms as the input size increases.

P Class (Solvable in Polynomial Time):

The P class consists of decision problems that can be solved by a deterministic Turing machine in polynomial time. In simpler terms, these problems have algorithms with efficient solutions. An example is the problem of determining whether a given graph is connected. Algorithms like depth-first search or breadth-first search can solve this problem in polynomial time, making it a member of the P class.

NP Class (Unsolvable in Polynomial Time)

NP (nondeterministic polynomial time) problems represent a class of decision problems that, in theory, can be efficiently solved by a nondeterministic Turing machine in polynomial time. These problems share the common characteristic that solutions can be verified quickly, but finding a solution efficiently is not guaranteed.

One example of an NP problem is the Subset Sum Problem. This problem involves determining whether a given set of positive integers has a subset that adds up to a specified target sum. The verification of a candidate subset, ensuring it sums to the target, can be done in polynomial time.

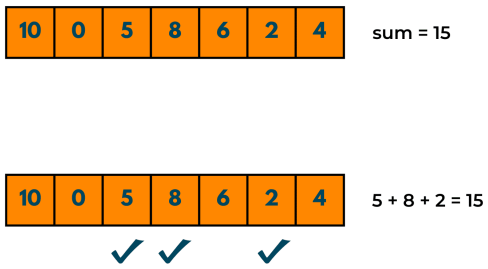

Figure 1: Subset Sum Problem Example

In assessing the inherent difficulty of computational tasks and forms the basis for many theoretical and practical considerations in computer science, understanding these complexity classes is crucial

1. Introduction TSP

In the dynamic field of computational mathematics and artificial intelligence, the Travelling Salesman Problem (TSP) has emerged as a central problem that contains the compound relationship between problem simplicity and computational complexity. This problem, deeply rooted in the fields of optimization and combinative mathematics, presents a fundamental

challenge that resonates across a plethora of scientific disciplines, from operations research to the expanding field of machine learning.

1.1 Nature and Origin of the TSP

In the realm of computational theory, problems are often categorized based on the time required to solve them. The question of whether P equals NP, that is, whether every problem for which a solution can be verified quickly can also be solved quickly, remains one of the most significant unsolved questions in computer science.

TSP is another classic example of an NP-hard problem, meaning that no polynomial-time solution is known for it, and it's unlikely that one exists. The origins of the TSP can be traced back to the 1800s. Mathematicians W.R. Hamilton and Thomas Kirkman introduced the general idea of the problem, though it was not yet named as such. Over the years, the TSP has not only been a subject of mathematical curiosity but has also found applications in various fields, from logistics to genetics. Its seemingly simple nature, juxtaposed with its computational complexity, has made it a benchmark problem in optimization and algorithm research.

The TSP briefly asks a given a set of cities and the pairwise distances between them, what is the shortest possible route that visits each city exactly once and returns to the origin city? Despite its seemingly straightforward premise, the TSP is a member of the NP-hard class of problems, indicating a computational complexity that grows exponentially with the number of cities involved. This complexity has turned the TSP from a practical challenge into a fundamental topic within the fields of theoretical computer science and combinatorial optimization. The TSP's deceptive simplicity is what makes it so intriguing. It might appear to be a straightforward

optimization problem; however, as the number of cities (or nodes) increases, the number of possible routes grows factorially. This exponential growth in possibilities is what places the TSP firmly in the NP-hard category. For instance, for just 20 cities, there are over 2.4 x 10^{18} ≈ 20! possible routes to consider.

1.2. Applications and Relevance

The fundamental nature of the TSP has led to its utilization across a diverse range of fields, with each field customizing the problem to suit its distinct needs and limitations. Some of the notable applications include:

1.2.1 Vehicle Routing: In logistics, the TSP aids in minimizing transportation costs by determining optimal routes for goods delivery, extending to the Vehicle Routing Problem (VRP) where multiple vehicles and additional constraints are considered.

1.2.2. Microelectronics Manufacturing: In the production of printed circuit boards, the TSP assists in determining the most efficient sequence for drilling holes, optimizing manufacturing processes.

1.2.3. DNA Sequencing: In bioinformatics, the TSP finds utility in ordering DNA fragments efficiently, where the "distance" between fragments represents the likelihood of one fragment following another, facilitating genome sequencing.

