# Hybrid Approach for Travelling Salesman Problem in Clustered Spaces

27-04-2024

MTECH PROJECT REPORT
FOR THE FULFILLMENT OF REQUIREMENTS FOR THE DEGREE OF

**Dual Degree in Electronics and Electrical Communication Engineering with specialization in Vision and Intelligent Systems**

by

**Arsh Agarwal**
(Roll No: 19EC39005)

Under the supervision of
**Prof. Rajarshi Roy**
(Dept. of Electronics and Electrical Communications Engineering)


**Electronics and Electrical Communications Engineering**
**Indian Institute of Technology Kharagpur**
**April 2024**

Department of Electronics and
Electrical Communications Engineering
Indian Institute of Technology,
Kharagpur
India - 721302

# CERTIFICATE

This is to certify that the project report entitled **Hybrid Approach for Travelling Salesman Problem in Clustered Spaces**, submitted by **Arsh Agarwal** (Roll Number: *19EC39005*) an undergraduate student of **Department of Electronics and Electrical Communication** towards fulfilment of requirements for the award of degree of **Dual Degree in Electronics and Electrical Communication Engineering with specialization in Vision and Intelligent Systems**, is a record of bona-fide work carried out by him under my supervision and guidance during Spring Semester, 2023-24

**Professor Rajarshi Roy**

**Department of Electronics and Electrical Communications Engineering**
Indian Institute of Technology, Kharagpur

**Place: Kharagpur**
**Date: 27th April, 2024**

# Contents

**Abstract**

We introduce a novel approach to addressing the Traveling Salesman Problem (TSP) in clustered spaces. By leveraging the inherent structure of clustered spatial configurations, our method enhances route planning efficiency through cluster-aware optimizations. We integrate these optimizations with the Christofides algorithm, resulting in better solutions with reduced computational overhead. Extensive comparative studies demonstrate the efficacy of our approach, showcasing improvements in solution quality and computational efficiency compared to traditional TSP solvers. This research contributes to advancing optimization techniques for spatially clustered problems, offering practical solutions for route planning in diverse domains such as logistics, manufacturing, and networking.

# 1 Introduction

The Traveling Salesman Problem (TSP) is a classic combinatorial puzzle that has intrigued mathematicians, computer scientists, and operations researchers for decades. Let us consider a group of cities and the edges between them, the goal is to find the least distance route that visits each city exactly once and returns to the starting city. The NP-hard nature of the TSP renders exact solutions computationally infeasible for large instances, motivating the development of innovations.

This paper presents a novel TSP approximation algorithm for clustered spaces. We begin by discussing the fundamental concepts and complexities associated with the TSP and Christofides algorithm, providing a theoretical foundation for understanding the approximations.

Furthermore, we explore bio-inspired algorithms, including ant colony optimization, particle swarm optimization, and genetic algorithms, which leverage iterative approaches and heuristics to approach the optimal solution. We then use these for comparative study with our algorithm.

# 2 Motivation

The Traveling Salesman Problem (TSP) in clustered spaces is a paradigm that fits with various real-world applications. These include logistics and delivery services, urban planning and public transportation, wireless sensor networks, manufacturing and supply chain management, telecommunications and network routing, etc. By developing a tailored solution, this research aims to fill a critical gap in existing

methodologies. Most available approaches use either completely deterministic algorithms or fully heuristic-based techniques. Leveraging the best of both worlds, there is room for improvement in this domain for faster and artificially intelligent solutions.

# 3 The Travelling Salesman Problem (TSP)

Let us consider a group of cities and the edges between them, the goal is to find the least distance route that visits each city exactly once and returns to the starting city.

Formally, let $n$ be the number of cities, and let $d_{ij}$ represent the distance between city $i$ and city $j$. The goal is to find a permutation of the cities $(1, 2, \ldots, n)$ that minimizes the total distance traveled.

## 3.1 Challenges

- **Combinatorial Explosion:** The number of possible solutions to the TSP grows factorially with the number of cities, making it computationally intractable to solve for large instances using exact algorithms.
- **NP-Hardness:** There is no known polynomial-time algorithm to solve it optimally unless $P = NP$.

# 4 Christofides Approximation

## 4.1 Primary Assumptions

**Metric TSP:** Metric TSP is a subcase of the generic Travelling Salesman Problem where the Triangle Inequality holds, i.e., for nodes $i$, $j$ and $k$, the following holds:

$$\forall(i, k, j) \quad d(i, j) \leq d(i, k) + d(k, j) \tag{1}$$

where $d_{ij}$ represents the distance between nodes $i$ and $j$. This is a very generic and valid assumption for most real-life problems and practical solutions as it basically brings into the picture the physical structure of cities and towns which inherently have this property. Other assumptions include:

- **Non-negativity:** The distance between any two points must be non-negative $(d(u, v) \geq 0; \forall u, \forall v)$.

- **Identity:** The distance between a point and itself is zero ($d(u, u) = 0; \forall u$).
- **Symmetry:** The distance from A to B is the same as traveling from B to A ($d(u, v) = d(v, u); \forall u, \forall v$).
- **Complete Graph:** The algorithm assumes the input is a complete graph, meaning there exists an edge between each pair of nodes.

**Revisiting nodes is allowed:** We modify the original problem of visiting each node exactly once to visiting a node at least once. If the graph is complete (i.e. there is an edge between each pair of edges), then the metric TSP assumption renders this assumption redundant. On the contrary, if the graph is incomplete, then there may exist shorter paths between a pair of nodes that require revisiting a previously visited node. We can use this to create an auxiliary graph using the Floyd Warshall algorithm which would be complete. Thus, this enables us to cover incomplete graphs as well. Moreover, this is in accordance with practicality and real-life situations, hence the assumption is justified.

## 4.2 Algorithm Description

Here's a breakdown of the key steps:

- **Minimum Spanning Tree (MST):** The algorithm starts by finding a minimum spanning tree (MST) for the complete graph representing the cities and their distances. The MST ensures the least total edge weight overall.
- **Odd Degree Vertices:** Identify the set of nodes with odd degrees (the number of edges connected to the node) in the MST. These represent starting and ending points for additional connections.
- **Minimum Weight Perfect Matching:** We first create a subgraph using only the above-identified odd-degree nodes. Then, we go on to find a minimum-weight perfect matching within this. It pairs all the odd-degree nodes and uses the edge between them, minimizing the total intra-pair distance. (Note that the number of odd-degree nodes will always be even as the sum of degrees of all nodes is even because every edge adds 2 to the sum.)
- **Combine MST and Matching:** Combine the edges from the MST with the edges from the matching. This creates a new graph with all even degree nodes only, suitable for making an Eulerian circuit.

- **Eulerian Circuit:** Find an Eulerian tour in the newly constructed graph. An Eulerian tour is a path that visits every edge exactly once, guaranteed to exist due to the even degree of all nodes.
- **Shortcut the Tour (Hamiltonian Circuit):** The Eulerian tour might visit some nodes multiple times. By carefully eliminating these repetitions (shortcuts), we obtain a Hamiltonian circuit – a cycle visiting every node once only.

## 4.3 Implementation Details

**Finding the MST** To find the MST, we use the Kruskal's algorithm. MST is a subgraph that includes all the vertices of the original graph with the minimum possible number of edges and the smallest possible total edge weight. A step-by-step description of Kruskal's algorithm:

- **Sort Edges by Weight:** Begin by sorting all the edges of the graph in a non-decreasing order of their weights.
- **Initialize Empty MST:** Create an empty set to hold the edges of the minimum spanning tree.
- **Iterate Over Sorted Edges:** Starting with the smallest edge, iterate through all the edges.
- **Check for Cycle:** For each edge, check if adding it connects 2 previously disconnected components. This can be efficiently done using a disjoint-set data structure.
- **Repeat:** Continue this process until we have $n - 1$ edges in the MST, where $n$ is the number of nodes in the graph.

**Finding the odd degree nodes** We can maintain a list or array of the count of edges for each of the nodes. Then, we iterate over each edge and add 1 in the corresponding degrees of both ends. Finally, we can separate out the nodes with odd degrees for further processing.

