CSE 462 Project Report

Travelling Salesman Problem

Report Prepared by:

Sumonta Nandy Amit - 1805069
Sheikh Hasanul Banna - 1805094
Utchchhwas Singha -1805100
Partho Kunda - 1805107
Udayon Paul Dhrubo - 1805109

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Contents

1 Introduction

1.1 Problem Definition

TSP is an NP- hard problem where a salesman needs to travel from a starting location, visit all the cities exactly once and return to his initial location. This is a minimization Problem. A graph formulation for the problem could be as follows:

- Represent each city as a vertex
- Each city is connected to all the other cities by edges (complete graph)
- Cost of traveling between cities is represented by weights of the edges.
- A tour is a hamiltonian cycle of the graph. Cost of the tour is sum of the edges in the hamiltonian cycle. Minimizing the cost is the goal of the problem.

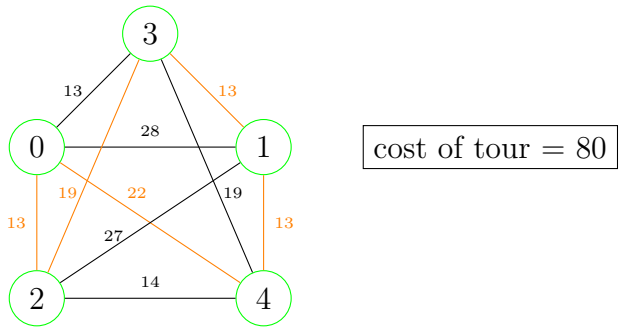


Figure 1: Travelling Salesman Problem Graph formulation.

Formal Definition:

- Cost: c_{ij} = cost of travelling between cities i and j
- Number of cities: n
- Cyclic permutation of cities: $k(1), k(2), \ldots k(n), K(1)$ where each city j is listed as a unique node $k(j)$.
- Then, cost of the tour is
  $C(\pi) = C_{k(n)k(1)} + \sum_{i=1}^{n-1} C_{k(i)k(i+1)}$
- Goal of the problem: Find permutation π such that $C(\pi)$ is minimum

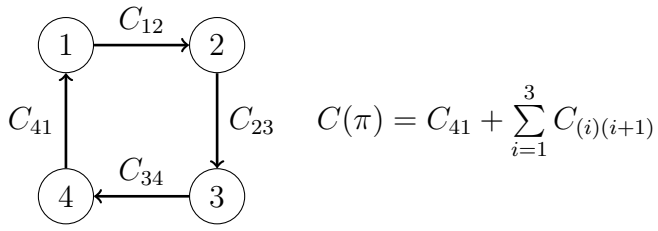


Figure 2: Cost of a Permutation

Integer Linear Programming Formulation:

- Variables: c_{ij} = cost of travelling between cities i and j

  $x_{ij} = \begin{cases} 1 & \text{if path goes between i and j} \\ 0 & \text{otherwise} \end{cases}$

- minimize cost: $\min \sum_{i=1}^{n} \sum_{\substack{j \neq i \\ j=1}}^{n} c_{ij} x_{ij}$

  subject to:

- one incoming edge per vertex: $\sum_{\substack{i \neq j \\ i=1}}^{n} x_{ij} = 1 \qquad [\forall j = 1, 2, \ldots n]$

- one outgoing edge per vertex: $\sum_{\substack{j \neq i \\ j=1}}^{n} x_{ij} = 1 \qquad [\forall i = 1, 2, \ldots n]$

- subtour elimination: $\sum_{i \in Q} \sum_{\substack{j \neq i \\ j \in 1}} x_{ij} \leq |Q| - 1 \quad \forall Q \subset \{1, 2, \ldots n\}, |Q| \geq 2$

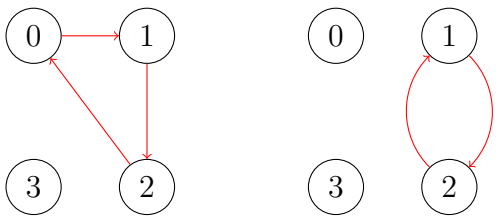


Figure 3: Left: $\sum_{i \in Q} \sum_{\substack{j \neq i \\ j \in 1}} x_{ij} = 3 > |\{0, 1, 2\}| - 1$. Right: $\sum_{i \in Q} \sum_{\substack{j \neq i \\ j \in 1}} x_{ij} = 2 > |\{1, 2\}| - 1$.
Subtour exist for both cases, so they will be eliminited due to subtour elimination.

1.2 Problem Variations

- **Optimization Version**: The goal here is to find the tour with minimum cost. This is NP-hard, but not NP-complete. Because for a given instance, there is no known efficient way to check if it is the optimal tour.

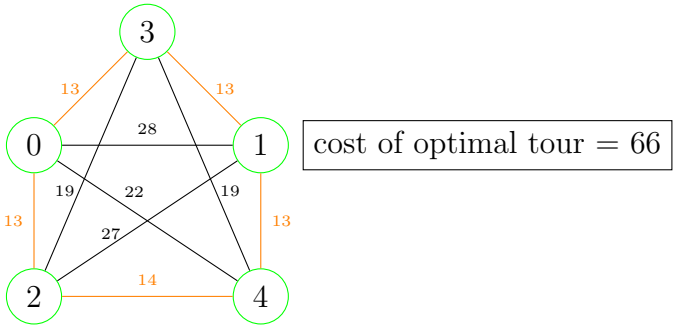


Figure 4: TSP Optimization Version

- **Decision Version**
  - for a certain value k, is there a tour with cost $\leq k$?
  - NP-Complete.[NP-hard and a given instance can be verified in polynomial time]

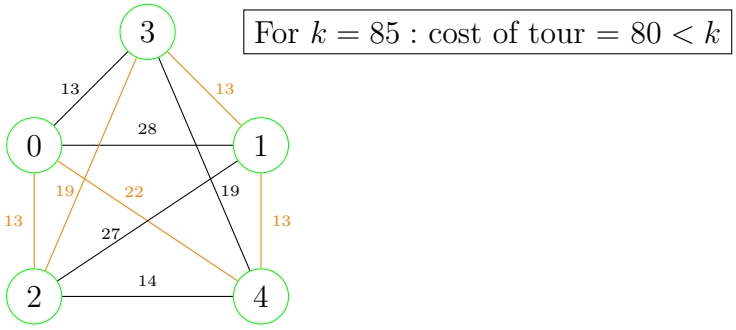


Figure 5: TSP Decision Version

1.3 Decision - TSP is NP-complete

TSP is in NP

We can verify for a given tour $\{V_1, V_2, \ldots V_n, V_{n+1}\}$

- Check if all the cities are visited exactly once [O(n)]
- Check if tour starts and ends with the same city [$V_{n+1} = V_1$.O(1)]

- Check if total cost is within the given value k [O(n)]

Total time complexity of verification is O(n).

TSP is Np-hard

We will reduce Hamiltonian cycle problem(NP-hard) to TSP

- **Hamiltonian Cycle Problem**: Given a graph G, Check if there is a cycle in the graph that starts from a vertex, visits all other vertices exactly once and returns to the starting vertex.

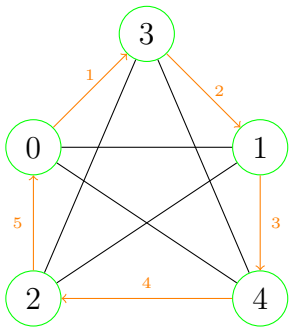


Figure 6: A Sequence of Edges for Hamiltonian Cycle

Reduction from Hamiltonian Cycle Problem to TSP

- Start with a graph G
- Construct G' by adding remaining edges to complete the graph and setting weights in the following way:
  - if (u, v) is an edge in G, set weight to 0
  - if (u, v) is an not an edge in G, set weight to 1

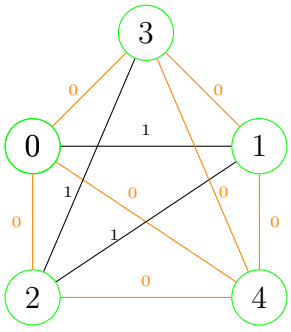


Figure 7: Graph G' with edges of G colored in orange

- G has a hamiltonian cycle if and only if G' has a TSP solution with cost ≤ 0

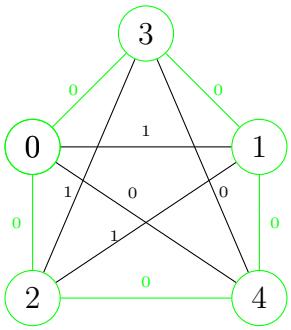


Figure 8: TSP with cost ≤ 0

So, we can say that Hamiltonian Cycle Problem$\leq_p$TSP
Other NP-hard Problems can be reduced to TSP as well

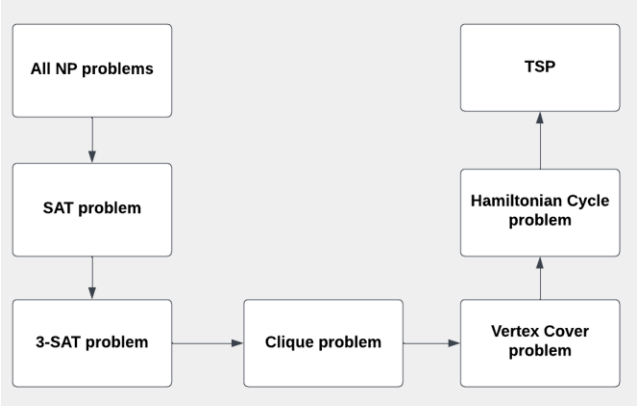


Figure 9: Reduction from Other Problems to TSP

- TSP is in NP
- TSP is NP-hard

Hence, TSP is NP-Complete.

