An Evaluation of the Traveling Salesman Problem

Connor Chase
Department of Computer Science
California Polytechnic University, Pomona
Pomona, United States
chchase@cpp.edu

Harrison Chen
Department of Computer Science
California Polytechnic University, Pomona
Pomona, United States
hbchen@cpp.edu

Alex Neoh
Department of Computer Science
California Polytechnic University, Pomona
Pomona, United States
neoh.alex@gmail.com

Maximum Wilder-Smith
Department of Computer Science
California Polytechnic University, Pomona
Pomona, United States
mwildersmith@cpp.edu

***Abstract*—**The Traveling Salesman Problem is a popular NP problem in combinatorial optimization which asks, given a list of cities, what is the shortest route a salesperson can travel from one city to another, while having a Hamiltonian cycle. This paper aims to discuss the research of the Traveling Salesperson Problem's theory along with the implementation of exact and heuristic algorithms.

***Keywords*— Traveling Salesman Problem, P=NP, Heuristics, Optimization**

I. INTRODUCTION

The Traveling Salesman Problem, or TSP, is one of the most intensively studied problems by computer scientists. The concept of the Traveling Salesman Problem was first proposed in a board game created by mathematicians W.R. Hamilton and Thomas Kirkman [1]. The main objective of this game was to find the Hamitonian cycle, which is going through every point on the game board exactly once and ending at the starting point. This game concept became known as the Traveling Salesman Problem in 1930 when mathematicians started studying this problem [1]. Since then, there were a large number of exact and heuristic algorithms developed to solve the problem. Traveling Salesman Problem can be modelled as a graph problem where the cities are the vertices and the path are the edges. The graph of the Traveling Salesman Problem could be symmetric, where the graph is undirected, or asymmetric, where the graph is directed.

The Traveling Salesman Problem can be solved with exact solutions such as a brute force approach or the Held-Karp algorithm [2]. These exact algorithms require solving an optimization problem for the most optimal result. Unless P = NP, the Traveling Salesman Problem's exact solutions are slower than polynomial time. Due to the slow nature of exact algorithms, even smaller problems would require a large amount of time. More approaches for exact solutions to the Traveling Salesman Problem include branch and bound algorithms and use of linear programming techniques, however these still suffer from the P = NP constraint making them slower than polynomial time.

The Traveling Salesman Problem (TSP) is a NP problem with the TSP decision and search problem both being NP complete. They are both NP Complete because the solutions of these can be tested in polynomial time, yet a solution cannot be found in polynomial time. For example, in a decision problem, a solution is given and the question would be to find out if it is less than a certain distance. Verifying the question would take polynomial time because the answer could be found by going through the cities in the solution and adding up the costs. However, the TSP optimization problem is NP- hard but not NP, which means that the solution and the verification cannot be done in polynomial time. For example, there hasn't been an algorithm found yet that can find the solution of the most optimal cycle in polynomial time. In order to verify that solution, there is no way to do it except by going through an algorithm with longer than polynomial time. However, there are heuristic algorithms that may solve the Traveling Salesman Problem in polynomial time.

Such heuristic algorithms include the Nearest Neighbor algorithm, Ant Colony Optimization, or the Christofides algorithm [2]. A heuristic requires an algorithm to solve an optimization problem with the most optimal speed, while risking that the solution may not be the absolute best. A larger problem can be solved with these algorithms because using a heuristic means shorter run time. This is useful as there are many cases where it is better to lose some efficiency for a massive decrease in run time.

This problem's popularity extends beyond research into real life applications as it could be used for scheduling, logistics, and the manufacture of microchips. For example, the Traveling Salesman Problem is applied with package delivery planning. The houses are the “vertices” and the company must find the shortest path for delivery and returning to the delivery center in order to save the most money on delivery costs.

II. ALGORITHMS

A. Brute Force Algorithm

The brute force algorithm is the easiest algorithm to implement for Traveling Salesman Problem exact solutions. However, it also has the slowest time complexity because the algorithm requires every permutation of a solution to be checked. When every solution has been processed, the cheapest one is chosen. The brute force algorithm functions as follows, with G being the graph representing theTSP:

1. Pick a starting point vertex in G
2. Calculate the number of tours from the starting point and mark each tour.
3. Calculate the distance of each tour.
4. Select the cheapest tour, which would be the most optimal solution.