1.3 Mathematical Formulation of the TSP

Given a set of cities $C = \{c_1, c_2, ..., c_n\}$ and a distance matrix where dij represents the distance between city i and city j, the TSP can be mathematically formulated as:

Objective: Minimize the total distance traveled:

$$\min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} d_{ij} x_{ij}$$

Constraints:

- Each city is visited exactly once:

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \quad \forall i \in C$$

- No sub-tours (using subtour elimination constraints or cutting-plane methods).

Where $x_{ij} = 1$ if the path between city *i* and city *j* is taken, and $x_{ij} = 0$ otherwise.

Conceptually, TSP's solution appears to be a permutation of specific cities (or nodes) which shows that its naive solution's complexity is O(n!). Mathematically speaking, this means that the number of possible paths would give us a solution of (n-1)! coming from the initial premise that the first step and the last step will only be one of the cities (or nodes).

1.4 Proof that TSP is NP-Hard

To establish that the Travelling Salesman Problem is NP-Hard, we need to reduce an already known NP-Hard problem to prove it. In this case, Hamiltonian Cycle problem is used as the starting point and reduces it to the Travelling Salesman Problem.

For each instance of the Hamiltonian Cycle problem, which comes with an input graph:

G = (V, E), where V = vertices and E = edges

This can be transformed into an instance of the Travelling Salesman Problem by creating a new graph G' = (V', E'). This new graph will have a specified maximum cost, K, set to N (the number of vertices). The construction of G' is carried out as follows:

- For each edge e in E, assign a cost of c(e) = 1.

- Add the remaining edges, denoted as e', to make the graph complete. These new edges belong to E' and were not in the original graph G.
- Assign a cost of c(e') = 2 for these edges.
- Set the maximum cost K to the number of vertices N.

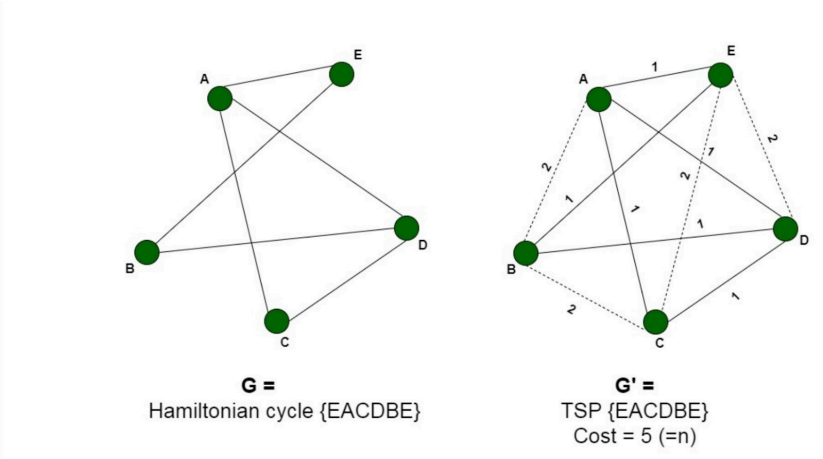

Figure 2: New and Old Hamiltonian Cycle

This construction can be performed in polynomial time, turning G into a complete graph G' with associated costs. This reduction can be verified through the following two statements:

1. Assuming the original graph G has a Hamiltonian Cycle, this cycle goes through all vertices V. Thus, it translates into a TSP solution in graph G' with a total cost of N. This is because it utilizes all original edges, each having a cost of c(e) = 1, and returns to the starting vertex.
2. Assuming the modified graph G' contains a TSP solution with cost K = N, this means that the TSP route covers all vertices and returns to the starting point. Because the total cost

sums up to N, it must necessarily use only the edges with a cost of 1 from the original graph E. This implies that it forms a Hamiltonian Cycle in the original graph G.

Therefore, if a Hamiltonian Cycle exists in graph G, then a corresponding TSP solution exists in the modified graph G' and vice versa. This shows that the Hamiltonian Cycle problem can be transformed into a TSP instance, confirming that TSP is NP-Hard.

2. Methodology To Approaches of Solving TSP

This paper will narrow its focus to analyze specific facets of algorithmic approaches. Through looking at historical trends, general algorithms, and case-sensitive techniques, this analysis aims to shed light on the evolving nature of optimality in TSP solutions.