1.4 Types of TSP

- **Symmetric TSP(sTSP)**: For this type of TSP problem, weights of edges between two vertices are symmetric[$c_{ij} = c_{ji}$ where c_{ij} denotes cost of travelling from i to j]
- **Asymmetric TSP(aTSP)**: For this type of TSP problem, weights of edges between two vertices are not always symmetric[$c_{ij} \neq c_{ji}$ where c_{ij} denotes cost of travelling from i to j]

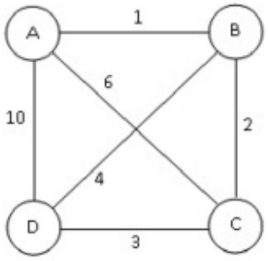


Figure 10: Symmetric TSP

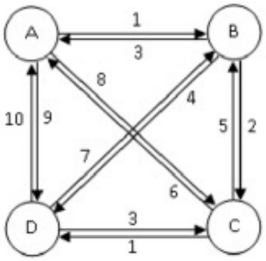


Figure 11: Asymmetric TSP

- **Multiple TSP(mTSP)**: In a given set of nodes, let there are m salesmen located at a single depot node. The remaining nodes (cities) that are to be visited are intermediate nodes. Then, the mTSP consists of finding tours for all m salesmen, who all start and end at the depot, such that each intermediate node is visited exactly once and the total cost of visiting all nodes is minimized.

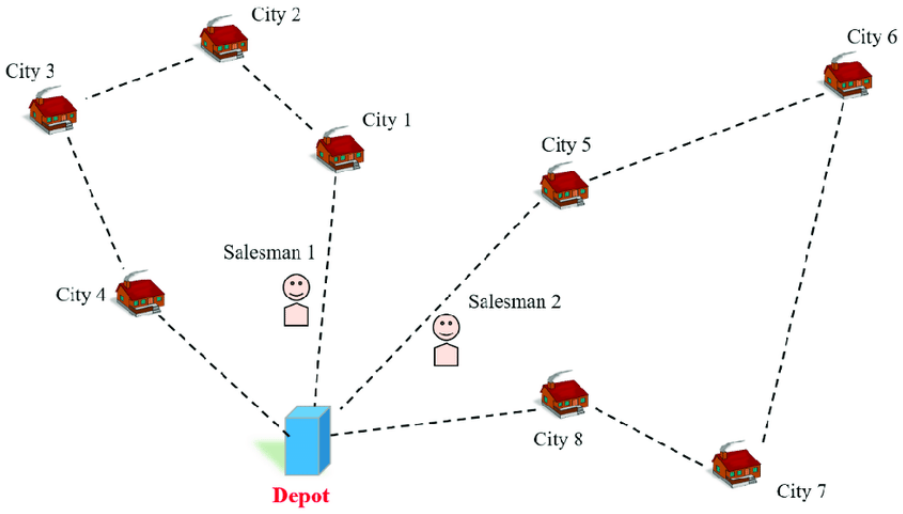


Figure 12: Multiple TSP

- **Theorem: There is no constant factor approximation for TSP, unless P=NP**
  - For contradiction, suppose there is a ρ approximation algorithm for TSP. Let G=(V,E) be an unweighted graph. $|V| = n$
  - Construct G' from G by adding edges and assigning weights to them as follows:
    $c_{ij} = \begin{cases} 1 & \text{if (i,j)} \in \text{E} \\ n\rho & \text{if (i,j)} \notin \text{E} \end{cases}$

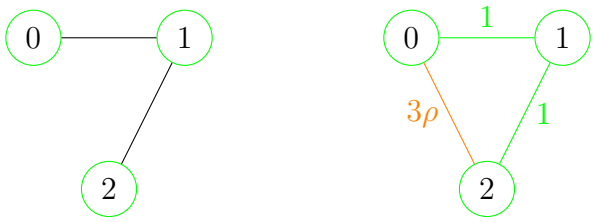
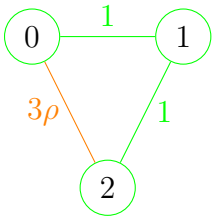


Figure 13: Construction of G' from G

  - Now suppose we run an approximate TSP solver on G'. If G has a hamiltonian cycle, then G' has a TSP of total cost n, otherwise, TSP will give a total cost greater than $n\rho$.
    So using approximation algorithm, we solved the hamiltonian cycle problem in polynomial time. But this is not possible unless P=NP.
  - **Hence, The theorem is proved.**



cost of TSP $> 3\rho$. No hamiltonian cycle

Figure 14: TSP For G'

  - Because there is no constant factor approximation for general TSP, we will use the following special case of TSP:

- **Metric TSP**: The input to TSP is called metric if for each triplets of vertices, the triangle inequality holds.
  That means, if $i, j, k \in V, c_{ik} \leq c_{ij} + c_{jk}$.
  - Approximate Algorithms exist for metric TSP.

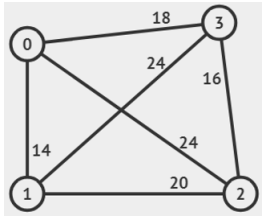


Figure 15: Metric TSP

- **Euclidean TSP**: This is a special type of metric TSP where each of the vertices are represented by Cartesian coordinates (x_v, y_v, z_v) and the distance calculation is done using euclidean distance formula: $c_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$. Euclidean TSP also maintains the following properties:
  - The optimum TSP does not intersect itself, as shown in the following figure:

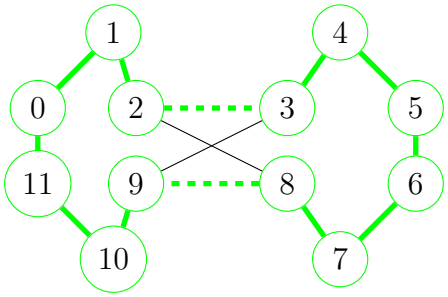


Figure 16: intersecting links $(2, 8), (9, 3)$ can be substituted by $(2, 3), (8, 9)$

  - Let m of the n points in the Euclidean TSP define the convex hull of the points. Then the order in which these m points appear in the optimum TSP tour must be the same as the order in which these same points appear on the convex hull

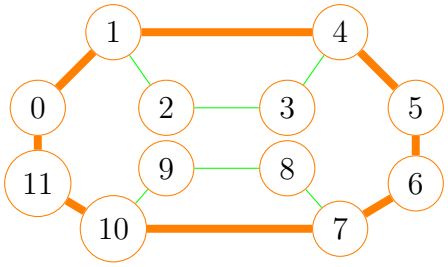


Figure 17: The figure shows the convex hull property of Euclidean TSP. $\{\ldots 4, 5, 6\}$ is the sequence in the convex hull. If there is a tour $\{\ldots 4, 6, 5\}$ found, then we can eliminate that tour.

2 Applications of TSP

Drilling of printed circuit boards

Problem Description

To connect a conductor on one layer with a conductor on another layer, or to position the pins of integrated circuits, holes have to be drilled through the board.
The holes may be of different sizes. To drill two holes of different diameters consecutively, the head of the machine has to move to a tool box and change the drilling equipment. This is quite time consuming.

TSP Formulation

It is clear that one has to choose some diameter, drill all holes of the same diameter, change the drill, drill the holes of the next diameter, etc.

Thus, this drilling problem can be viewed as a **series of TSPs, one for each hole diameter.**

- $'cities'$ are the initial position and the set of all holes that can be drilled with one and the same drill.
- The $'distance'$ between two cities is given by the time it takes to move the drilling head from one position to the other.

Vehicle routing

Problem Description

Suppose that in a city n mail boxes have to be emptied every day within a certain period of time, say 1 hour. The problem is to find the minimum number of trucks to do this and the shortest time to do the collections using this number of trucks.
As another example, suppose that n customers require certain amounts of some commodities and a supplier has to satisfy all demands with a fleet of trucks.

The problem is to find an assignment of customers to the trucks and a delivery schedule for each truck so that the capacity of each truck is not exceeded and the total travel distance is minimized.

TSP Formulation

- Constraints:
  - No time and capacity constraints

  - Number of trucks is fixed (say m)

- n mail boxes can be viewed as n cities

Related problems

- School bus routing problem
- Mission planning problem
- Search and Rescue problem
- Monitoring and surveillance.
- etc.