The brute force algorithm has exactly $n!$ permutations that need to be checked. Thus the time complexity is $O(n!)$. Although the brute force algorithm is the slowest exact algorithm, there are many improvements for it that can achieve times that are less than *n!*, such as the use of branch and bound, linear programming, and dynamic programming techniques.

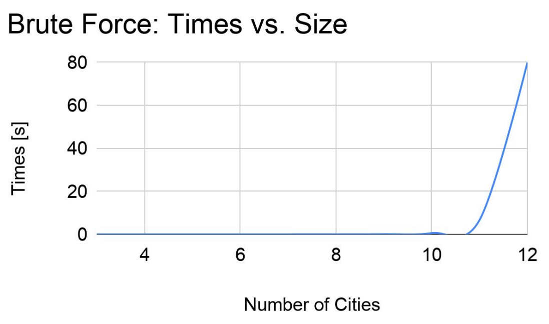

Figure 1.
Time in milliseconds vs. the number of cities in a Traveling Salesman Problem.

Being of factorial time complexity, the algorithm can process a very small Traveling Salesman Problem in a reasonable amount of time. This is evident in Figure 1, where it took almost 80 seconds to find the path for just 12 cities. This just demonstrates how ineffective the factorial time complexity is. On top of this, as the brute force approach runs recursively, the system can easily run out of memory errors. After running 11 cities the Java heap size needed to be increased to 22 GB to run for 12 cities. This algorithm is expressed in the following pseudocode:

```
paths = {all possible paths}
optimal = min( paths.map( path => C(path),
          path))
return optimal
```

B. Held- Karp Algorithm

The Held-Karp algorithm was developed by Held and Karp in 1962, and independently by Bellman in the same year [3]. The Held-Karp algorithm uses dynamic programming to find more efficient solutions to a series of sequencing problems, including the Traveling Salesman Problem. As this algorithm uses dynamic programming, it offers a solution to a sub-problem that can be recursively called to get the solution to the overall problem. This is done by first creating a distance matrix from each city/node to another city/node. For a situation of S = {1,2,3,..., *n*} where S is the set of *n* cities, this distance matrix would be *n* x *n* in size. A minimum cost function is then defined as *C(U, l)* where $U \subseteq S$ such that all the cities in *U* are visited on the way to *l* where $l \in S$. *C(U, l)* is then defined such that:

$$(1)\quad \begin{array}{ll}\text{(a) } (n(U) > 1): & C(\{l\}, l) = a_{1l}, \text{ for any } l \\ \text{(b) } (n(U) > 1): & C(\{S\}, l) = \min_{m \in S-1}(C(S - l, m) + a_{ml}), \\ & \text{where } m \text{ is the city that before } l\end{array}$$

If the original city is *l* and it is returned to after the journey, then the shortest route would have a cost of

$$(2)\quad e = \min_{l \in \{2,3,\dots,n\}}(C(\{2,3,\dots,n\}, l) + a_{l1})$$

The first step of the Held-Karp algorithm uses (1) to recursively compute all values for C(*S*, *l*) and (2) to compute the shortest possible circuit. The Held-Karp paper also states that a permutation of cities is optimal if and only if:

$$(3)\quad e = C(\{2, 3, \dots, n\}, i_n) + a_{i_n 1}$$

and

$$(4)\quad C(\{i_2, i_3, \dots, i_p, i_{p+1}\}, i_{p+1}) = C(\{i_2, i_3, \dots, i_p\}, i_p) + a_{i_p i_{p+1}}$$

The second step of the Held-Karp algorithm then uses (3) and (4) to find the optimal path to visit each city [3]. The algorithm first find i_n then i_{n-1} and so on until the optimal path is found. This technique allows for an improvement on the brute-force approaches O(*n!*) time complexity. Instead the first step involves

$$(\sum_{k=2}^{n-1} k(k-1)\binom{n-1}{k}) + (n-1) = (n-1)(n-2)2^{n-3} + (n-1)$$

operations while the second step involves

$$\sum_{k=2}^{n-1} k = \frac{n(n-1)}{2} - 1$$

operations. This yields an approximate time complexity of O($2^n n^2$) [2]. While this time complexity is exponential, it is still much better than the factorial time complexity of the brute force approach, and offers an exact ultimate solution.