In the historical exploration of TSP solutions, advancements in processing power have played a vital role in changing the landscape of problem-solving approaches. With the ever-increasing capabilities of technology, methods for calculating optimal solutions have not only become more feasible but have also redefined the very notion of optimality.

The second area of this exploration delves into general algorithms which impact the time complexity of reaching effective TSP solutions. These algorithms, including the nearest neighbor, 2-opt, 3-opt, and k-opt, represent a spectrum of techniques aimed at iteratively refining solutions. By adjusting the order of city visits, these algorithms navigate the solution space with the goal of converging towards more optimal routes, using a subset of rules. This paper will

examine the intricacies of each algorithm and their contributions to mitigating the TSP's inherent computational challenges.

In the third section of this paper, will be case-sensitive algorithms that offer guaranteed levels of optimality within a set time complexity. The Christofides Algorithm, a notable example, stands out for its ability to provide solutions within a polynomial time frame while ensuring a certain level of optimality. However, these algorithms come with specific criteria and constraints, requiring a nuanced understanding of the problem instances for their effective application.

3. Historical Growth of Solvable TSP Permutations

Over the years, the solutions to different TSP scenarios has notably advanced. A clear indication of these enhancements is evident in the growing scale of nontrivial instances successfully tackled, progressing from Dantzig, Fulkerson, and Johnson's resolution of a 49-city problem in 1954 to the solution of an 85,900-point problem 52 years later. As the trajectory of progress continues, TSP will lead to an increasing number of solutions for even higher complexity problems, but will require long periods of time for progress to be made.

Computational power Improvement

Moore's Law, postulated by Gordon Moore in 1965, posits that the number of transistors on a microchip would double approximately every two years, leading to an exponential increase in computing capabilities.

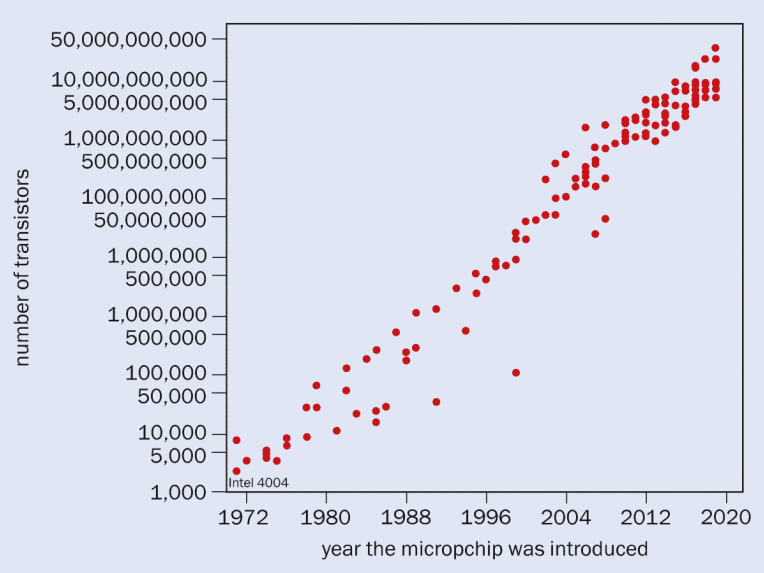

Figure 3: Graph demonstrating Computational Improvement Over Time

This increased computational powers has large implications for tackling complex computational problems such as TSP. The escalating number of computations per second that computers can handle paves the way for more sophisticated algorithms and heuristics to be practically implemented.

Eventually a typically inefficient algorithm of $O(n^2)$ will become optimally processisable as computational power increases and the amount of tasks able to be performed everyone second increases.

4. Proposed Optimization Methods and Analysis

The crux of these optimization methods are through the concept of heuristics. Heuristics are problem-solving approaches or methods that, while not guaranteed to be optimal or perfect, are employed to efficiently find satisfactory solutions, especially in situations where an exhaustive search or calculation may be impractical or too time-consuming. Heuristics provide practical, rule-of-thumb strategies that guide decision-making and problem-solving by narrowing

down the set of possible solutions, often through a process of trial and error. These approaches are valuable in situations where finding an exact solution is challenging or computationally expensive, allowing for quicker decision-making and problem resolution based on a set of reasonable, though not guaranteed, steps.