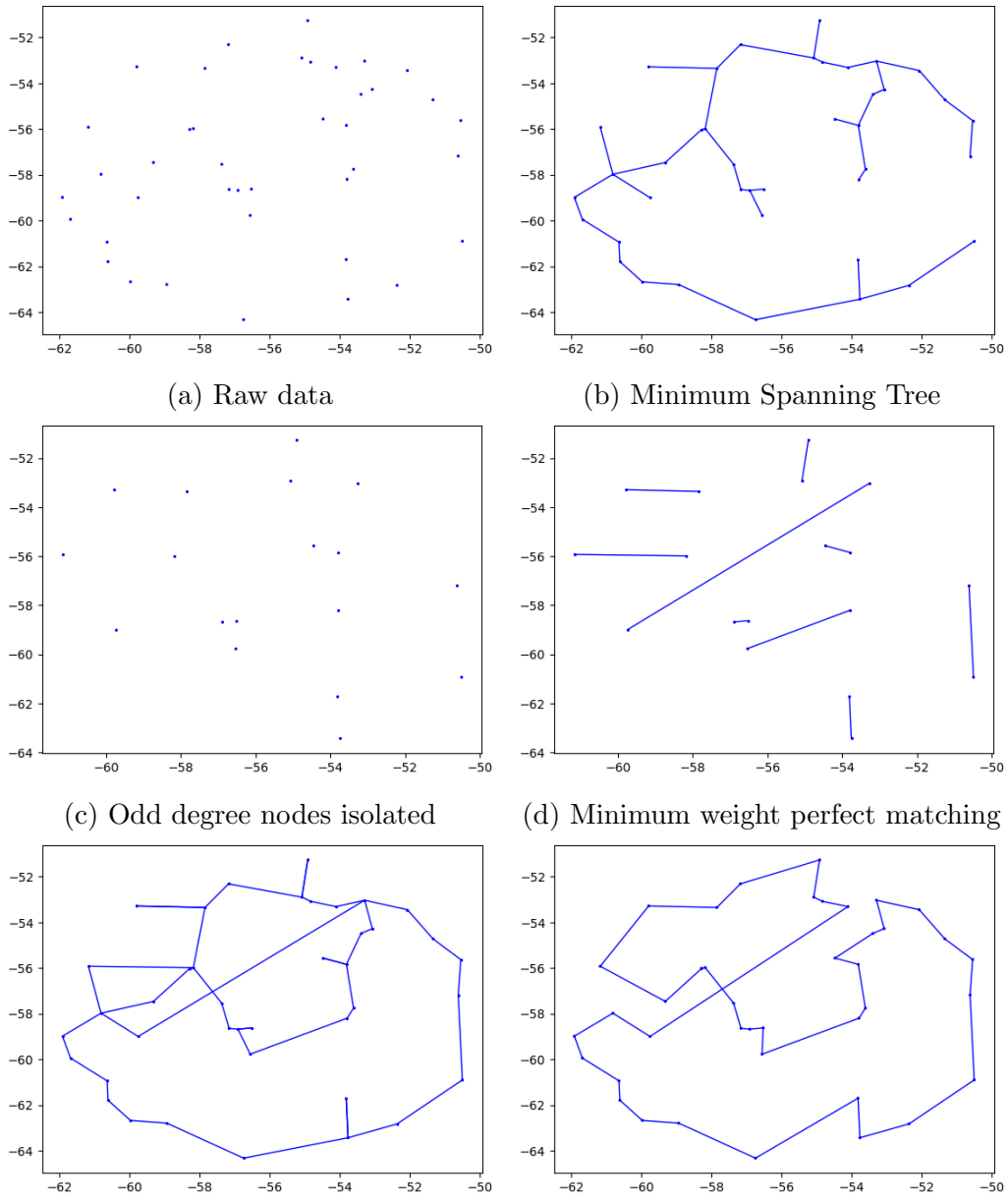
**Minimum Weight Perfect Matching** As described above, the number of odd-degree nodes will always be even in number. We now leverage the Blossom V algorithm [9] to find the minimum weight perfect matching among these odd degree nodes.

**Finding the Eulerian circuit** Now, we find an Eulerian circuit in our graph, which is connected and all nodes have even degrees. We leverage Hierholzer's algorithm for this purpose. Here are the steps to use the same:

- **Choose a starting node:** Pick any node as a starting point.
- **Explore edges:** Traverse the graph, using each edge only once. Choose edges that haven't been used yet, but ensure that we always have a way to continue the traversal if we get stuck (i.e., backtrack if necessary).
- **Repeat until all edges are used:** Continue traversing edges until we return to the starting node and have used all edges in the graph.

## 4.4 Illustrative Example

We have presented an illustrative example below to break the Christofides algorithm into small parts and visualize the working of the algorithm:



(a) Raw data

(b) Minimum Spanning Tree

(c) Odd degree nodes isolated

(d) Minimum weight perfect matching

(e) Euler cycle on MST+matching

(f) Hamiltonian cycle

Figure 1: Christofides Algorithm breakdown through an example

## 4.5 Time Complexity:

Let $n$ be the number of nodes and $m$ be the number of edges in the graph. The time complexity of each of these steps can be stated as follows:

| Algorithm | Time Complexity |
|---|---|
| MST (Kruskal's) | $O(mlogm)$ |
| Odd degree nodes isolation | $O(n)$ |
| Minimum Weight Perfect Matching (Blossom V[9]) | $O(n^2m)$ |
| Eulerian Circuit (Hierholzer[19]) | $O(n+m)$ |
| Hamiltonian Circuit | $O(n)$ |

Table 1: Christofides algorithm time complexity breakdown

Thus, the effective time complexity can be stated as:

$$O((mlogm) + (n) + (n^2m) + (n+m) + (n))$$

In worst case, $O(m) = O(n^2)$. Thus, we have:

$$O(n^2logn + n^4) = O(n^4)$$

# 5 Hybrid Idea

We present a hybrid approach for solving the traveling salesman problem in clustered spaces. This incorporates the idea of using local space search and then integrating these local results with the global data to optimize the time taken to search for an optimal solution. The idea is based on the general civilization pattern and distribution of sites or nodes across a 2-dimensional space. Since, the general distribution of nodes in real life can be seen as groups of nodes that are close to each other, i.e., clusters of nodes distributed all over the space. Intuitively, it would be unoptimal to visit a node in a cluster, go to another cluster, and return to the previous one as we would be covering longer distances more times. This will be further proved mathematically when we expand upon the underlying assumptions and considerations.

## 5.1 Assumptions

We consider the 2-dimensional space distribution of the nodes in the form of distant clusters far apart from each other. More specifically,

$$d(a_i, b_i) << d(a_i, b_j);\ i \neq j,\ \forall i,\ \forall j \tag{2}$$

where,

- $a_i$ is any point in $i_{th}$ cluster
- $b_i$ is any point in $i_{th}$ cluster
- $b_j$ is any point in $j_{th}$ cluster
- $d(x, y)$ is the Euclidean distance among points $x$ and $y$

In simple terms, this means the inter-node distance of a cluster is significantly less than the intra-node distance, i.e., the distance between any 2 nodes of a cluster is much less than the distance between 2 nodes belonging to different clusters.

## 5.2 Clustering

Clustering is a fundamental unsupervised machine-learning technique aimed at grouping data points into clusters or subgroups based on their similarity or proximity. We can use the underlying principles of clustering which are based on the distance between the points in a $n$-dimensional space. In our case, the space happens to be 2-dimensional and the distance measure used is the Euclidean distance. This way, we can group the nearby points and get the clusters required for further processing.

### 5.2.1 Clustering algorithms used

Below is the list of all the clustering algorithms used for further development and comparative study:

- **K-Means [10]:** Given a dataset $X$ with $N$ data points and a predefined number of clusters $K$, the K-Means algorithm iteratively performs the following steps:

  1. **Initialization:** Randomly initialize $K$ cluster centroids $a_1, a_2, ..., a_K$.
  2. **Assignment Step:** For each data point $x_i \in X$, assign it to the nearest centroid based on Euclidean distance:

  $$c^{(i)} = \arg\min_k ||x^{(i)} - a_k||^2 \tag{3}$$

  where $c^{(i)}$ is the index of the cluster assigned to $x_i$.