C. Christofides Algorithm

The Christofides algorithm was developed by Nicos Christofides in 1976 [4]. The Christofides algorithm was designed with the purpose of finding approximate solutions for the Traveling Salesman Problem. It has some prerequisites for the situation, such as being symmetric and obeying the triangle inequality. The triangle inequality states that for every three vertices *x*, *y*, and *z*, it should be the case that for the weight of the edges, *w*, $w(x\,y) + w(y\,z) \geq w(x\,z)$ [4]. To solve the Traveling Salesman Problem, start out by letting G be the symmetric complete graph. The pseudocode of Christofides algorithm is as follows:

1. Construct a minimum spanning tree, T, from G
2. Next find all the odd degree vertices in T and place them in set O. A vertex with an odd degree means that an odd number of edges are connected to the vertex
3. Connect odd degree vertices using the minimum weight perfect matching adding the edges to the matched odd degree vertices, which would create an Euler cycle for T. Every vertex in T should have an even degree after this step
4. Remove repeated edges from vertices with a degree greater than 2 to create a Hamiltonian cycle

To this day, Christofides algorithm has the best approximation ratio that has been proven for the Traveling Salesman Problem on general metric systems though there are some special cases where other algorithms are better. It is an approximation algorithm because the solution is guaranteed to be within a factor of $\frac{3}{2}$, but is able to achieve this results in far less time than an exact algorithm. Often this time complexity comes down to the algorithm used to find the minimum spanning tree [5].

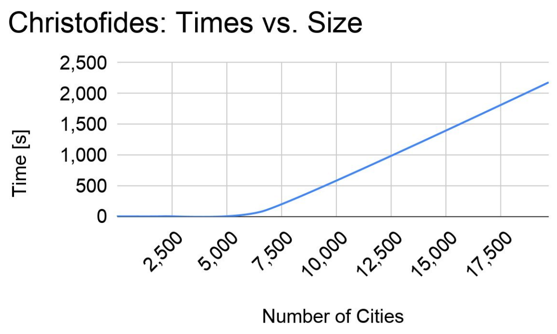

Figure 2.
This graph shows the time in seconds it takes to run the Christofides algorithm on a given number of cities.

With a time complexity of $O(n^4)$, this algorithm can process larger Traveling Salesman Problem situations with greater efficiency than the brute force method [5]. However this comes at the cost of accuracy as Christofides only finds an approximate solution as opposed to the optimal.

D. Ant Colony Optimization

The Ant Colony Optimization algorithm, or ACO, was proposed by Marco Dorigo in 1992 in his PhD thesis. ACO is an algorithm that searches and finds the most optimal solution for the Traveling Salesman Problem graph based on the behavior of ants between their colony and a food source. The algorithm is a probabilistic technique due to the ants wandering randomly. The idea is that as ants search for food, they leave a pheromone trail for other ants in the colony to follow. However, the pheromone trail starts to evaporate as time goes on. The longer it takes for an ant to travel a path, the more the pheromone trail evaporates, leaving a weaker attractive strength. Thus, the shorter the path to the food source is, the stronger the attractive strength due to a greater number of ants following the stronger scent [6].

Given an n-city Traveling Salesman Problem with distances d_{ij}, artificial ants are distributed to these n cities randomly. Based on the pheromone trail remaining on the paths, each ant will choose the next city to visit. The main differences between artificial ants and real ants are that artificial ants will not visit cities that they have already visited, and they can also know the distances between two cities, unlike real ants. Thus, the probability that city *j* is selected by ant *k* to be visited after city *i* could be written as follows:

$$(1)\quad p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{s \in allowed_k} [\tau_{is}]^{\alpha} \cdot [\eta_{is}]^{\beta}} & j \in allowed_k \\ 0 & \text{otherwise} \end{cases}$$

where τ_{ij} is the intensity of pheromone trail between cities *i* and *j*, α the parameter to regulate the influence of τ_{ij}, n_{ij} the visibility of city *j* from city *i*, which is always set as 1/ d_{ij} (d_{ij} is the distance between city *i* and *j*), *β* the parameter to regulate the influence of n_{ij} and allowed *k* the set of cities that have not been visited yet, respectively.