Computer wiring

Problem Description

For a Computer board, an interface consists of a number of modules, and on each module several pins are located where the position of each module has been determined in advance, a given subset of pins has to be connected. In order to avoid signal cross-talk and to improve ease and neatness of wirability, the total wire length has to be minimized.

TSP Formulation

- Constraints:
  - Number of wires is fixed (say m)
- n pins can be viewed as n cities

Delivery System

Problem Description

With m vehicles need to deliver goods to n spots and return to specific spot.

TSP Formulation

- m vehicles can be considered as m salesman
- n spots can be considered as n cities
- edge between nodes are the roads connecting the cities

3 Extisting Algorithms

3.1 Exact Algorithms

Algorithm	Description
Brute-force search	Approach: Try all permutations Complexity: $O(n!)$ Suitable only for a small network with few cities
Held–Karp algorithm	Approach: Dynamic programming Complexity: $O(n^2 2^n)$ Suitable for an intermediate number of cities
Concorde TSP solver	Approach: Branch-and-cut-and-price Can solve for a network of 85,900 cities but taking over 136 CPU-years

Table 1: Exact algorithms for solving the TSP

3.2 Approximation Algorithms

Algorithm	Description
Nearest Neighbour	Approach: Greedy search Complexity: $O(n^2)$ Approximation ratio: 2
Double Tree	Approach: Constructs Eulerian circuit by doubling edges of min spanning tree Complexity: $O(n^2)$ Approximation ratio: 2
Christofides Algorithm	Approach: Combines the min spanning tree with minimum-weight matching Complexity: $O(n^3)$ Approximation ratio: $\frac{3}{2}$

Table 2: Approximation algorithms for solving the TSP

3.3 Heuristics Algorithms

Algorithm	Description
Greedy heuristic	Approach: Selecting shortest edge Complexity: $O(n^2 \log_2(n))$
Match Twice and Stitch	Approach: Combines two sequential matching(min-cost edge cycle) Complexity: $O(n^2)$

Table 3: Heuristics algorithms for solving the TSP

3.4 Metaheuristics Algorithms

Algorithm	Description
Lin-Kernighan Algorithm	Approach: Local Search Complexity: $O(n^{2.2})$
Tabu Search	Approach: Local Search with tabu-list Complexity: $O(n^3)$
Simulated annealing	Approach: Explores the solution space by combining local search with random jumps, governed by a cooling schedule Complexity: $O(n^2)$
Genetic algorithm	Approach: population of solutions evolves over generations through operations like selection, crossover, and mutation Complexity: $O(n^3)$
Memetic algorithm	Approach: Combines the global search capability of genetic algorithms with local search heuristics Complexity: $O(n^3)$

Table 4: Metaheuristics algorithms for solving the TSP

4 TSPLib95

TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types. Instances of the following problem classes are available:

- Symmetric Travelling Salesman Problem (TSP)
- Hamiltonian Cycle (HCP)
- Asymmetric Travelling Salesman Problem (ATSP)
- Sequential Ordering Problem (SOP)
- Capacitated Vehicle Routing Problem (CVRP)

We only used Symmetric TSP instances for our experiments. Which means distance from node i to node j is equal to distance from node j to node i. All data are instances of real life, maintaining our requirement for Metric-TSP. We used 75 samples from the TSPLIB dataset with dimension ranging from 14 to 1000. add how many dataset. node and edge count

4.1 File Format

The instances are recorded in a .tsp file. All the entries in an instance are of the form $< keyword >:< value >$ form.

- **NAME :** burma14
- **TYPE :** TSP
- **COMMENT :** 14-State in Burma (Zaw Win)
- **DIMENSION :** 14
- **EDGE_WEIGHT_TYPE :** GEO
- **EDGE_WEIGHT_FORMAT :** FUNCTION
- **DISPLAY_DATA_TYPE :** COORD_DISPLAY
- **NODE_COORD_SECTION**
  1: 16.47 96.10
  2: 16.47 94.44
  ...
  14: 20.09 94.55
  EOF

5 Christofides Algorithm

Christofides Algorithm or Christofides–Serdyukov algorithm is capable of generating a 3/2-approximation algorithm for Metric TSP problems.

5.1 Pseudocode:

Algorithm 1 Christofides algorithm

Require: A complete weighted $G = (V, E, w)$ graph with Metric Property
Ensure: A Hamiltonian Cycle H

1: $T \leftarrow$ MST(G)
2: $O \leftarrow$ ODD-VERTICES(T).
3: $M \leftarrow$ MIN-WEIGHT-PERFECT-MATCHING(O)
4: $U \leftarrow$ UNION(T, M)
5: $C \leftarrow$ EULER-CIRCUIT(U)
6: $H \leftarrow$ SHORTCUTTING(C) // remove visited vertices
7: **return** H

5.2 Example

We first take $G(V, E)$, a complete weighted graph with Metric Property. 18

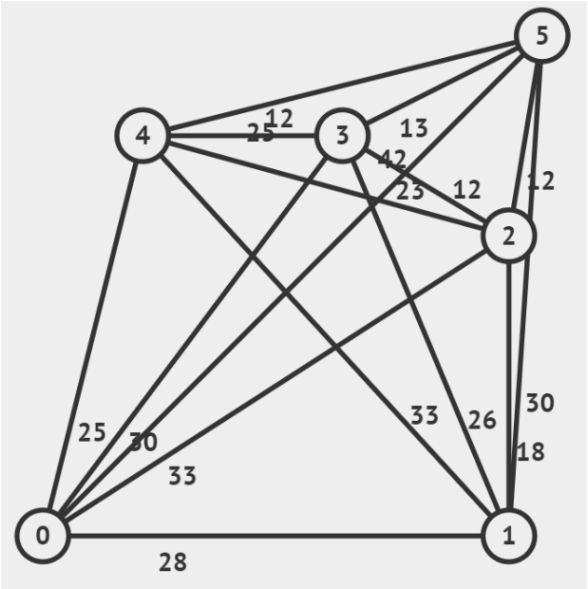


Figure 18: Original Graph

We then build a Minimum Spanning Tree 19. If we use Kruskal's Algorithm, it will take $O(E \log V)$ time complexity.

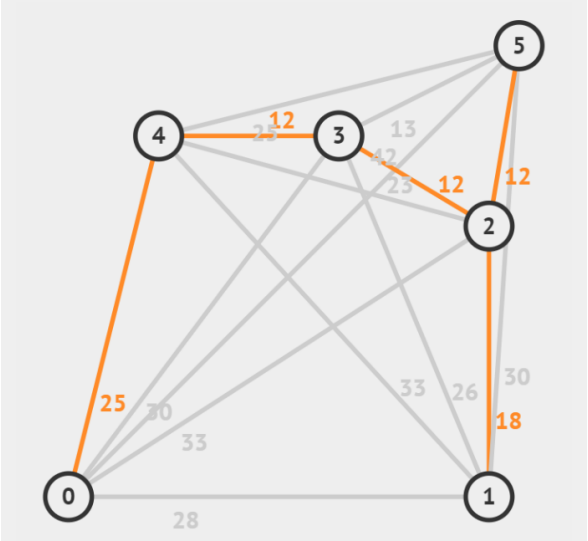


Figure 19: Minimum Spanning Tree

We are looking for an eulerian path. For that, we need all even degree vertices. Let us choose odd degree vertices, which are 0, 1, 2, 5. 20

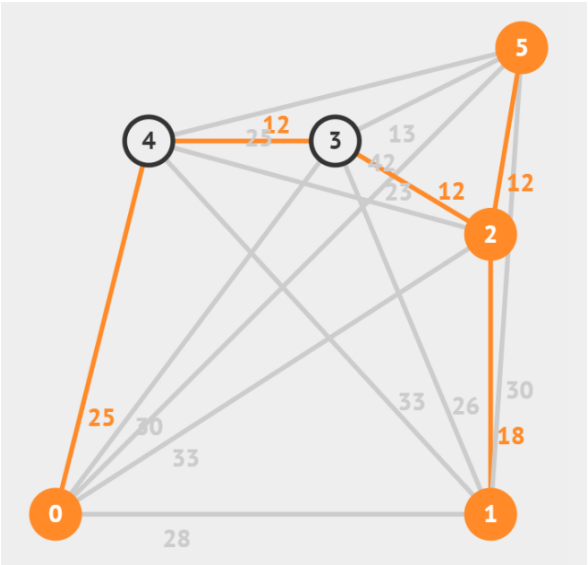


Figure 20: Odd Vertices in MST

Number of odd degree vertices must be even. Find a minimum cost perfect matching among the odd vertices 21 It has $O(n^3)$ time complexity.