  3. **Update Step:** Again compute the centroids based on the average of the data points assigned to each cluster:

  $$a_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \tag{4}$$

where $C_k$ is the set of data points assigned to cluster $k$.

4. **Repeat** Repeat the post-initialization steps until convergence.

- **K-Medoids [1]:** Given a dataset $X$ with $N$ data points and a predefined number of clusters $K$, the K-Medoids algorithm iteratively performs the following steps:

  1. **Initialization:** Select $K$ data points randomly as initial medoids.
  2. **Assignment Step:** Assign each $x_i \in X$ to the nearest medoid $m_j$, where $j \in \{1, 2, ..., K\}$, based on some dissimilarity measure (e.g., Manhattan, Euclidean, etc.):

$$c^{(i)} = \arg\min_j d(x_i, m_j) \tag{5}$$

     where $c^{(i)}$ is the index of the cluster assigned to $x_i$ and $d(x_i, m_j)$ is the dissimilarity between $x_i$ and $m_j$.
  3. **Update Step:** For each cluster, try replacing each medoid with each non-medoid data point and compute the total cost (e.g., total dissimilarity) of the clustering. Select the configuration with the lowest total cost.
  4. **Repeat** Repeat the post-initialization steps until convergence.

Other algorithms [22] that can be used include DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [2], Agglomerative Hierarchical Clustering [11], and Expectation Maximisation [7].

### 5.2.2 Finding the optimal number of clusters

For some of the above-listed algorithms, it is required to pass in the number of clusters we wish to learn the data upon beforehand. For this, we leverage various techniques and mathematical scores (quantities). These include:

- **Elbow method:** In this method [6], we compute the within-cluster sum of squares (WCSS) for various numbers of clusters. WCSS measures the compactness of the clusters. As the number of clusters increases, WCSS tends to decrease because smaller clusters are formed. However, the rate of decrease slows down after a certain point. The "elbow" point on the plot indicates the

optimal number of clusters, where adding more clusters doesn't significantly reduce WCSS.

WCSS is defined as:

$$\text{WCSS} = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2 \tag{6}$$

where,

- $k$ is the number of clusters
- $C_i$ represents the $i - th$ cluster
- $\mu_i$ is the centroid (mean) of the points in the cluster $C_i$
- $x$ represents each data point in the cluster $C_i$
- $\|x - \mu_i\|^2$ represents the squared Euclidean distance between each data point and its centroid.

- **Silhouette Score:** Silhouette score [[14], [18]] measures how similar an object is to its own cluster compared to other clusters. The score ranges from -1 to 1, where a high value indicates that the object is well-matched to its own cluster and poorly matched to neighboring clusters. We can calculate the silhouette score for different numbers of clusters and choose the one with the highest silhouette score as the optimal number of clusters.

$$\text{Silhouette Score} = \frac{1}{N} \sum_{i=1}^{N} \frac{b_i - a_i}{\max(a_i, b_i)} \tag{7}$$

where,

- $N$ is the total number of samples
- $a_i$ is the mean distance from the $i_{th}$ sample to other samples in the same cluster
- $b_i$ is the mean distance from the $i_{th}$ sample to samples in the nearest cluster that the $i_{th}$ sample is not a part of

The Silhouette Score measures how similar an object is to its own cluster compared to other clusters.

## 5.3 Primary Approach

The primary idea is to solve the clusters internally first, thereby forming internal Hamiltonian paths, then joining these paths end-to-end to form a Hamiltonian cycle or the solution to our traveling salesman problem.

### 5.3.1 Generating the internal Hamiltonian cycles

The next challenge is to actually generate the Hamiltonian cycle among all the nodes inside a cluster. This step has to be performed for each cluster and can be done in parallel if such a processing architecture is available (discussed later). To perform this, we can either use a deterministic algorithm such as the standard dynamic programming solution for TSP, or an approximate algorithm such as the Christofides algorithm [21] or the Nice Ears approach [13].

### 5.3.2 Connecting the clusters

Now, that we have generated the internal cycles, we proceed to connect the clusters. This step can be done in several ways depending upon the number of clusters in the dataset, processing time available, and tolerance level acceptable for the final solution. For this, the most near to optimal solution feasible should be used as this would contribute greatly to the final cost of the path or the total distance covered.

The underlying assumption that the clusters are far apart from each other compared to their diameters, allows us to proceed by taking the clusters as single points represented by their centers for inter-cluster optimizations and processing. Thus, we now have an auxiliary graph that is complete, undirected, and made of nodes representing the cluster centers.

Primarily, we have 2 cases based on the number of clusters:

**Approach 1** ($n <= 20$)**:** Since, the number of clusters is significantly less in this case, we can use the dynamic programming approach to solve the inter-cluster connection problem, giving the most optimal solution.

**Approach 2** ($n > 20$)**:** The number of clusters being large in this case forces us to use approximate approaches such as the Christofides algorithm, thereby adding to the error of the final solution.

Now, that we have solved the inter-cluster connection problem in an abstract manner, we can proceed to merge the intra-cluster solutions with it. This comes with another problem. We found the Hamiltonian cycle solutions for each of the

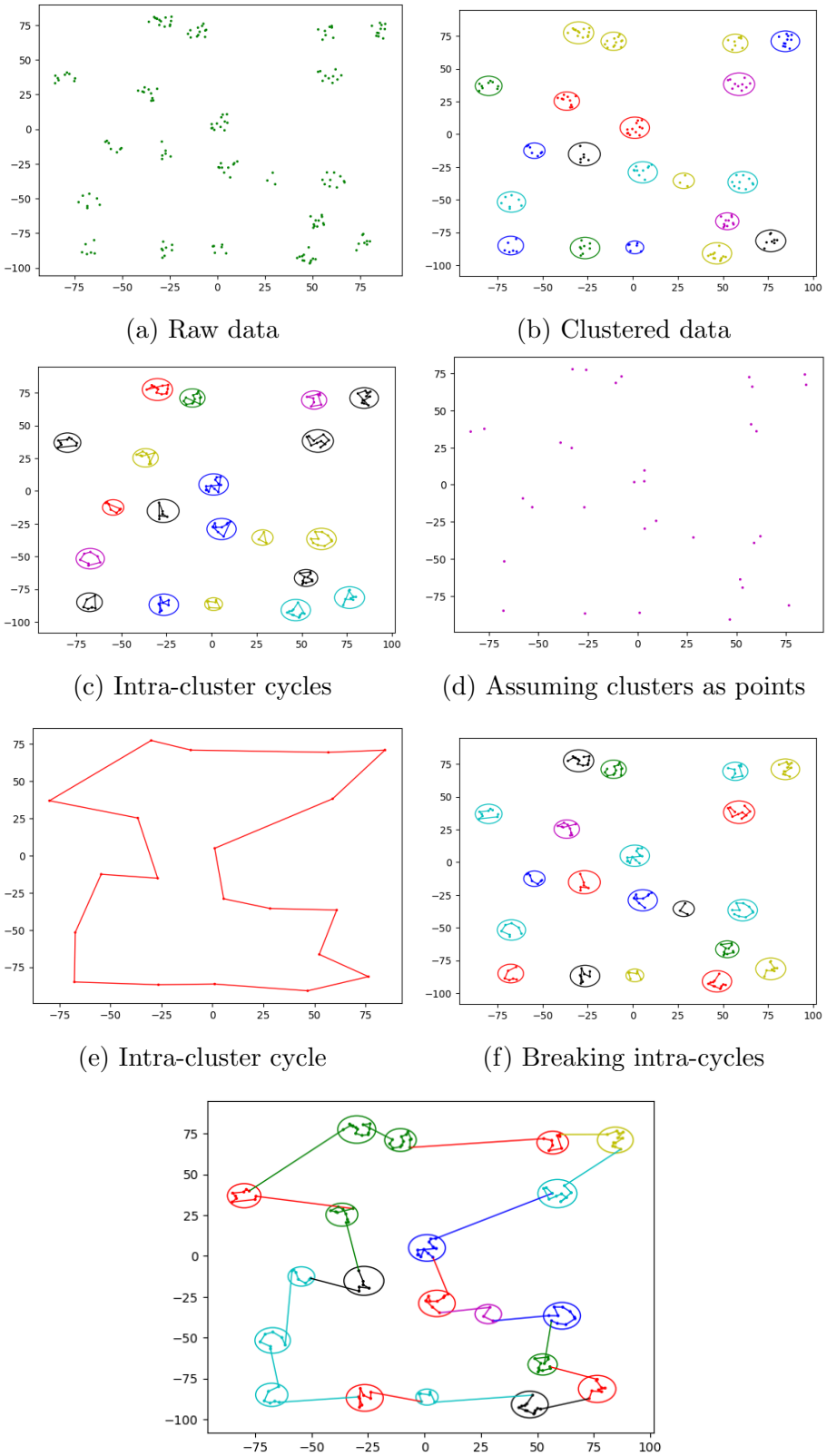
clusters. These cycles have to be converted into chains to incorporate the inter-cluster incoming and outgoing edges.