Greedy Algorithm, Nearest neighbor

The Greedy Algorithm and the Nearest Neighbor's Method are one of the most basic heuristic approaches frequently employed to find approximate solutions to problems, like the TSP. The Greedy Algorithm creates a solution incrementally by making isolated optimal choices at each step, with the hope that these choices will collectively lead to a total optimal solution. Despite its simplicity and computational efficiency, the Greedy Algorithm doesn't guarantee an optimal solution.

A variant of the Greedy Algorithm, the Nearest Neighbor's Method, also focuses on selecting the nearest unvisited city at each step but starts with a specified initial city. This method iteratively builds a tour by choosing the nearest unvisited city from the current one until all cities are visited, concluding by returning to the starting city, resulting in a path like the figure bellow.

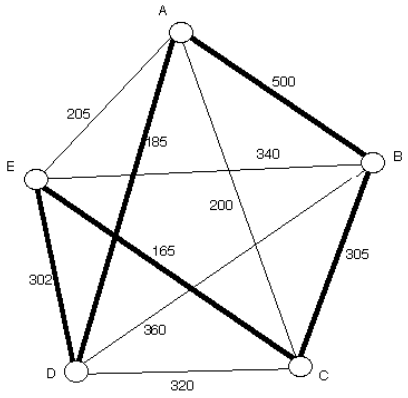

Figure 4: Nearest Neighbor Example

Similar to the Greedy Algorithm, the Nearest Neighbor's Method is straightforward and efficient, but its solution depends on the choice of the initial city. Additionally, it may encounter suboptimality issues similar to the Greedy Algorithm.

A weakness of this solution can be highlighted throughout the algorithm, in the scenario in which there are two options: city A and B, with an arbitrary 1 unit versus 2 units of space between the starting point. Postulated by this algorithm the path that will be taken is City A, 1 unit apart, however, if the nearest city next to City A is City C which is 100 units apart versus City B's nearest city being City D, 3 units apart the route that is taken by this algorithm will certainly have taken an extremely suboptimal route.

Despite their drawbacks, these algorithms offer advantages in terms of simplicity and efficiency, especially when tackling extremely large sets of cities. The algorithm's time complexity is generally $O(n^2)$, where n represents the number of cities, as the process of finding the nearest city involves iterating through the list of unvisited cities, resulting in a quadratic time complexity. While providing practical and quick solutions to optimization problems, these algorithms are limited, as they do not provide guaranteed most optimal solutions.

K-Opt

Another algorithmic method used to address TSP, would be the K-Opt method.

2-Opt

The fundamentals of K-Opt, essentially builds off the 2-Opt algorithm. Originally introduced by G. A. Croes in 1958, this algorithm seeks to optimize solutions through improvements in intersection points.

It stems from the concept that tours featuring overlapping edges are not optimal. The 2-opt algorithm evaluates all potential two-edge swaps, executing the swap only when it leads to an enhanced tour.

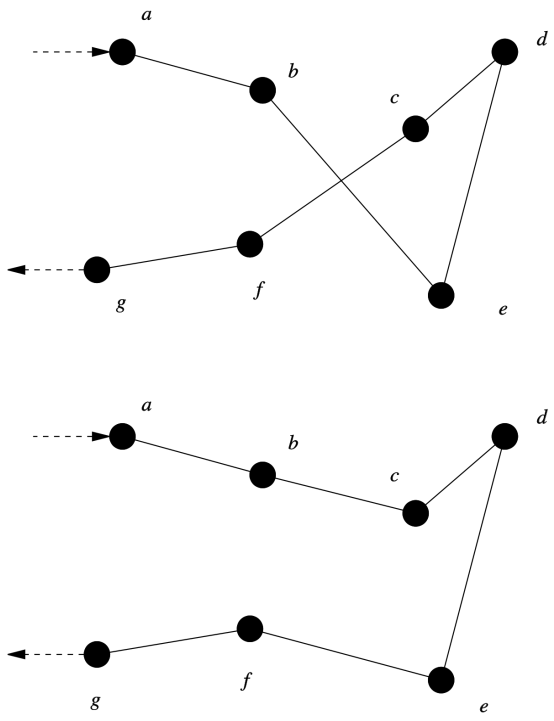

Figure 5: 2-opt Example

The primary idea of the 2-opt algorithm lies in iteratively exploring potential edge swaps to improve the path. The algorithm examines all pairs of edges in the tour and checks whether swapping them would lead to a shorter tour. In the worst case, it needs to consider all possible pairs.