After *n* iterations, every ant completes a trip. The ants with shorter trips should leave more pheromone than those with longer trips. Therefore, the trail levels are updated as on a trip each ant leaves a pheromone quantity given by *Q/Lk*, where *Q* is a constant and *Lk* the length of its trip. On the other hand, the pheromone will evaporate as time goes by. Then the updating rule of τ_{ij} could be written as follows:

$$(2)\quad \tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$(3)\quad \Delta\tau_{ij} = \textstyle\sum_{k=1}^{l} \Delta\tau_{ij}^{k}$$

$$(4)\quad \Delta\tau_{ij}^{k} = \begin{cases} Q/L_k & \text{if ant } k \text{ travels on edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

where τ is the iteration counter, *ρ ϵ [0, 1]* the parameter to regulate the reduction of τ_{ij}, $\Delta\tau_{ij}$ the total increase of trail level on edge *(i, j)* and $\Delta\tau_{ij}^{k}$ the increase of trail level on edge *(i, j)* caused by ant *k*[5].

E. Nearest Neighbour Algorithm

The Nearest Neighbour algorithm, or NN, is an approximation algorithm and was one of the first algorithms used to solve the Traveling Salesman Problem. Because it is a greedy algorithm, the solutions may not be optimal, however, it is easy to implement and quick to run with a worst case time complexity of $O(n^2)$ and a worst case space complexity of $O(n)$. The pseudocode is as follows:

1. Set all the cities to unvisited
2. Select an origin city
3. Find the shortest edge
4. If the city is unvisited, connect that city to the previous city
5. If every city is visited then go back to the origin city, otherwise go back to step 3

This algorithm's optimality is around 25% of the Held-Karp's lower bound. The cost of the path depends on where the origin city is for the Nearest Neighbor algorithm. The Repetitive Nearest Neighbor, or RNN, says to try every city as an origin, then select the shortest path. The best path found will be better than at least $\frac{n}{2} - 1$ other tours, with *n* being the number of cities [7].

III. REAL LIFE APPLICATIONS

A. Wiring

The Traveling Salesman Problem can be used to help find the most efficient way to route power lines through rural areas in between major power grids. This setup would follow very closely to the original TSP with cities acting as the vertices with the distance between them as the edge weights. Minimization of the distance to visit all the cities would result in decreased construction costs, and more efficient power delivery. Not only does TSP help with large scale electrical grids, but it can optimize the layouts of small circuits. In this case varying wire distances can affect a circuit's heat, resistance and operating speeds.

B. Deliveries

Once again, the original TSP, of finding the most efficient method of travel between multiple stops, can be used to optimize long distance deliveries. In this use case, the vertices would not only be the cities for delivery, but the gas stations and rest stops between them. The weights for the edges would not only be distance, but could be based on a heuristic that takes into account travel conditions, distance, and gas prices to optimize for not only the shortest route, but the cheapest as well.

C. Aerospace

The Traveling Salesman Problem can be used to find the most efficient flight paths for planes that need to stop at multiple cities. Another example would be the Starlight Interferometer Program, which is a program designed to optimize flight paths for satellites that need to point at far away celestial objects [8]. In this situation the vertices of the TSP problem would be the positions needed to point at stars, while the cost to travel between the positions in orbit is the edge cost.

IV. COMPARISONS

A. Brute Force vs. Christofides

In comparing the brute force approach to the Christofides solutions to the Travelling Salesman Problem, it is quite clear that Christofides runs much faster and uses less space.

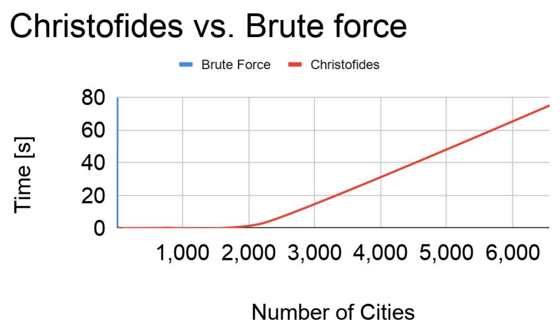

Figure 3.
This graph compares the runtimes for the brute force approach and the Christofides approach. Note: Brute force is the blue line on the left.

As Figure 3 shows, the brute force approach takes so much memory and time it becomes a nearly vertical line in comparison to the Christofides algorithm. This also shows that a brute force approach on a situation with more than 20 or so cities is impractical for most cases. While Christofides does not always guarantee the absolute optimal solution, it does guarantee a pretty good solution that is at most 1.5 times the optimal distance. While this can be costly in some situations, such as circuit board design, this increased distance is likely to be worth the time and memory cost of running the brute force algorithm. Especially since if the system that performed the presented computations were to run with 24 cities, the computation would take roughly $3.280\text{x}10^9$ years, or roughly the age of our sun. Even in situations where it is absolutely necessary to get the exact optimal solution, it is inadvisable to run the brute force approach and instead run the Held-Karp algorithm or some other optimized exact algorithm. Given Held-Karp's theoretical time complexity of $O(2^n\ n^2)$, and the given times for this system, the Held-Karp algorithm should be able to calculate 24 in approximately 26.5 minutes. While this is much better, it is still exponential time and unrealistic for thousands of vertices.