**Heuristic for edge removal:** To solve this, we use a novel heuristic-based approach to remove an edge from each of the intra-cluster cycles independently of the other. Although this will generate some error, it will be negligible compared to the total solution (proved ahead). Now, that we know the 2 neighboring clusters for each cluster, we know the directions of the incoming and outgoing edges for each of them. Again, assuming the neighboring clusters to be single points located at their centers, we can assume the inter-cluster edges coming from these centers. Now, the total cost of this operation would be:

$$\text{cost} = \text{inter-edge}_1 + \text{inter-edge}_2 - \text{intra-edge removed} \tag{8}$$

## 5.4 Illustrative Example

We have presented an illustrative example of the steps followed by the above-proposed algorithm to solve TSP in a clustered space as in Figure[2]:



(a) Raw data

(b) Clustered data

(c) Intra-cluster cycles

(d) Assuming clusters as points

(e) Intra-cluster cycle

(f) Breaking intra-cycles

(g) Final Solution

Figure 2: Hybrid Approach steps breakdown with illustrative example

## 5.5 Time Complexity Analysis

The time complexity of the current solution can be broken down into a few parts as shown below:

| Part | Time Complexity |
|---|---|
| Clustering | $O(\alpha)$ |
| Intra-cluster Processing | $O(\beta)$ |
| Intra-edge Removal | $O(\gamma)$ |
| Inter-cluster Processing | $O(\delta)$ |

Table 2: Hybrid Approach time complexity analysis

The effective time complexity of the entire hybrid approach can be stated as follows:

$$O(\alpha + \beta + \gamma + \delta)$$

**Computing $\alpha$:** $\alpha$ depends upon the clustering algorithm we choose to embed in our pipeline along with the time complexity of computing the hyperparameters to be supplied (example: number of the cluster for the K-means algorithm, $\epsilon$ and $MinPts$ for the DBSCAN algorithm, etc.). For illustration purposes and further analysis, we use the K-means algorithm because of its simplicity of implementation and reliability. For this, we have:

$$O(\alpha) = O(NTK)$$

where,

- $N$: Number of points in the dataset
- $T$: Number of iterations for convergence
- $K$: Number of clusters

Since the number of clusters and a number of iterations are to be kept small for our purpose, we can approximate these terms by a small constant real number. Hence, for our case:

$$O(\alpha) = O(n)$$

**Computing $\beta$:** $\beta$ depends on the total number of clusters and the maximum number of points in a cluster. Since we are using the Christofides algorithm for

intra-cluster processing, the time complexity can be computed using:

$$\sum_{i=1}^{m} O(|C_i|^4) \leq m \times O(|C|_{max}^4) = O(m \times |C|_{max}^4) = O(\beta)$$

where $|C|_{max}$ represents the maximum number of points in a cluster over all the clusters, apart from this, we can leverage the fact that the intra-cluster processing of one cluster is independent of the other. This enables us to parallelly process each of the clusters. Consider a system with $p$ number of processors or cores. Modifying the above value of $O(\beta)$:

$$O(\beta) = O(\left\lceil \frac{m}{p} \right\rceil \times |C|_{max}^4)$$

**Computing $\gamma$:** $\gamma$ is based on the heuristic approach used to eliminate the edges. For our case, we use the above-explained approach with a straight-forward time complexity of $\sum_{i=1}^{m} O(|C_i|) = O(n)$.

**Computing $\delta$:** $\delta$ is determined by the number of clusters that decide the algorithm to be used for processing. For approach 1 [5.3.2], the time complexity is $O(m^2 \times 2^m)$. For approach 2 [5.3.2], we use the Christofides Algorithm with the time complexity of $O(m^4)$.

Summing up, the overall time complexity of the hybrid approach can be stated as:

- **Approach 1 [5.3.2]**

$$O(n + \left\lceil \frac{m}{p} \right\rceil \times |C|_{max}^4 + m^2 \times 2^m)$$

- **Approach 2 [5.3.2]**

$$O(n + \left\lceil \frac{m}{p} \right\rceil \times |C|_{max}^4 + m^4)$$

## 5.6 Solution Analysis

### 5.6.1 Definitions

For further proof and analysis, we use certain notations which are defined as follows:

- **Node Set** ($S$): A set which contains all the nodes of the graph.

- **Intra-cluster distance** ($d_{ixy}$): The distance between the nodes $x$ and $y$ where the nodes $x$ and $y$ belong to the $i_{th}$ cluster.
- **Inter-cluster distance** ($D_{ijxy}$): The distance between the nodes $x$ and $y$ where the nodes $x$ and $y$ belong to $i_{th}$ and $j_{th}$ clusters respectively and $i \neq j$.
- **Maximum intra-cluster distance** ($A$)**:** We define it as:

$$A = max(d_{ixy});\ \forall i\ \forall x\ \forall y \tag{9}$$

- **Minimum inter-cluster distance** ($B$)**:** We define it as:

$$B = min(D_{ijxy});\ \forall i\ \forall j\ \forall x\ \forall y \tag{10}$$

- **Separation factor** ($\alpha$): A positive constant ($> 1$) that relates the inter-cluster and intra-cluster distance as follows:

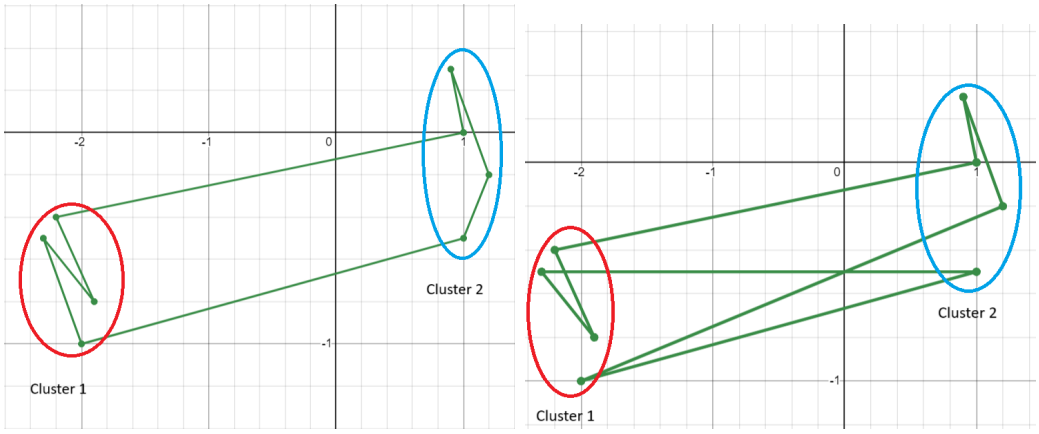
$$B > \alpha * A \tag{11}$$

- **Cluster:** A group of nodes that are close to each other, i.e., the distance between any pair of nodes within the cluster follows 11.
- **Cluster Diameter** ($r_i$)**:** We define the diameter of the $i_{th}$ cluster as the diameter of the smallest circle circumscribing the cluster (i.e. containing all the points inside or on it).