For a tour with n cities, there are (n-1) edges. The 2-opt algorithm considers each pair of edges, resulting in a nested loop structure. The outer loop runs for (n-1) iterations, and for each iteration, the inner loop also runs for (n-1) iterations.

Therefore, the total number of pairwise comparisons made by the 2-opt algorithm is proportional to $(n-1) \times (n-1)$, or $(n-1)^2$. In asymptotic notation, this simplifies to $O(n^2)$ because lower-order terms and constants are dropped when analyzing time complexity.

The essence of the k-opt algorithm is derived by evaluating rather than just 2 edges, a general k amount of edges for increasing accuracy.

For the general k-opt algorithm by considering all combinations of k edges in the tour. For a tour with n cities, there are (n-k+1) edges to choose from. The algorithm systematically explores all combinations of k edges using nested loops. The outer loop runs for (n-k+1) iterations, and for each iteration, (k-1) inner loops run for (n-k+1) iterations each.

Therefore, the total number of combinations considered by the k-opt algorithm is proportional to $(n-k+1) \times (n-k+1) \times ... \times (n-k+1)$, k times.

In terms of time complexity, the k-opt algorithm can be expressed as $O((n-k+1)^k)$. Which simplifies to $O(n^k)$.

As the value of k increases in a k-opt algorithm, the potential for achieving a more optimal solution also increases. This is because higher-order k-opt algorithms explore a broader space by considering a greater number of edges in each operation. The flexibility provided by these larger k values enables the algorithm to escape the fallacy of greedy local solutions and uncover configurations that lead to shorter path lengths. However, it's important to acknowledge that the pursuit of higher optimality comes with an associated increase in computational complexity.

5. Case Sensitive Optimization Methods and Analysis

Christofides Algorithm

Christofides Algorithm is an approximation algorithm designed to solve the metric TSP with the guarantee of a performance ratio of $\frac{3}{2}$. The process of the algorithm goes as so:

1. **Minimum Spanning Tree (MST):**

The algorithm starts by constructing a MST from the given set of cities, a tree that spans all the cities while minimizing the total edge weight. This tree ensures connectivity between all cities with the least possible total distance.

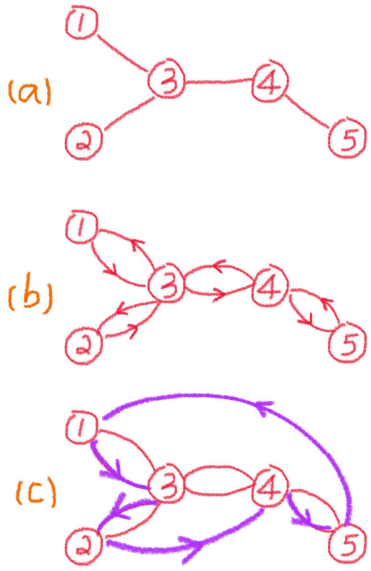

Figure 6: Minimum Spanning Tree Example

2. **Matching Odd-Degree Vertices:**

Next, the algorithm identifies vertices in the MST that have an odd degree, meaning they are connected to an odd number of edges. These odd-degree vertices become potential endpoints for an Eulerian circuit, a closed walk that covers every edge exactly once.

3. **Minimum Weight Perfect Matching:**

The algorithm creates a minimum weight perfect matching for the odd-degree vertices, matching odd degrees with other points for the lowest distance possible. This matching connects these vertices in a way that, when combined with the MST, ensures that every vertex in the graph has an even degree.

4. **Eulerian Circuit:**

By combining the MST and the minimum weight perfect matching, the algorithm constructs an Eulerian circuit in the augmented graph. This circuit visits each edge once, providing a systematic way to traverse all cities.

