SOLVING TRAVELING SALESMAN PROBLEM VIA CLUSTERING AND A NEW ALGORITHM FOR MERGING TOURS

Anonymous authors
Paper under double-blind review

ABSTRACT

The use of clustering-based methods to solve the large-scale Traveling Salesman Problem is a common approach among researchers. This approach consists of three main stages: clustering, solving the problem within each cluster, and combining the solutions into a single cycle. Our focus is on the last stage. An effective algorithm has been introduced for pairwise combinations of cycles. Additionally, the connection between the error and the stage of solving subproblems in the process of merging solutions has been investigated.

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is one of the most renowned problems in discrete optimization, involving finding the shortest path that visits each specified city or vertex exactly once and returns to the origin, forming a closed loop. Data about the distances between vertices can be represented in two main ways: via a distance matrix or by coordinates on a plane. This paper uses the latter method, referring to the problem as the Euclidean TSP (ETSP).

In general, the problem is NP-hard Deineko et al. (2022), as is its variant, the ETSP Antoniadis et al. (2020), and neither has a polynomial-time algorithm.

TSP is utilized in various fields, including logistics, transportation planning, and network routing to optimize data transmission paths between network nodes, as well as in the optimal control of drones Spyridis et al. (2023), Schaumann et al. (2024).

In solving real-world problems, the number of vertices can be so large that employing algorithms for exact solutions is impractical. In such cases, a decomposition approach may be employed, involving:

1. Clustering: identifying subproblems of smaller dimensions;
2. Solving: solving each subproblem by finding a cycle within a cluster;
3. Merging: forming the final solution by merging the solutions of the subproblems.

This approach is widely discussed in the literature: Anaya Fuentes et al. (2018), Romanuke (2023), Jaradat et al. (2019). Most papers propose new methods for breaking down a problem into smaller subproblems and provide algorithms for solving each subproblem. However, algorithms for merging the solutions to these subproblems are often neglected. In our work, we propose a new merging algorithm based on the pairwise merging of clusters and compare it with others. This algorithm allows you to achieve more stable and accurate results.

The main aim of our work is an analysis of the errors associated with this approach, evaluating the impact of errors at each stage on the overall solution error. The use of clustering restricts the solution space of TSP, as transitions between clusters are limited. Whether the optimal solution falls within this restricted area cannot be determined during the clustering phase. Therefore, the study examines errors in the second and third stages and their interrelations.

The structure of this paper is as follows: Chapter 2 provides an overview of algorithms used in the decomposition approach. Chapter 3 describes the new algorithm we propose, along with the algorithms for comparison. Chapter 4 presents the design of the numerical experiments and their results. Chapter 5 draws conclusions from the findings.

2 Literature Review

2.1 Clustering and Subproblem Solving Algorithms

There are numerous methods for decomposing the Traveling Salesman Problem into subproblems. Khachaturov et al. (2000) discusses the division of a set of points into non-overlapping subsets using partition quality functions. This approach is multi-step, includes adjustment procedures, and is labor-intensive.

The method for hierarchical clustering based on the construction of dendrograms is proposed in Tsai & Chiu (2006). The quality of the clusters obtained can be inconsistent.

Liao et al. Liao & Liu (2018) propose a decomposition of the general problem into subproblems using the Density Peaks Clustering algorithm. The algorithm assumes that cluster centers are surrounded by neighbors with lower local density and focuses on identifying centers with higher local density.

The k-means algorithm, one of the most prevalent machine learning algorithms, is extensively utilized as outlined in Anaya Fuentes et al. (2018), Romanuke (2023), and Jaradat et al. (2019). This algorithm efficiently tackles the task of dividing a set of points into k subsets. A key limitation is the necessity to pre-specify the number of clusters. Jahwar & Abdulazeez (2020) provides a review of various k-means variations.

The affinity propagation clustering algorithm, used for dividing problems into subproblems as discussed in Ashour et al. (2018) and El-Samak & Ashour (2015), differs from k-means in that it does not require specifying the number of clusters upfront.

A multitude of heuristic algorithms are available for solving subproblems. Recent popular ones include genetic algorithms El-Samak & Ashour (2015), Romanuke (2023), ant colony algorithms Liao & Liu (2018), local search algorithms Anaya Fuentes et al. (2018), and the firefly algorithm Jaradat et al. (2019). The use of various algorithms at the subproblem solving stage is a primary means of researching and enhancing the efficiency of the decomposition approach.

2.2 Combining algorithms

The least attention is paid to algorithms for combining solutions to subproblems. All algorithms are heuristic. One of the criteria for the optimality of combining algorithms is the absence of self-intersection of the final solution cycle Quintas & Supnick (1965).