### 5.6.2 Proof of optimality under constraints

Let the total number of nodes be $n$, the total number of clusters be $m$ and the total distance covered for the proposed strategy solution be $\beta$. In the case of the optimal solution, we have a total of $m$ inter-cluster links or edges that we cover and $n - m$ intra-cluster links. Now, let us assume that the method proposed does not yield the most optimal solution, then there is at least 1 case where we go away from a cluster before visiting all of its nodes and return to it for completion. Thus, the total number of inter-cluster links will be $m+1$ and the total distance will be $\beta + A - B$ in the best case. Now, according to our initial assumption of cluster separation factor ($\alpha$), $A > \alpha * B$. The new solution requires $\beta + (\alpha - 1) * B + \epsilon$ distance, which is $> \beta$ as $\alpha > 1, \epsilon > 0$ more distance to be covered than the proposed solution. Hence, the

proposed strategy yields optimal results. An illustration of this can be seen in the Figure 3.

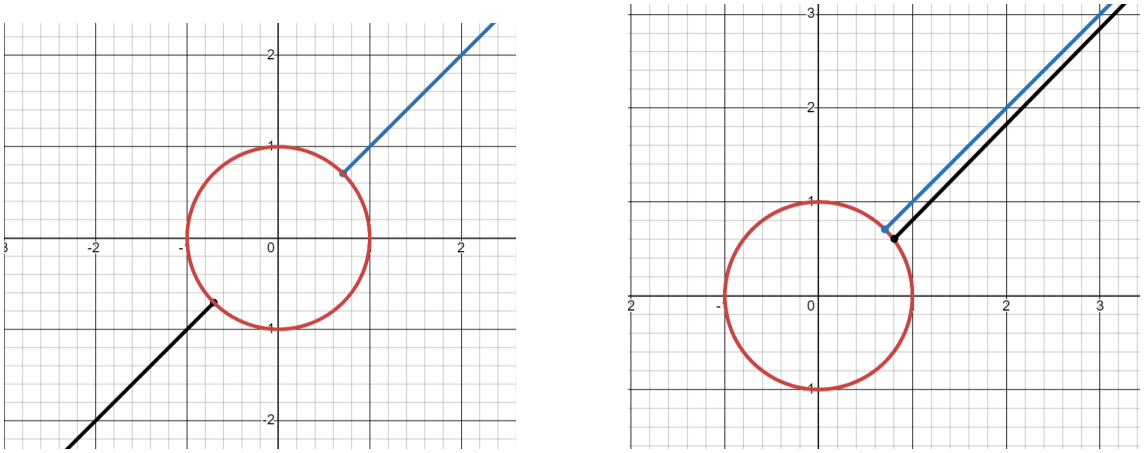



(a) Proposed strategy (b) Unoptimal contradiction

Figure 3: Proof by contradiction

### 5.6.3 Approximation and Error analysis

**Breaking the cycles:** The best and worst cases of the cycle-breaking conditions are illustrated in Figure[4].



(a) Best Case Scenario (b) Worst Case Scenario

Figure 4: Solution analysis (diameter)

From Figure[4], we can clearly infer that in the worst case, we will have an extra cost of $3 * r_i$ for the $i^{th}$ cluster in the worst case. But, this will almost never be achieved as we are using the removal heuristic[5.3.2].

Now, let us assume that $H_i$ is the convex hull of the cluster. First, we claim that there exists a pair of points with distance $A_i$ both of which lie on $H_i$ based on

[[20]]. Now that we have a convex polygon for further processing, the max ratio of the $r_i$ and $A_i$ is $2/\sqrt{3}$. This can be proved as follows. Let's fix the diameter of the circumscribing circle, then the triangle that produces the largest ratio is equilateral. If we still fix $D(C)$ and try to construct a polygon $C$ in this circle, we can always embed a triangle into $C$, which has the same diameter as $C$. However, this diameter is $\geq$ to the diameter of an equilateral triangle with the same circumscribing circle. Hence, the ratio $\frac{D(C)}{d(C)}$ is $\leq$ to the ratio of an equilateral triangle. Thus, in worst case, the excess cost will be:

$$3 \times r_i = 3 \times \left(\frac{2}{\sqrt{3}} A_i\right) = 2\sqrt{3} A_i$$

For Approach 1[5.3.2], we are applying the Christofides algorithm for intra-cluster processing. Our solution will be at most 1.5 times worse (larger) than the most optimal solution for the intra-cluster edges. Now, since we are using dynamic programming for inter-cluster solutions, there will be no error for that and hence, no more cost than the most optimal solution. Apart from this, we have added cost from [5.6.3] Hence, the total cost will be:

1.5*(Intra-cluster optimal) + (Inter-cluster optimal) + $2\sqrt{3} * \sum_{i=1}^{m} A_i$

For Approach 2[5.3.2], we are applying the Christofides algorithm for intra-cluster processing as well as inter-cluster processing. Our solution will be at most 1.5 times worse (larger) than the most optimal solution with an added cost [5.6.3]. Hence, the total cost in the worst case will be:

1.5*(Global Optimal Solution) + $2\sqrt{3} * \sum_{i=1}^{m} A_i$

# 6 Comparative Study

## 6.1 Ant Colony Optimization (Ant System Approach)

Ant System (AS) is one of the earliest and most fundamental algorithms in the field of Ant Colony Optimization (ACO). It mimics the foraging behavior of real ants to construct solutions to optimization problems. It can be described as:

Description of various components of the algorithm:

**Pheromone Trails:** Pheromone trails represent the collective memory of the artificial ant colony. They guide the construction of solutions by influencing the probability of selecting edges during the tour construction process. Pheromone levels are updated based on the quality of the solutions found by the ants.

**Algorithm 1** Ant System (AS)

**Input:**
$N$: Total cities
$M$: Total ants
$T$: Total iterations
$\rho$: Pheromone evaporation rate
$\alpha$: Trail importance factor
$\beta$: Heuristic importance factor
$d_{ij}$: Distance between city $i$ and city $j$
$\tau_{ij}$: Pheromone level on edge $(i, j)$
$\eta_{ij}$: Heuristic information between city $i$ and city $j$

**Initialization:**
Initialize pheromone levels on all edges: $\tau_{ij} = \tau_0$
Place $M$ ants at randomly chosen cities
**for** $t = 1$ to $T$ **do**
  **for** each ant $k = 1$ to $M$ **do**
    Initialize empty solution $S_k$
    **for** $i = 1$ to $N - 1$ **do**
      Compute probability $p_{ij}$ of selecting city $j$ from city $i$
      Select next city $j$ using roulette wheel selection based on $p_{ij}$
      Add city $j$ to $S_k$
    **end for**
    Complete the tour $S_k$ by adding the remaining unvisited city
    Update pheromone levels on the edges traversed by ant $k$
  **end for**
  Evaporate pheromone on all edges: $\tau_{ij} = (1 - \rho)\tau_{ij}$
  Update pheromone levels based on the solutions constructed by ants
**end for**=0

**Heuristic Information:** In addition to pheromone levels, ants use heuristic information to bias their decisions. Heuristic values typically reflect the distance between cities in the TSP. Heuristic information helps balance the exploration and exploitation of the solution space.

**Exploration vs. Exploitation:** Ant System maintains a balance between searching for new, potentially better solutions and refining existing promising solutions. Exploration is achieved through probabilistic decision-making during tour construction, allowing ants to explore diverse regions of the solution space. Exploitation occurs through the reinforcement of pheromone trails on shorter tour segments, guiding ants toward promising regions of the solution space.

## 6.2 Particle Swarm Optimization

This is a bio-inspired approach [[8]]. It searches concurrently for increasingly optimal solutions. Each solution can be perceived as a particle that moves over the solution space. The particles are compared based on the least distance solution provided by each one of them. We associate a velocity value with each particle to help it navigate the solution space in an optimized and reactive manner. These particles then leverage their past experiences as well as their neighbors' best experiences to correct their respective velocities and move to a better position in the solution space.