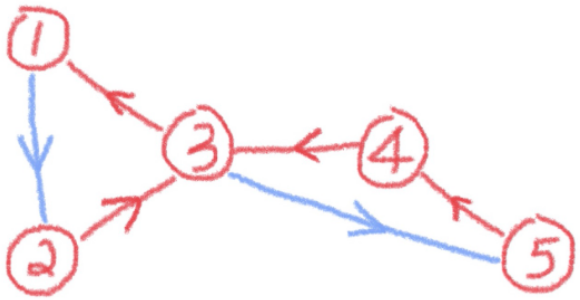

Figure 7: Eulerian Circuit Example

5. **Shortcutting:**

While traversing the Eulerian circuit, the algorithm avoids revisiting vertices by "shortcutting" repeated visits, ensuring each city is visited only once. This process transforms the Eulerian circuit into a Hamiltonian circuit, visiting each city exactly once.

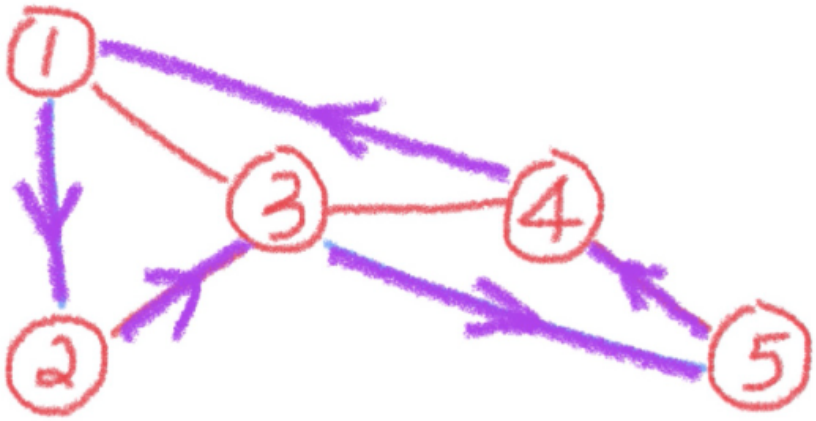

Figure 8: Shortcutting Example

6. **Tour Shortening:**

The final step involves shortening the Hamiltonian circuit by removing redundant edges, further optimizing the total length of the tour

The overall time complexity of the Christofides Algorithm is influenced by the individual steps of the algorithm. Firstly, the construction of the Minimum Spanning Tree. Common algorithms like Kruskal's or Prim's have time complexities of $O(E \times \log V)$ and $O(V^2)$, respectively, where E is the number of edges and V is the number of vertices. Afterward, identifying and matching odd-degree vertices, which involves traversing the vertices of the MST, takes O(V) time. The next steps of finding a minimum weight perfect matching among the odd-degree vertices, often accomplished using algorithms like Blossom or Hungarian, has a time complexity ranging from $O(V^3)$ to $O(V^4)$. Constructing the Eulerian circuit from the augmented graph takes linear time O(E). Finally, the processes of shortcutting and tour shortening, both involving traversing the edges of the Eulerian circuit, have linear time complexities with respect to the number of edges, denoted as O(E).

Combining these steps, the overall time complexity of the Christofides Algorithm can be expressed as the sum of the complexities of individual steps:

$O(E \log V) + O(V) + O(V^3) + O(E)$

As the highest degree in this set would be V^3, the simplified time complexity results in $O(V^3)$.

3/2 Optimal Proof

The algorithm ensures that the cost of the solution it produces is within $\frac{3}{2}$ of the optimum. In order to prove this, consider the optimal traveling salesman tour (denoted as C). Removing an edge from C results in a spanning tree, and its weight is at least that of the minimum spanning tree (denoted as T). Therefore, $w(T) \leq w(C)$. Next, enumerate the vertices of the tour in cyclic order around C and divide C into two sets of paths based on whether the first path vertex in cyclic order has an odd or even number. Each set of paths corresponds to a perfect matching of the vertices, and the weight of this matching is at most equal to the weight of the paths. Since these sets of paths partition the edges of C, one of them has at most half the weight of C. Due to the triangle inequality, the corresponding matching also has a weight that is at most half the weight of C. The minimum-weight perfect matching, denoted as M, cannot have a weight greater than this, so $w(M) \leq \frac{w(C)}{2}$. By adding the weights of T and M, we obtain the weight of the Euler tour, which is at most $3\frac{w(C)}{2}$. Importantly, due to the triangle inequality, the process of shortcutting does not increase the weight, ensuring that the weight of the output remains at most $\frac{3}{2}w(C)$.