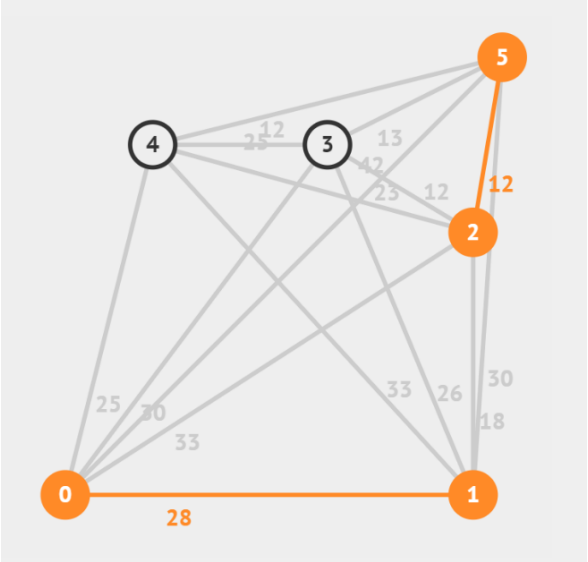


Figure 21: Minimum Cost Matching

Then we combine edges from the MST and Minimum Matching. It will create a multigraph 22.

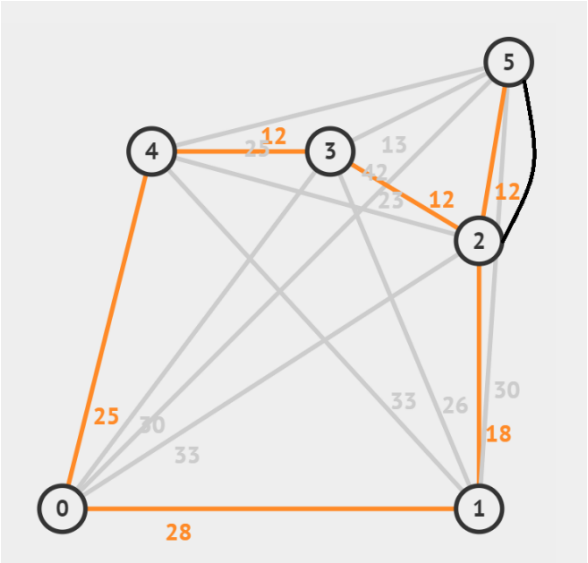


Figure 22: Multigraph

We then find the eulerian tour : 0, 1, 2, 5, 2, 3, 4, 0. But there are duplicates (in this instance 2). We directly go from 5 to 3, skipping the second 2. This is called shortcutting. Finally we arrive at our solution with $Cost = 108$.

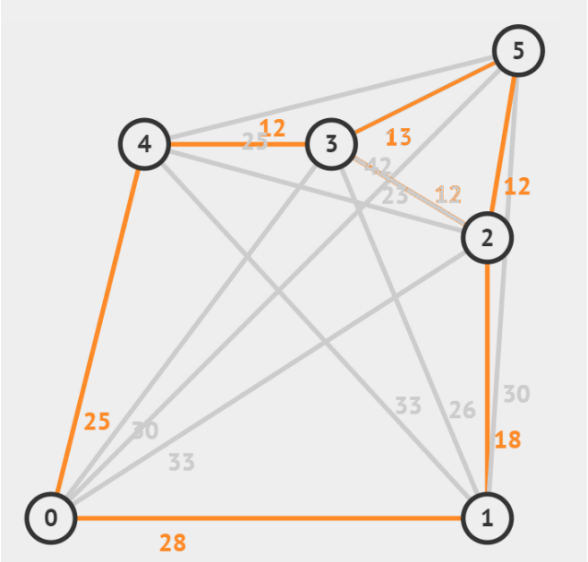


Figure 23: TSP Solution

This algorithm has overall $O(n^3)$ time complexity.

5.3 Result

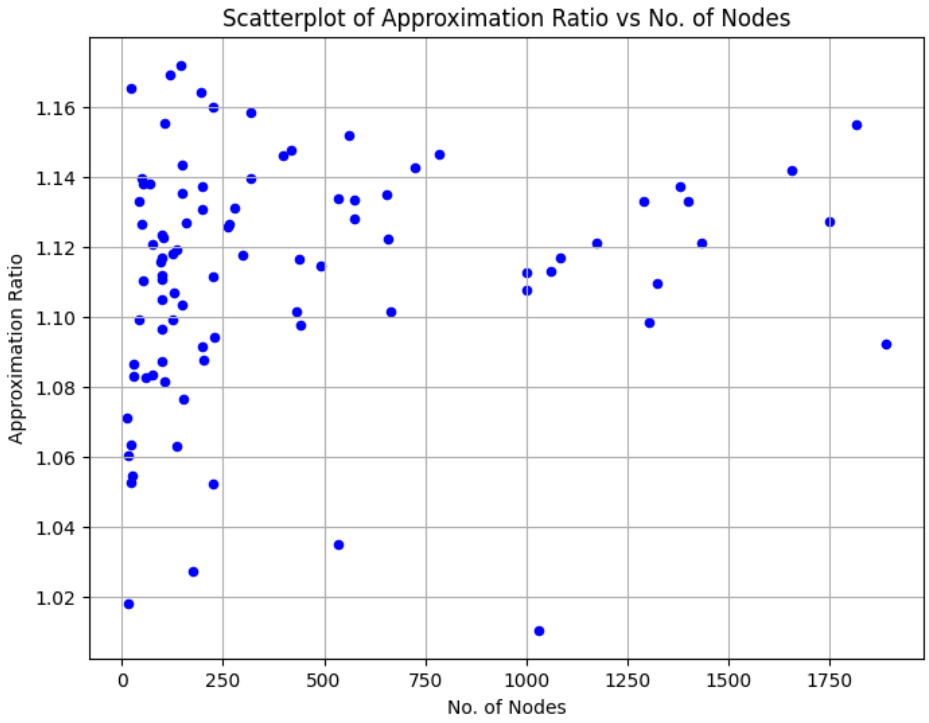


Figure 24: Scatterplot of APX ratio for Christofides algorithm

In Fig. [24], we see that Christofides gives approximation ratio in between 1.01 to 1.17. It performs quite consistently across small or large number of nodes.

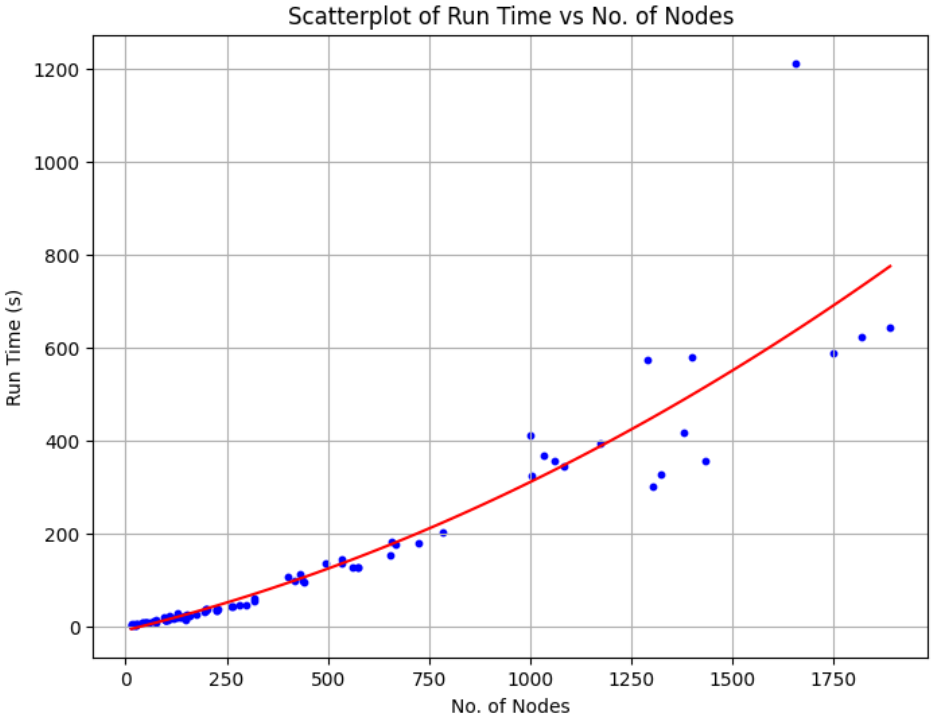


Figure 25: Christofides algorithm runtime plot

In Fig. [25], we notice runtime of our algorithm is increasing rapidly in consistent with $O(n^3)$ time complexity.

6 Simulated Annealing

6.1 Introduction

Simulated annealing, inspired by metallurgical annealing, is a metaheuristic algorithm used for finding near-optimal solutions to combinatorial optimization problems. It iteratively explores the solution space, gradually reducing search intensity to avoid local optima.

This approach is particularly effective for solving the Traveling Salesman Problem (TSP). Simulated annealing perturbs the current solution iteratively, occasionally accepting moves that lead to worse solutions with a certain probability of escaping local optima and converging towards a globally optimal or near-optimal solution.

Simulated annealing's ability to efficiently explore the solution space and avoid local optima makes it well-suited for addressing the complexities of the TSP. Through its iterative search process and probabilistic acceptance of inferior solutions, simulated annealing offers a robust approach to finding high-quality solutions to the TSP, making it a valuable tool in combinatorial optimization.