The velocity and position of every particle are updated in each iteration. Basically, particle swarm optimization provides a paradigm for problem-solving and navigating through the solution space like many other algorithms such as the Firefly Algorithm, Bat Algorithm, and Simulated Annealing.

We use the classical version of the problem with the modifications discussed in [[5], [17], [16]]. Each particle has associated with it a position and a velocity. Besides this, each one of them knows their own position, the best position they ever achieved, and their neighbors' best positions.

Now, to update the position or perform a move, we use a combination of the best position achieved by the particle so far, the best position achieved by its neighbors so far, and the previous position. The details of the algorithm and updation equations are stated below.

$$\mathbf{v}_{ij}(t+1) = w\mathbf{v}_{ij}(t) + c_1 r_1(\mathbf{p}_{ij} - \mathbf{x}_{ij}(t)) + c_2 r_2(\mathbf{g}_{ij} - \mathbf{x}_{ij}(t)) \tag{12}$$

where:

- $v_{ij}(t+1)$ is the velocity of particle $i$ in dimension $j$ at time $t+1$.

**Algorithm 2** Particle Swarm Optimization (PSO) for TSP

**Initialization**:
Initialize swarm of particles with random positions $\mathbf{X} \in \mathbb{R}^{N \times D}$
Initialize particle velocities randomly $\mathbf{V} \in \mathbb{R}^{N \times D}$
Initialize personal best positions for each particle $\mathbf{P} \in \mathbb{R}^{N \times D}$
Initialize global best position $\mathbf{G} \in \mathbb{R}^{D}$
**while** stopping criterion not met **do**
  **for** each particle **do**
    **Evaluation**:
    Evaluate fitness of current position {Total distance traveled}
    **if** fitness better than personal best **then**
      Update personal best position
    **end if**
    **if** fitness better than global best **then**
      Update global best position
    **end if**
    **Updating Particle Velocity**:
    Update particle velocity using Eq. (12)
    **Updating Particle Position**:
    Update particle position
  **end for**
**end while**
**Termination**:
Return global best position =0

- $w$ is the inertia weight controlling the influence of the previous velocity.
- $c_1$ and $c2$ are acceleration coefficients representing the influence of personal and global best positions, respectively.
- $r_1$ and $r_2$ are random values between 0 and 1.
- $p_{ij}$ is the personal best position of particle $i$ in dimension $j$.
- $g_{ij}$ is the global best position among all particles in dimension $j$.
- $x_{ij}(t)$ is the current position of particle $i$ in dimension $j$ at time $t$.

## 6.3 Simulated Annealing

It is a probability-based method to explore the solution space with a mechanism in place to escape local minimums. It consists of a parameter - temperature, which is used to generate the probability by which we accept less optimal solutions. While exploring the solution space or modifying our current solution, if the transition results in a better solution, we accept it with 100% probability, while if the solution is less optimal, it is chosen according to a probability function that takes into account the amount of loss we will have if we choose it. The said probability function is defined as follows:

$$p = \begin{cases} 1 & \text{, if } f(y) \geq f(x) \\ e^{-(f(y)-f(x))/T} & \text{, otherwise} \end{cases}$$

where,

- $p$: the probability of accepting the new solution candidate, $y$
- $x$: initial solution candidate (In this case, that is a route)
- $y$: new solution candidate
- $f(x)$: is the function that measures the performance of the solution candidate (fitness function or objective function)
- $T$: the temperature which is the control parameter

This algorithm stops converging after a high number of steps and hence doesn't converge to the most optimal solution after a point. The algorithm can be summarized as shown in 5:

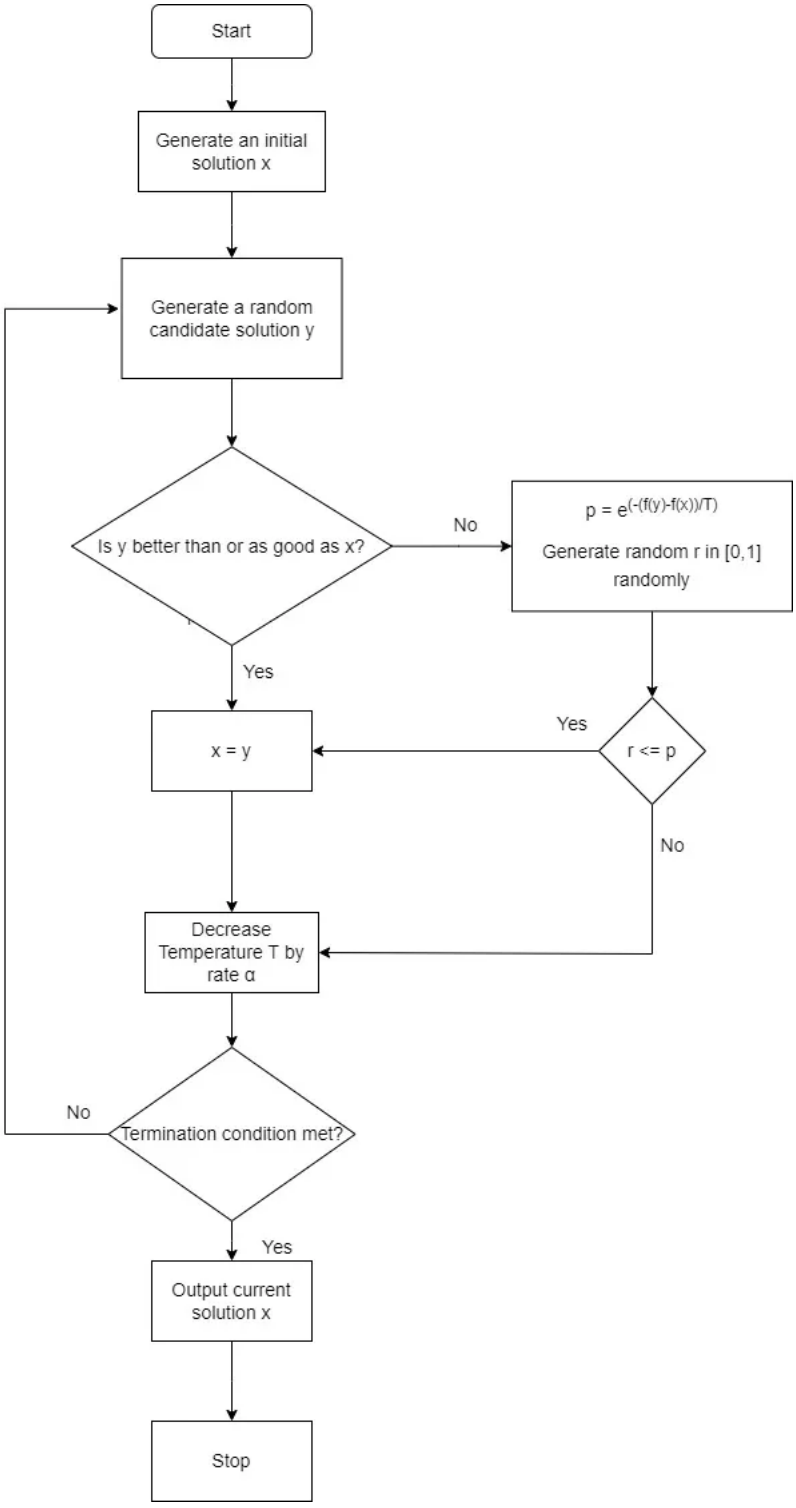


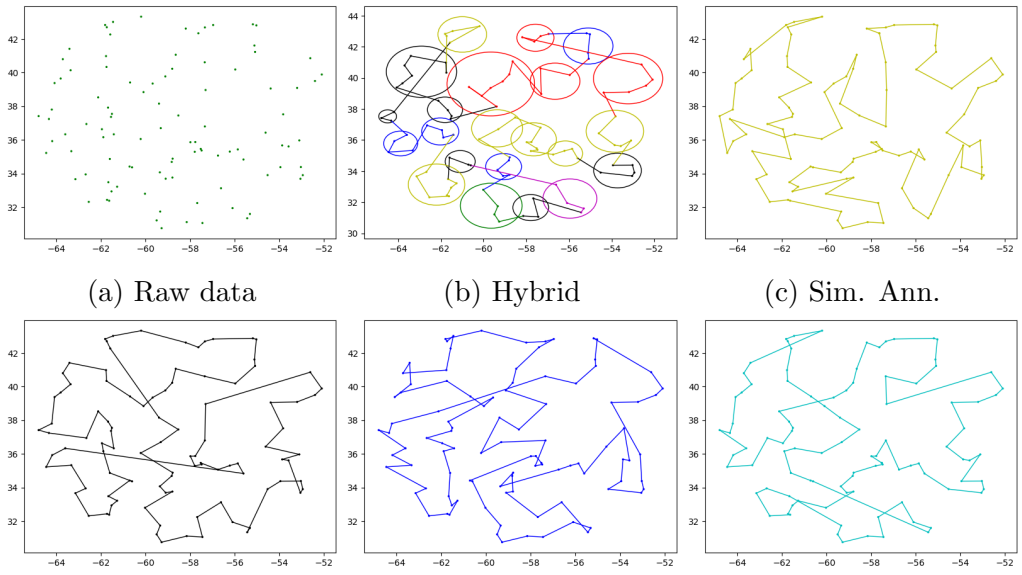

Figure 5: Simulated Annealing Algorithm

**Algorithm 3** Simulated Annealing for TSP

Initialize $T_{\text{max}}$, $T_{\text{min}}$, $T_{\text{current}}$
Initialize current tour $T_{\text{current}}$
Set initial temperature $T_{\text{current}} = T_{\text{max}}$
**while** $T_{\text{current}} > T_{\text{min}}$ **do**
Generate a new tour $T_{\text{new}}$ by making a small change to $T_{\text{current}}$
Calculate the cost difference $\Delta C = \text{cost}(T_{\text{new}}) - \text{cost}(T_{\text{current}})$
  **if** $\Delta C < 0$ **then**
Accept $T_{\text{new}}$ as the current tour
  **else**
Generate a random number $r \in [0, 1]$
    **if** $r < e^{-\Delta C / T_{\text{current}}}$ **then**
Accept $T_{\text{new}}$ as the current tour
Update temperature $T_{\text{current}} = T_{\text{current}} \times \alpha$ where $\alpha$ is a cooling rate
**return** $T_{\text{current}}$ =0

## 6.4 Simulations and Results

### 6.4.1 Random distribution, small number of points



(a) Raw data (b) Hybrid (c) Sim. Ann.

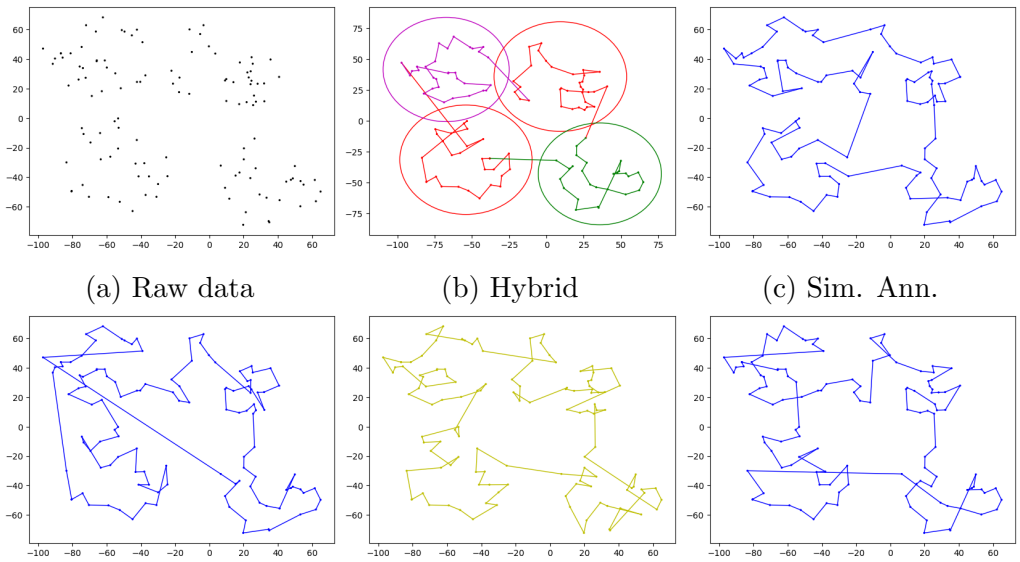
(d) PSO (e) ACO (f) Christofides

Figure 6: Comparative Study - Random distribution, less points

| Algorithm | Hybrid | Sim. Ann. | PSO | ACO | Christofides |
|---|---|---|---|---|---|
| Distances | 106.150 | 105.607 | 104.444 | 114.1788 | 101.947 |

Table 3: Comparative Study - Random distribution, less points

### 6.4.2 Overlapping clusters



(a) Raw data (b) Hybrid (c) Sim. Ann.

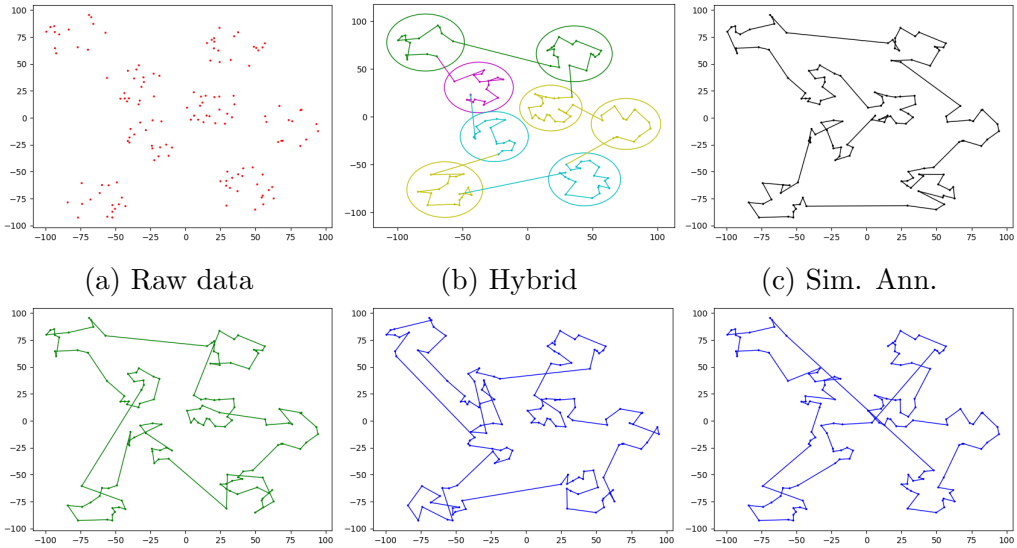
(d) PSO (e) ACO (f) Christofides

Figure 7: Comparative Study - Overlapping Clusters

| Algorithm | Hybrid | Sim. Ann. | PSO | ACO | Christofides |
|---|---|---|---|---|---|
| Distances | 1217.855 | 1163.033 | 1302.163 | 1282.478 | 1221.201 |

Table 4: Comparative Study - Overlapping Clusters

### 6.4.3 Close clusters



(a) Raw data (b) Hybrid (c) Sim. Ann.