With a fixed length of time and a consistent level of optimal the algorithm is a very good one, however, only effective when the distances between cities satisfy the triangle inequality,

which means that the direct route between any two cities is always the shortest route, therefore this cannot be used for every case.

6. Conclusion and Reflection

In the exploration of TSP solutions, our analysis has demonstrated various approaches, each with its unique set of advantages and drawbacks. The historical lens reveals solution based on computational advancements, highlighting the waiting game for increased processing power to render larger TSP instances more manageable. While this strategy aligns with the trajectory of technological progress, it places a temporal constraint on solution scalability.

General algorithms, such as the widely employed 2-opt and k-opt, provide consistent and efficient solutions by iteratively refining routes. However, the trade-off lies in their tendency to sacrifice accuracy for efficiency. These algorithms offer practical solutions in a variety of scenarios but may fall short in achieving the pinnacle of optimization.

On the other hand, case-sensitive algorithms, exemplified by the Christofides Algorithm, postulate a guaranteed level of optimality within a known time complexity. However, their applicability is contingent on specific conditions being met, limiting their versatility in addressing diverse TSP instances.

In the real-world crucible of companies like Waze or Google, faced with complex routing problems involving myriad variables, the approach is multifaceted. Hundreds of factors are considered, and the choice of methodology is a nuanced decision. While general algorithms may provide quick and satisfactory solutions for real-time navigation, there remains a constant push towards refining these algorithms to deliver increasingly optimal routes.

Overall, the pursuit of optimal solutions to the TSP is an ongoing journey, marked by a delicate balance between computational efficiency, solution optimality, and real-world applicability. As we navigate this complex terrain, a comprehensive understanding of the strengths and limitations of historical, general, and case-sensitive algorithms paves the way for informed decision-making in the quest for efficient and effective TSP solutions.

Works Cited

"2-Opt." *Wikipedia*, 11 Jan. 2024. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=2-opt&oldid=1194969927.

"11 Animated Algorithms for the Traveling Salesman Problem." *STEM Lounge*, 28 Dec. 2019, https://stemlounge.com/animated-algorithms-for-the-traveling-salesman-problem/.

"Christofides Algorithm." *Wikipedia*, 16 Jan. 2024. *Wikipedia*, https://en.wikipedia.org/w/index.php?title=Christofides_algorithm&oldid=1196163575.

Davis, Adam. "Traveling Salesman Problem With the 2-Opt Algorithm." *Medium*, 19 May 2022, https://slowandsteadybrain.medium.com/traveling-salesman-problem-ce78187cf1f3.

"Eulerian Path and Circuit for Undirected Graph." *GeeksforGeeks*, 13 June 2013, https://www.geeksforgeeks.org/eulerian-path-and-circuit/.

McKenzie, James. "Moore's Law: Further Progress Will Push Hard on the Boundaries of Physics and Economics." *Physics World*, 20 June 2023, https://physicsworld.com/a/moores-law-further-progress-will-push-hard-on-the-boundaries-of-physics-and-economics/.

"Proof That Traveling Salesman Problem Is NP Hard." *GeeksforGeeks*, 3 June 2020, https://www.geeksforgeeks.org/proof-that-traveling-salesman-problem-is-np-hard/.

"Subset Sum Problem Explained (Dynamic Programming)." *FavTutor*, https://favtutor.com/blogs/subset-sum-problem. Accessed 22 Jan. 2024.

"Traveling Salesman Problem and Approximation Algorithms." *Traveling Salesman Problem and Approximation Algorithms*, https://bochang.me/blog/posts/tsp/. Accessed 22 Jan. 2024.

"Travelling Salesman Problem | Greedy Approach." *GeeksforGeeks*, 31 July 2020, https://www.geeksforgeeks.org/travelling-salesman-problem-greedy-approach/.

TSP History Home. https://www.math.uwaterloo.ca/tsp/history/index.html. Accessed 22 Jan. 2024.