6.2 Flow Chart

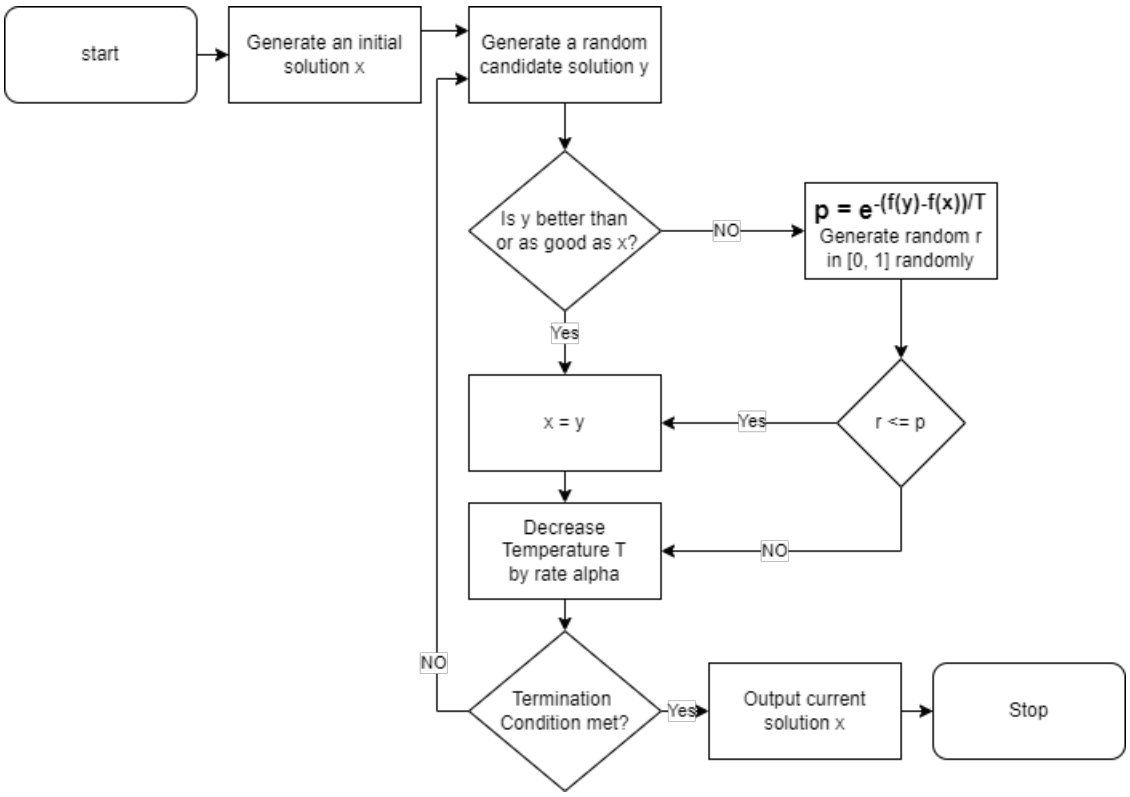


Figure 26: Flow Chart of Simulated Annealing

6.3 Pseudo Code

Algorithm 1 Simulated Annealing

```
Initialize parameters;
S ← generate initial solution();
T ← T_0;
N ← 0;
while T > 0 and N ≤ Iter do
    Generate solution S' in the neighborhood of S;
    if f(S') < f(S) then
        S ← S';
    else
        Δ ← f(S') − f(S);
        r ← random();
        if r < exp(−Δ/(k · T)) then
            S ← S';
        end if
    end if
    N ← N + 1;
    T ← a · T;
end while
Return the best solution found;
```

Figure 27: Pseudo code of Simulated Annealing

6.4 Initial Solution

The initial solution for the Traveling Salesman Problem (TSP) is formed by randomly shuffling the cities. For example, an initial solution could be represented as [1, 2, 3, ..., N, 1], where N represents the total number of cities.
We can alternatively utilize a greedy approach as an initial solution. Below is the pseudo-code outlining the greedy algorithm:

Algorithm 1 Simulated Annealing Pseudo-code

```
nodes ← list of all nodes in the graph
curr_node ← randomly choose a node from nodes
initialize current_solution as a list containing curr_node
available_nodes ← set of all nodes in the graph
available_nodes ← available_nodes \ {curr_node}
while available_nodes ≠ ∅ do
    next_node ← node in available_nodes with minimum edge weight from curr_node
    available_nodes ← available_nodes \ {next_node}
    current_solution ← current_solution + {next_node}
    curr_node ← next_node
end while
```

Figure 28: Initial Greedy Solution

6.5 Definition of Neighborhood

To determine the next neighboring solution, we randomly choose two indices from the current solution and then reverse the segment defined by these indices. An example is given below:

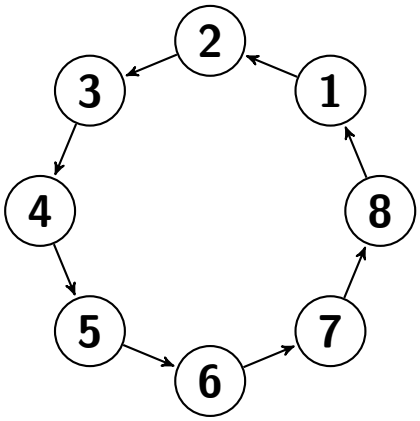


Figure 29: Current Tour

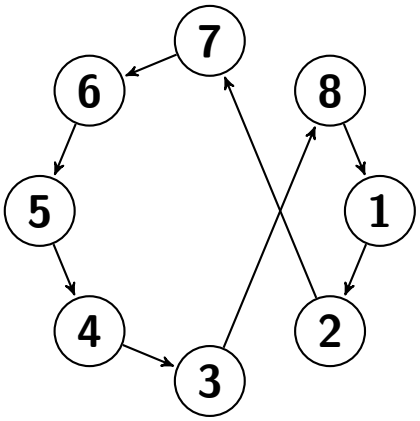


Figure 30: New Candidate Solution

Other strategies to define neighborhood can be:

- **2-opt Exchange:** This strategy involves removing two edges from the current solution and reconnecting them differently to form a new solution. It's a simple yet effective method for generating neighboring solutions.
- **Insertion:** In this strategy, a node is removed from the current solution and inserted at a different position, creating a new neighboring solution.
- **Swap:** This strategy involves swapping the positions of two nodes in the current solution to create a neighboring solution.
- **3-opt Exchange:** Similar to 2-opt exchange, but instead of removing and reconnecting two edges, it involves removing and reconnecting three edges to form a new solution.
- **Cross-Exchange:** This strategy involves exchanging segments between two tours or solutions to generate a new neighboring solution.

6.6 Fitness Function

In optimization algorithms such as simulated annealing, the fitness function evaluates the quality of a candidate solution. For the TSP, the fitness function measures the total length of the tour represented by the solution.

For instance, let $f(S)$ denote the fitness function, where S is a candidate solution or tour in the TSP. The fitness function can be expressed as:

$$f(S) = \sum_{i=1}^{n-1} d(S_i, S_{i+1}) + d(S_n, S_1)$$

Here, $S = (S_1, S_2, \ldots, S_i, \ldots, S_n)$ represents the sequence of visited cities in the tour. $d(S_i, S_{i+1})$ calculates the distance between consecutive cities S_i and S_{i+1}, while $d(S_n, S_1)$ computes the distance from the last city S_n back to the starting city S_1, completing the tour.

The fitness function $f(S)$ quantifies how well a candidate solution minimizes the total tour length. In simulated annealing, this function guides the search towards solutions with lower fitness values, aiming to converge towards an optimal or near-optimal solution with a shorter tour length.

6.7 Temperature Control Strategies

Temperature control is a crucial aspect of simulated annealing, an optimization algorithm inspired by metallurgical annealing. The algorithm employs a temperature parameter, denoted by T, which influences the acceptance probability of candidate solutions during the search process. The temperature is updated iteratively using the formula:

$$T = T - \alpha \cdot T$$

Here, α represents the cooling parameter, determining the rate at which the temperature decreases over time. By adjusting α, practitioners can fine-tune the balance between exploration and exploitation. A higher α value encourages more exploration, allowing the algorithm to escape local optima, while a lower α value promotes exploitation, focusing on refining promising regions of the solution space. Temperature control through α plays a pivotal role in guiding simulated annealing towards optimal or near-optimal solutions in combinatorial optimization problems.

6.8 Termination Condition

In simulated annealing for the Traveling Salesman Problem (TSP), termination conditions dictate when the algorithm ceases searching for better solutions. One common condition is to halt after reaching a maximum number of iterations, ensuring the algorithm explores the solution space adequately. Additionally, termination occurs when the temperature drops below or equals zero, signifying significant cooling and convergence toward an optimal or near-optimal solution.

Other termination conditions for TSP simulated annealing may include:

- **Stagnation**: Terminate the algorithm if no improvement in solution quality is observed after a specific number of iterations or consecutive iterations.
- **Threshold Improvement**: Halt the algorithm when the improvement in the objective function falls below a predefined threshold, indicating diminishing returns from further exploration.
- **Runtime Limit**: Set a maximum allowable runtime for the algorithm to ensure termination within a specified timeframe, thereby optimizing computational resource usage.

These conditions ensure computational efficiency and guide the algorithm towards finding high-quality solutions within predefined constraints.