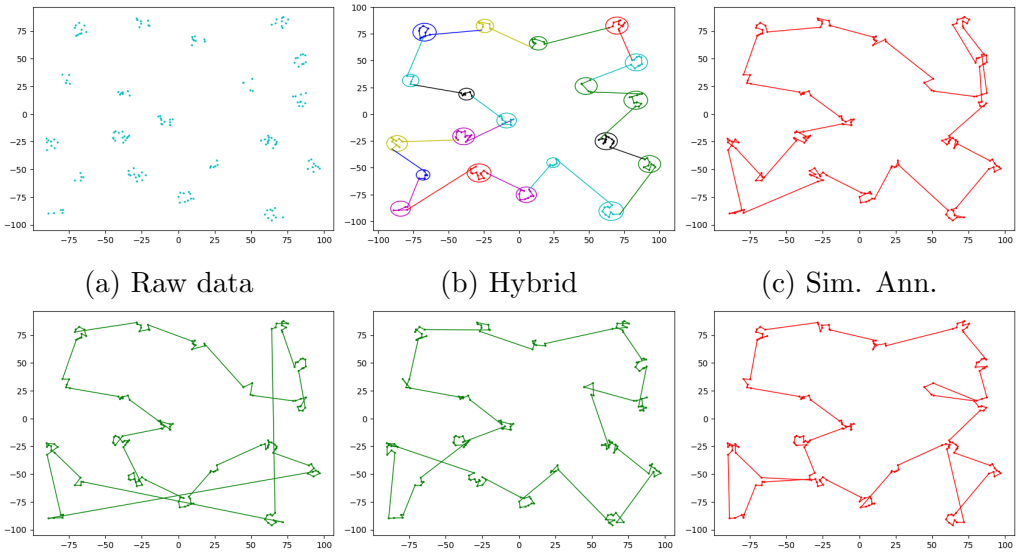
(d) PSO (e) ACO (f) Christofides

Figure 8: Comparative Study - Close Clusters

| Algorithm | Hybrid | Sim. Ann. | PSO | ACO | Christofides |
|---|---|---|---|---|---|
| Distances | 1459.894 | 1495.319 | 1570.428 | 1633.154 | 1522.854 |

Table 5: Comparative Study - Close Clusters

### 6.4.4 Distant clusters



(a) Raw data (b) Hybrid (c) Sim. Ann.

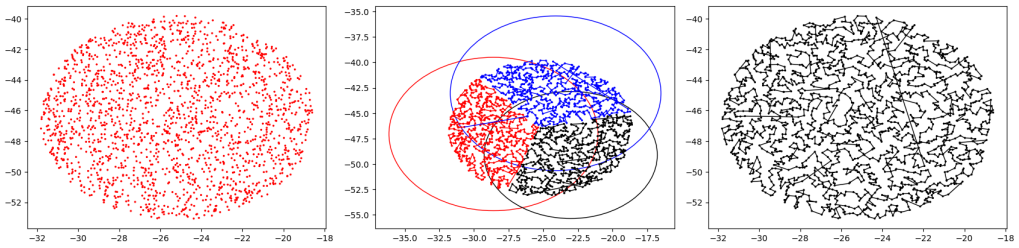
(d) PSO (e) ACO (f) Christofides

Figure 9: Comparative Study - Distant Clusters

| Algorithm | Hybrid | Sim. Ann. | PSO | ACO | Christofides |
|---|---|---|---|---|---|
| Distances | 1245.230 | 1337.726 | 1598.132 | 1409.744 | 1316.431 |

Table 6: Comparative Study - Distant Clusters

### 6.4.5 Dense spaces, large number of points



(a) Raw data (b) Hybrid (c) Christofides

Figure 10: Comparative Study - Temporal Analysis

| Algorithm | Hybrid | Christofides |
|---|---|---|
| Distances | 459.529 | 448.294 |
| Time Taken (millis) | 59090 | 151695 |

Table 7: Comparative Study - Temporal Analysis

### 6.5 Results analysis

From the above simulations, we infer that on completely random or far from assumed spaces, our algorithm yields comparable results as the state-of-the-art algorithms. However, if the spaces are as assumed (distant clusters), our algorithm yields significantly better results for a comparable time to run. Also, in spaces with a very large number of points ($> 1000$), our algorithm yields slightly less optimal solutions (around 2% worse in the above instance) in far less time (at least 2 times less).

## 7 Conclusion and future work

From the qualitative and quantitative analysis, and comparative studies presented above, we have shown that the hybrid approach we present to solve the traveling salesman problem in clustered spaces performs better and is more practical and faster for dense spaces.

Despite the detailed analysis presented above, there is still room for improvement in the above approach. The heuristic used for breaking intra-cycles, clustering algorithm, etc. can be further analyzed and developed for better results.

# Bibliography


[1] Norazam Arbin et al. "Comparative analysis between k-means and k-medoids for statistical clustering". In: *2015 3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*. IEEE. 2015, pp. 117–121.

[2] Dingsheng Deng. "DBSCAN Clustering Algorithm Based on Density". In: *2020 7th International Forum on Electrical Engineering and Automation (IFEEA)*. 2020, pp. 949–953. DOI: 10.1109/IFEEA51475.2020.00199.

[3] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization". In: *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[4] Marco Dorigo and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66.

[5] Elizabeth FG Goldbarg, Marco C Goldbarg, and Givanaldo R de Souza. "Particle swarm optimization algorithm for the traveling salesman problem". In: *Traveling Salesman Problem* 1 (2008), pp. 75–96.

[6] Hestry Humaira and R Rasyidah. "Determining the appropriate cluster number using elbow method for k-means algorithm". In: *Proceedings of the 2nd Workshop on Multidisciplinary and Applications (WMA) 2018, 24-25 January 2018, Padang, Indonesia*. 2020.

[7] Xin Jin and Jiawei Han. "Expectation Maximization Clustering". In: Jan. 2016, pp. 1–2. DOI: 10.1007/978-1-4899-7502-7_344-1.

[8] James Kennedy and Russell Eberhart. "Particle swarm optimization". In: *Proceedings of ICNN'95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[9] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1 (2009), pp. 43–67.

[10] Brady Lund and Jinxuan Ma. "A review of cluster analysis techniques and their uses in library and information science research: k-means and k-medoids clustering". In: *Performance Measurement and Metrics* 22.3 (2021), pp. 161–173.

[11] Daniel Müllner. "Modern hierarchical, agglomerative clustering algorithms". In: *arXiv preprint arXiv:1109.2378* (2011).

[12] Krit Salahddine et al. “The implementation of Kruskal’s algorithm for minimum spanning tree in a graph”. In: *E3S Web of Conferences*. Vol. 297. EDP Sciences. 2021, p. 01062.

[13] András Sebő and Jens Vygen. “Shorter tours by nicer ears: 7/5-approximation for graphic TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs”. In: *arXiv preprint arXiv:1201.1870* (2012).

[14] Ketan Rajshekhar Shahapure and Charles Nicholas. “Cluster quality analysis using silhouette score”. In: *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2020, pp. 747–748.

[15] Congming Shi et al. “A quantitative discriminant method of elbow point for the optimal number of clusters in clustering algorithm”. In: *Eurasip Journal on Wireless Communications and Networking* 2021 (2021), pp. 1–16.

[16] Yuhui Shi et al. “Particle swarm optimization: developments, applications, and resources”. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*. Vol. 1. IEEE. 2001, pp. 81–86.

[17] Yuhui Shi and Russell Eberhart. “A modified particle swarm optimizer”. In: *1998 IEEE International conference on evolutionary computation proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*. IEEE. 1998, pp. 69–73.

[18] Meshal Shutaywi and Nezamoddin N Kachouie. “Silhouette analysis for performance evaluation in machine learning with applications to clustering”. In: *Entropy* 23.6 (2021), p. 759.

[19] L.J. Simenthy, R. Bobanand, and M. Krishnan. “A comparison based analysis of euler circuit finding algorithms”. In: 10 (Jan. 2015), pp. 2511–2514.

[20] Vaclav Skala and Zuzana Majdisova. “Fast algorithm for finding maximum distance with space subdivision in E 2”. In: *Image and Graphics: 8th International Conference, ICIG 2015, Tianjin, China, August 13-16, 2015, Proceedings, Part II 8*. Springer. 2015, pp. 261–274.

[21] David P Williamson. “An Experimental Evaluation of the Best-of-Many Christofides’ Algorithm for the Traveling Salesman Problem”. In: (2015).

[22] Donghua Yu et al. “An improved K-medoids algorithm based on step increasing and optimizing medoids”. In: *Expert Systems with Applications* 92 (2018), pp. 464–473.