B. Exact Approaches

While all of the exact approaches have incredibly inefficient time and space complexities, the worst by far is the brute force approach. The upside of the brute force approach is that it is the easiest to implement and understand. However, it is still best to implement the branch and bound technique and save even a few possible path calculations that need to run all possible paths in the direct brute force approach. This is due to the fact that although the branch and bound methodology has at worst the time complexity of a direct brute force, it often runs faster. This is due to the pruning which can save up to *(n-1)!* paths to compute for a given situation. Beating the worst case of both of these approaches is the Held-Karp which runs in exponential time rather than factorial. While this is still horribly impractical and inefficient, it is much better than factorial time. Ideally an exact approach would not be needed as there is no known exact algorithm that runs in polynomial time on a deterministic Turing machine. This means the Traveling Salesman Problem is an NP-hard problem. As section IV. A. mentions it is often preferably to run a heuristic algorithm due to time and space complexities.

C. Heuristic Approaches

Among the heuristic algorithms, the most commonly used to this day is the Christofides method. It has been considered the most effective ever since its development in 1976. There are still some specific cases in which some methods may take less time. Though for most general cases Christofides will still be the most effective since its solution is within $\frac{3}{2}$ of the optimal solution.

V. CONCLUSION

In conclusion, unless absolutely needed, it is better to get an approximately optimal solution via an approach like Christofides, Nearest Neighbor algorithms or even ant colony simulation than to run a brute force approach for an exact solution. Even if an exact solution is needed, it is more efficient to run an algorithm like branch and bound or Held-Karp than a direct brute force. However, as this is presently an NP-Hard problem, any exact solution will not run in polynomial time on a deterministic Turing machine. It is better for the sake of speed and memory to run an approximate or heuristic algorithm to get a pretty good solution than to get the absolute best.

Given the numerous applications and importance of this problem, a solution in P would be incredibly beneficial, but depending on the approach's time complexity, it may still be better to find approximate solutions for certain situations.

REFERENCES

[1] Flood, Merrill M. "The Traveling-Salesman Problem." Operations Research, vol. 4, no. 1, 1956, pp. 61–75. JSTOR, www.jstor.org/stable/167517.

[2] Bellmore, M., and G. L. Nemhauser. "The Traveling Salesman Problem: A Survey." *Operations Research*, vol. 16, no. 3, 1968, pp. 538–558. *JSTOR*, www.jstor.org/stable/168581.

[3] Held, Michael, and Richard M. Karp. "A Dynamic Programming Approach to Sequencing Problems." Journal of the Society for Industrial and Applied Mathematics, vol. 10, no. 1, 1962, pp. 198–210. JSTOR, www.jstor.org/stable/2098806.

[4] Gendreau, Michel, et al. "An Approximation Algorithm for the Traveling Salesman Problem with Backhauls." Operations Research, vol. 45, no. 4, 1997, pp. 639–641. JSTOR, www.jstor.org/stable/172059.

[5] Golden, B., et al. "Approximate Traveling Salesman Algorithms." Operations Research, vol. 28, no. 3, 1980, pp. 694–711. JSTOR, www.jstor.org/stable/170036.

[6] Yang, Jinhui, et al. "An Ant Colony Optimization Method for Generalized TSP Problem." Progress in Natural Science, vol. 18, no. 11, 2008, pp. 1417–1422., doi:10.1016/j.pnsc.2008.03.028.

[7] Gregory Gutin, Anders Yeo and Alexey Zverovich, *Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP*. Discrete Applied Mathematics 117 (2002), 81-86.

[8] Riley Duren, Oliver Lay, Matt Wette, "The Starlight Interferometer Architecture and Operational Concepts", Pasadena, California, United States of America, NASA Jet Proportion Laboratory, Accessed: Dec. 6, 2019. [Online]. Available: https://trs.jpl.nasa.gov/bitstream/handle/2014/9712/02-1880.pdf?sequence=1