6.9 Results

6.9.1 Tour Cost vs Iteration

A tour cost versus iteration graph illustrates the progression of the cost of the tour as the number of iterations increases in the simulated annealing algorithm.

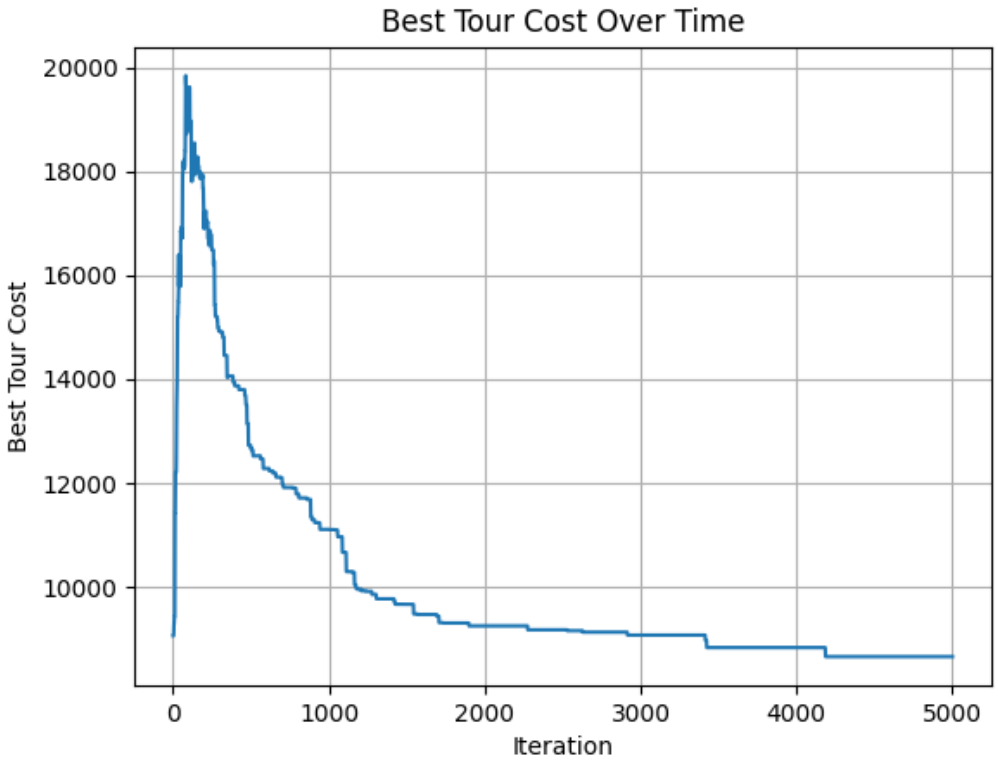


Figure 31: Tour Cost vs Iteration

The tour cost is decreasing over iterations, reflecting the algorithm's optimization process and convergence toward an optimal or near-optimal solution.

6.9.2 Tour Cost vs Temperature

In a tour cost versus initial temperature graph, the initial temperature on the x-axis represents varying starting temperatures for simulated annealing. The y-axis depicts the resulting tour cost after the algorithm's execution. This graph offers insights into how different initial temperatures influence the optimization process, with lower tour costs indicating better solutions.

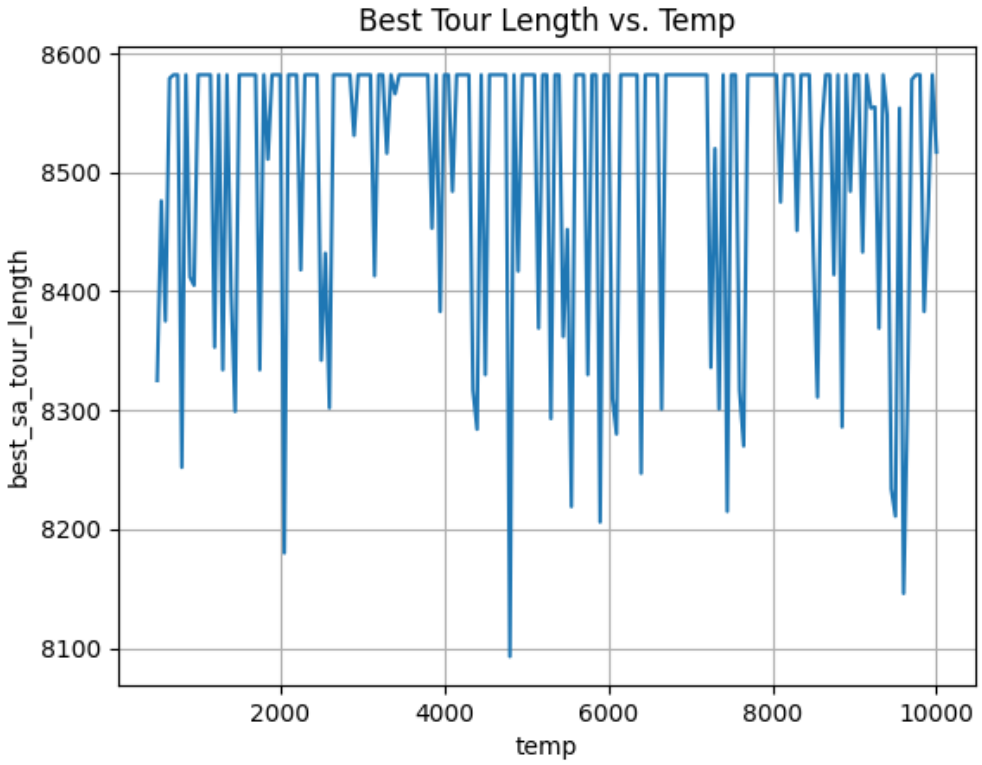


Figure 32: Tour Cost vs Temp

In the tour cost versus initial temperature graph, no significant trend is observed, but an initial temperature of approximately 5000 serves as a reasonable estimate. This suggests that an initial temperature around this value may yield satisfactory results in the simulated annealing algorithm for the given problem.

6.9.3 Improvement in Ratio

Dataset Name	Dimension (nodes)	C_Ratio	SA_Ratio	Improvement
fri26	26	1.054429	1.03095	0.023479
bays29	29	1.083168	1.006436	0.076733
swiss42	42	1.098979	1.087981	0.010998
hk48	48	1.139429	1.013088	0.126342
brazil58	58	1.082693	1.025517	0.057177
st70	70	1.137778	1.084444	0.053333
kroB150	150	1.135438	1.104669	0.030769
pr152	152	1.076477	1.046389	0.030089
rat195	195	1.164012	1.158846	0.005166
d657	657	1.122240	1.122240	0.000000

We observed a significant improvement in the solution ratio when the solution generated by the Christofides Algorithm was used as the initial solution in simulated annealing.

6.9.4 Empirical Ratios for Simulated Annealing

Name	Min_Ratio	Max_Ratio	Avg_Ratio
Christofides	1.0544290	1.1640120	1.1094644
C+SA	1.0064356	1.1588463	1.0680559

From the table, it's evident that in all cases, we achieved better results than the Christofides algorithm by using simulated annealing with the initial solution provided by the Christofides algorithm.

7 Introducing Randomization to TSP

7.1 Revising Christofides

Let's again look at the pseudocode for Christofides Algorithm.

Algorithm 2 Christofides Algorithm

Require: A complete weighted $G = (V, E, w)$ graph with Metric Property
Ensure: A Hamiltonian Cycle H
1: $T \leftarrow$ MST(G)
2: $O \leftarrow$ ODD-VERTICES(T).
3: $M \leftarrow$ MIN-WEIGHT-PERFECT-MATCHING(O)
4: $U \leftarrow$ UNION(T, M)
5: $C \leftarrow$ EULER-CIRCUIT(U)
6: $H \leftarrow$ SHORTCUTTING(C)
7: **return** H

Here, it is apparent that Christofides Algorithm is deterministic that is, it provides the same output every time it is run on a particular input instance. Moreover, Christofides Algorithms use MST as a subroutine for finding the minimum spanning tree. MST sub-routine can be implemented easily using Prim's Algorithm or Kruskal's Algorithm. We will look at Kruskal's Algorithm in the next section.

7.2 Kruskal's Algorithm

The pseudocode for Kruskal's Algorithm is provided below.

Algorithm 3 KRUSKAL-MST

Require: A connected, undirected, weighted graph $G = (V, E)$
Ensure: A minimum spanning tree T of G

```
1: T ← ∅
2: for v ∈ G.V do
3:     MAKE-SET(v)
4: end for
5: E ← LIST(G.E)
6: Sort E in non-decreasing order by weight
7: i ← 0
8: while |T| < |G.V| − 1 do
9:     e ← E[i]
10:    if FIND-SET(e.u) ≠ FIND-SET(e.v) then
11:        T ← T ∪ {e}
12:        UNION(e.u, e.v)
13:    end if
14:    i ← i + 1
15: end while
16: return T
```

The main idea of Kruskal's Algorithm is that we first sort the edges in non-decreasing order by their weight. We build our MST by adding one edge at a time by traversing the sorted edge list. If the current selected edge does not create a cycle, then we add that edge to our MST.

Time Complexity Analysis:

- $|V|$ MAKE-SET operations take $O(V \log V)$
- Sorting E takes $O(E \log E) = O(E \log V)$
- $|E|$ FIND-SET and UNION operations take $O(E \log E) = O(E \log V)$

For a complete graph, $|E| = O(V^2)$. Thus, the total time complexity of Kruskal's Algorithm is $O(E \log V) = O(V^2 \log V)$

7.3 Trying Other Spanning Trees

Christofides Algorithm uses a minimum spanning tree (MST) for finding a TSP tour. However, MST does not give the optimal tour. So, we can think of using spanning trees other than the minimum spanning tree. The obvious idea that comes to mind for trying other spanning trees is to sample random spanning trees since it is straightforward to sample random spanning trees. We do this by randomly picking up edges and trying to see if we can add edges to our incomplete spanning tree.

Now the question arises: do random spanning trees provide a better approximation to the TSP problem than MST? To find the answer to this question, we tried randomly sampling spanning trees on some TSPLIB datasets and finding the approximation ratios. The results are given below:

Name	Dimension	C_Ratio	R_Ratio
fri26	26	1.054429029	3.186766275
bays29	29	1.083168317	3.200495050
swiss42	42	1.098978790	3.825608798
hk48	48	1.139429369	4.826978449
brazil58	58	1.082693444	4.869895649

Here, C_Ratio denotes the approximation ratio using MST, and R_Ratio denotes the approximation ratio using a random spanning tree. It is apparent that random spanning trees perform significantly worse than MST and the approximation ratio explodes. Thus, randomly sampling spanning trees is not a viable option to improve the empirical approximability of Christofides Algorithm. So, is randomization of no use in Christofides Algorithm? As we will see in the next section, we can use controlled randomization to improve upon the classic Christofides Algorithm.

7.4 Randomized Spanning Tree

We propose a novel randomized algorithm for sampling a random spanning tree based on Kruskal's Algorithm. We introduce a randomization parameter to control the randomization from Kruskal's Algorithm. The primary difference between KRUSKAL-MST and k-RANDOMIZED-KRUSKAL-ST is that while KRUSKAL-MST always chooses the minimum weight edge among the currently available edges, k-RANDOMIZED-KRUSKAL-S randomly samples an edge from the k least weight edges. This random sampling is done by SAMPLE-INDEX sub-routine.

Algorithm 4 k-RANDOMIZED-ST

Require: A connected, undirected, weighted graph $G = (V, E)$
Ensure: A spanning tree T of G and a randomization parameter k

$T \leftarrow \emptyset$
for $v \in G.V$ **do**
    MAKE-SET(v)
end for
Sort E in non-decreasing order by weight
while $|T| < |G.V| - 1$ **do**
    $j \leftarrow \text{MIN}(|E|, k)$
    $W \leftarrow [e.w \textbf{ for } e \in E[0, j+1]]$
    $i \leftarrow$ SAMPLE-INDEX(W)
    $e \leftarrow E[i]$
    if FIND-SET($e.u$) $\neq$ FIND-SET($e.v$) **then**
        $T \leftarrow T \cup \{e\}$
        UNION($e.u, e.v$)
    end if
    ERASE(E, i)
end while
return T

Algorithm 5 SAMPLE-INDEX

Require: A list of weights W
Ensure: Sample a random index $i \in [0, |W|]$

Z-Normalize the values in W
$P \leftarrow [e^{-w} \textbf{ for } w \in W]$
$s \leftarrow$ SUM(P)
Sample $j \in [0, |W| - 1]$ using the probability distribution P
return j

Our algorithm performs identically to Kruskal's Algorithm when $k = 1$ as we always choose the minimum weight edge. Thus, for, $k = 1$ k-Randomized-ST gives the MST. However, as we increase the value of k, the spanning tree returned by k-Randomized-ST deviates from the MST. In this way, we can control the amount of deviation from the MST using the randomization parameter k.

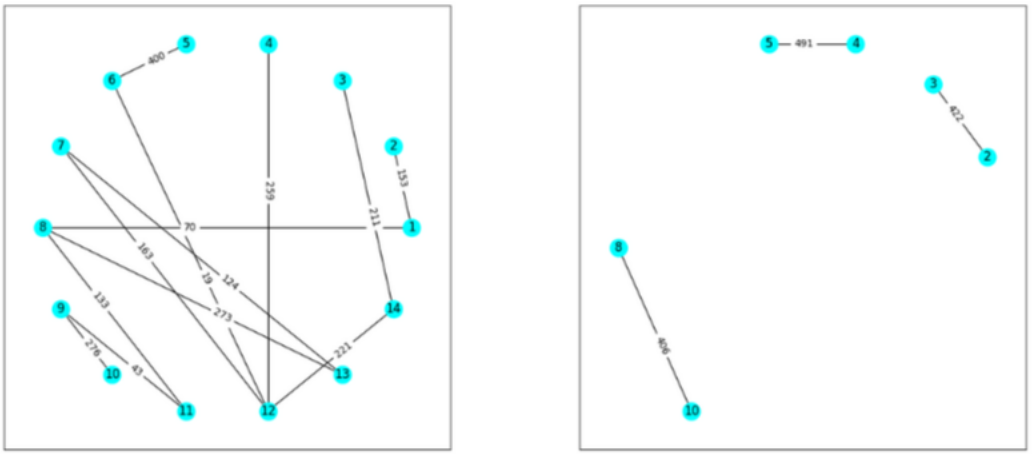
Time Complexity Analysis: k-RANDOMIZED-ST performs Additional $|E|$ calls to SAMPLE-INDEX take $O(kE) = O(kV^2)$ than KRUSKAL-MST. Moreover, $|E|$ ERASE operations take $O(kE) = O(kV^2)$. Thus, total time complexity of k-RANDOMIZED-ST is $O(V^2 \log V + kV^2)$. As a result, k-RANDOMIZED-ST does not

run much slower than KRUSKAL-MST for small values of k.

Now, we can use k-RANDOMIZED-ST to find a spanning tree in Christofides Algorithm to add randomization.

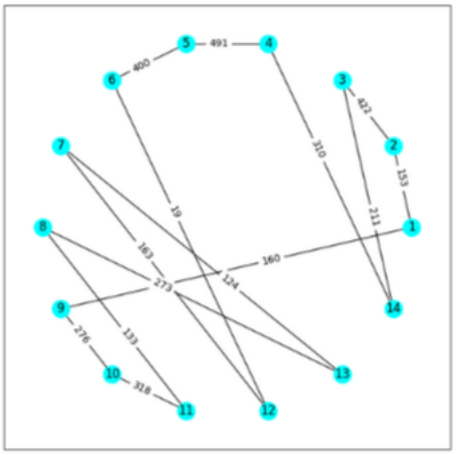
7.5 An Example Comparision

Here, we present a comparison between classic Christofides and randomized Christofides by running both the algorithms on *burma14* dataset from TSPLIB.

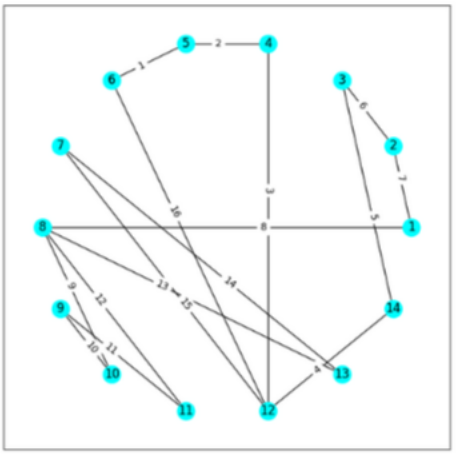


a. Minimum Spanning Tree (Cost = 2345)

b. Perfect Matching (Cost = 1319)



c. Minimum Spanning Tree
+
Perfect Matching
(Cost = 3559)



d. TSP Tour (Cost = 3559)

Figure 33: Simulation of Christofides Algorithm using Minimum Spanning Tree

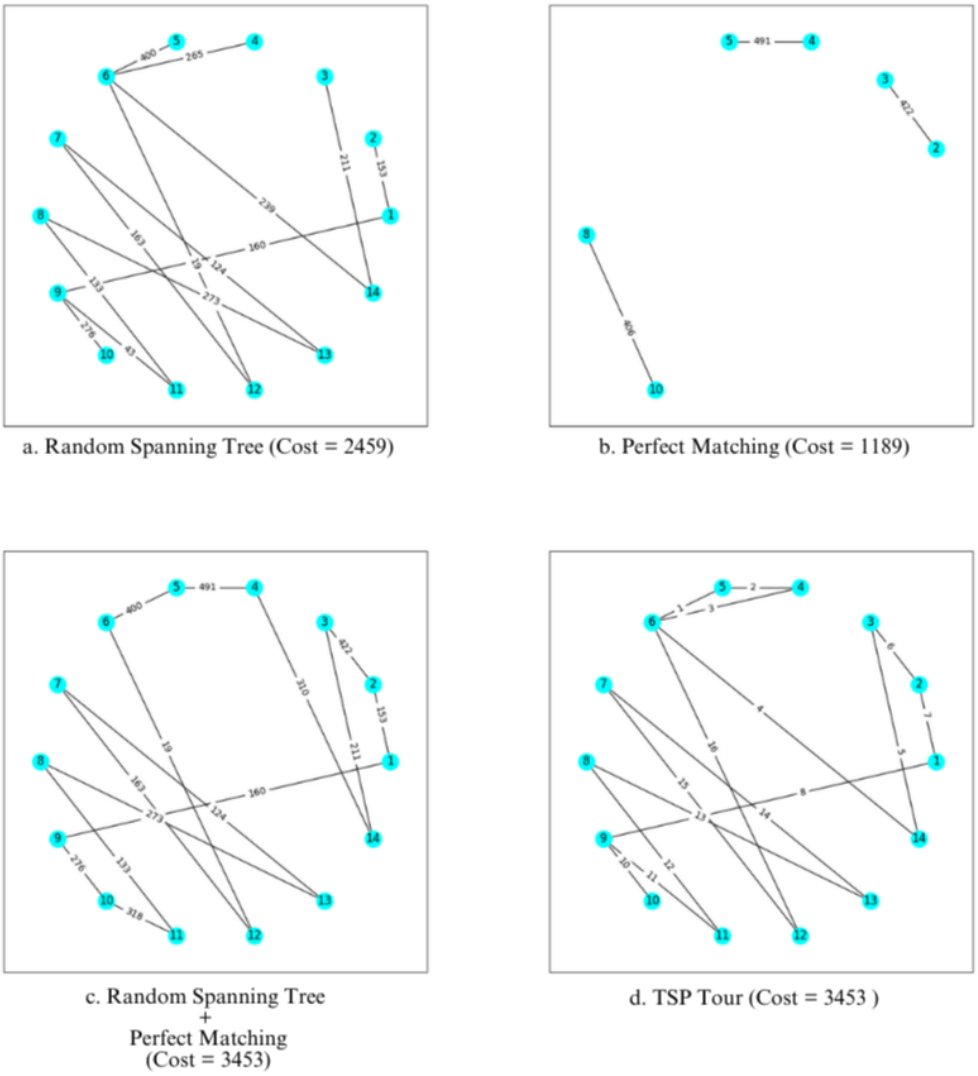


Figure 34: Simulation of Christofides Algorithm using Random Spanning Tree

From the above example, it is evident that a random spanning tree which is slightly different from the minimum spanning tree can result in a better TSP tour.

7.6 Experimental Results

Here, we present the results of various experiments performed using the TSPLIB dataset.

7.6.1 Scatter Plot: Ratio vs Dimension

Here we present the result of running classic Christofides, randomized Christofides, and simulated annealing on TSPLIB dataset. It is apparent from the scatterplot that randomized Christofides perform gives better ratio than classic Christofides at the expense of running randomized Christofides multiple times to find the best randomization parameter k.

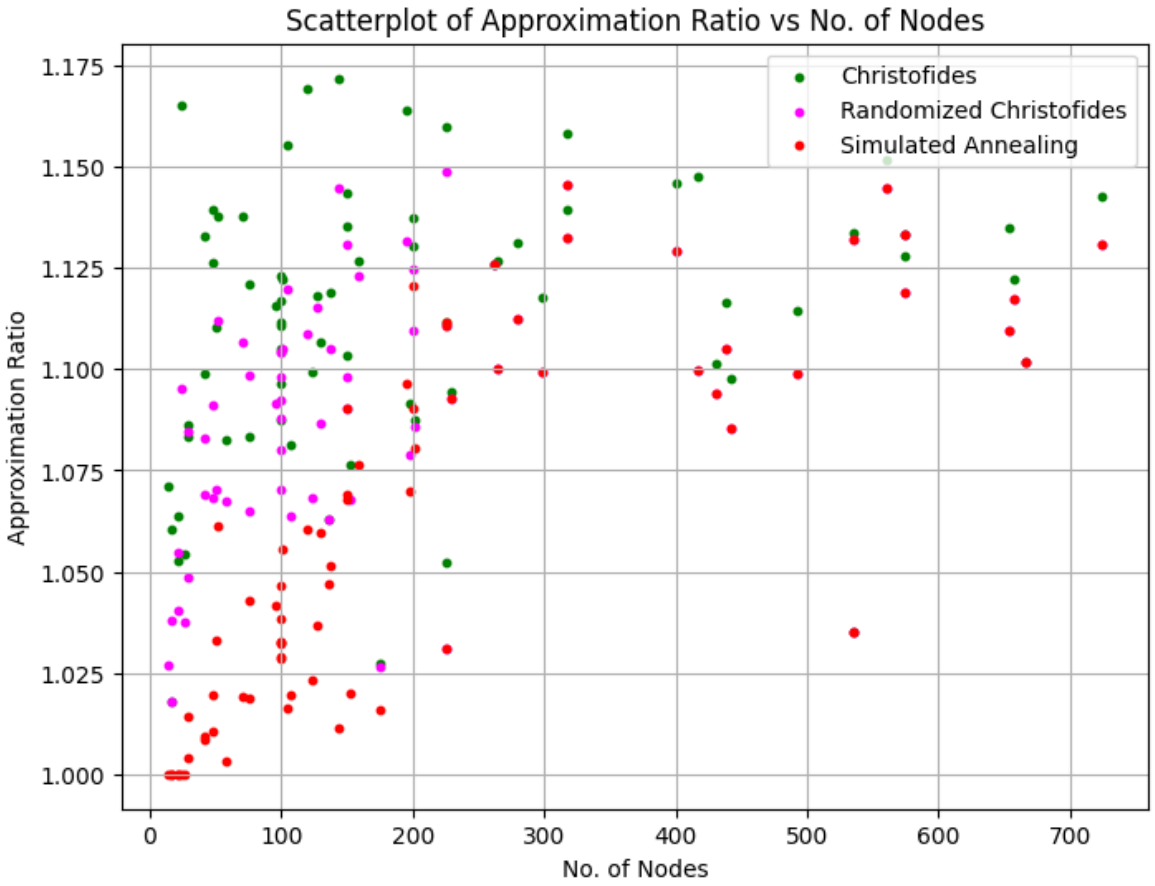


Figure 35: Ratio vs Dimension Scatterplot

7.6.2 Tabulation: Classic Christofides vs Randomized Christofides

Here we present a tabulation results for classic Christofides (C_Ratio) and randomized Christofides (RC_Ratio) along with the best randomization parameter on a subset of the TSPLIB dataset. For each sample dataset, randomized Christofides improve upon classic Christofides.

Name	Dimension	C_Ratio	RC_Ratio	Best_k	Improvement
fri26	26	1.054429	1.037353	10	0.017076
bays29	29	1.083168	1.048515	11	0.034653
swiss42	42	1.098979	1.069128	16	0.029851
hk48	48	1.139429	1.091179	13	0.048251
brazil58	58	1.082693	1.067533	2	0.015160
st70	70	1.137778	1.106667	9	0.031111
kroB150	150	1.135438	1.130884	6	0.004554
pr152	152	1.076477	1.067941	14	0.008537
rat195	195	1.164012	1.131726	13	0.032286
d657	657	1.122240	1.117170	9	0.005070

7.6.3 Tabulation: Simulated Annealing using Randomized Christofides

Here we present the tabulation results for running simulated annealing using randomized Christofides to find the initial solution on a subset of the TSPLIB dataset. For each sample dataset, simulated annealing significantly improves upon the solution found using classic Christofides.

Name	Dimension	C_Ratio	SA_Ratio	Best_k	Improvement
fri26	26	1.054429	1.000000	7	0.054429
bays29	29	1.083168	1.003960	15	0.079208
swiss42	42	1.098979	1.009427	19	0.089552
hk48	48	1.139429	1.010383	4	0.129046
brazil58	58	1.082693	1.003150	2	0.079543
st70	70	1.137778	1.019259	14	0.118519
kroB150	150	1.135438	1.067700	19	0.067738
pr152	152	1.076477	1.019761	14	0.056717
rat195	195	1.164012	1.096427	1	0.067585
d657	657	1.122240	1.117170	9	0.005070

7.6.4 Accumulated Empirical Result

Here we present the minimum, maximum, and average ratio of running different algorithms on the TSPLIB dataset (a total of 75 samples are used).

Name	Min_Ratio	Max_Ratio	Avg_Ratio
Classic Christofides	1.0179326	1.1718571	1.1115747
Rand. Christofides	1.0179326	1.1487638	1.0929558
Simulated Annealing	1.0000000	1.1456040	1.0602919

8 Conclusion

Our exploration of the Travelling Salesman Problem (TSP) revealed its enduring importance in the field of optimization. We provided a comprehensive overview, delving into its formal definition, complexity, and real-world applications. We then compared existing solutions, implemented two of them, and evaluated their performance on a diverse dataset. Finally, we proposed a novel randomization algorithm with promising results, suggesting a potential avenue for future research.