In the work Liao & Liu (2018), a hierarchical structure of the initial problem points is built. At the top level, clusters are considered as vertices, and the traveling salesman problem is solved. The result is a scheme for combining clusters. Two neighboring clusters are combined by finding the nearest vertices.

In the work Anaya Fuentes et al. (2018), the algorithm for creating the union is based on both the distance between cluster centroids and the distance between classes. This algorithm will be discussed in more detail in the following sections.

In Jaradat et al. (2019) and Khachaturov et al. (2000), algorithms for pairwise clustering were proposed. This approach has the advantage of ensuring at each stage of the solution that the solution forms a cycle. Additionally, this approach can detect the absence of new edge intersections in the cycle at each step of the algorithm.

In this paper, we propose a modification to the algorithm for combining the solutions of subproblems in a pairwise manner.

3 Problem Description and Algorithms

We consider the Euclidean Traveling Salesman Problem. Given coordinates (x_i, y_i) for N cities on a plane, the objective is to construct a tour that visits each city exactly once. The problem will be solved in three stages as follows. First, we partition the original set of cities into a certain number of clusters.

Next, we solve the TSP for each individual cluster using an open-source solver, obtaining a partial solution for each cluster. After this, we need to merge the clusters, with the goal of combining the partial solutions from each cluster into a solution for the original TSP. Two strategies for selecting clusters for merging have been formulated, with detailed descriptions provided in Appendix A.

Based on the chosen strategy, we select a pair of clusters to merge. The merging process is carried out using specific algorithms, which are described below. This procedure of selecting two clusters and merging them is repeated until only one cluster remains, which constitutes the solution to the original TSP problem.

3.1 CLUSTERING

Clustering is a key step in solving the TSP as it reduces the problem size and enhances the efficiency of routing algorithms. We explore two clustering methods: *k-means* and *affinity propagation*.

The *k-means* algorithm includes:

1. Random initialization of k centroids.
2. Assignment of each city to the nearest centroid.
3. Update of centroids based on the mean of the assigned cities' coordinates.
4. Iteration until centroids stabilize.

For a set of cities $\{1, 2, \ldots, n\}$ and k centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$, the algorithm minimizes the following function:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{k} r_{ij} \|x_i - \mu_j\|^2, \tag{1}$$

where $r_{ij} = 1$ if city with coordinates x_i belongs to cluster j with coordinates μ_j, and 0 otherwise.

Affinity propagation does not require a predefined number of clusters. It uses message passing between cities to form clusters, where each city can be a potential cluster center (exemplar).

The algorithm works as follows:

1. Initialization of the similarity matrix S, where $S(i,k)$ indicates the similarity between city i and city k. A higher value of $S(i,k)$ implies that city i is more likely to choose city k as its exemplar.
2. Exchange of two types of messages between cities: responsibility $r(i,k)$ and availability $a(i,k)$.
3. Iterative update of responsibility and availability values until convergence.

The responsibility $r(i,k)$ quantifies how well-suited city k is to serve as the exemplar for city i (relative to other candidate exemplars for city i). It is updated using the following formula:

$$r(i,k) \leftarrow S(i,k) - \max_{k' \neq k} a(i,k') + S(i,k'), \tag{2}$$

where $S(i,k)$ is the similarity between cities i and k, $a(i,k')$ is the availability of city k' to be the exemplar for city i. The availability $a(i,k)$ represents how appropriate it is for city i to choose city k as its exemplar, taking into account other cities' preferences. For cities that are not their own exemplars ($i \neq k$), the availability is updated as follows:

$$a(i,k) \leftarrow \min\left(0, r(k,k) + \sum_{i' \notin i,k} \max(0, r(i',k))\right) \text{ for } i \neq k, \tag{3}$$

where $r(k,k)$ is the self-responsibility of city k, $\sum_{i' \notin i,k} \max(0, r(i',k))$ is the sum of the positive responsibilities of other cities for city k. For cities that consider themselves as their exemplars ($i = k$), the availability is updated as:

$$a(k,k) \leftarrow \sum_{i' \neq k} \max(0, r(i', k)), \tag{4}$$

which accumulates the positive responsibilities of all other cities i' for city k.

By iteratively updating these messages, the algorithm eventually converges, and each city is assigned to the exemplar that maximizes the combined responsibility and availability.

The specific clustering method is not critical to our main goal, which is to analyze cluster merging algorithms post-clustering. Both *k-means* and *affinity propagation* are effective for clustering cities in the TSP, providing suitable results for further analysis.

3.2 Solving TSP within Clusters

After clustering, the original TSP is decomposed into multiple smaller TSPs within each cluster. We solve each of these subproblems separately using Google's 'or-tools' library. The computational effort for solving each subproblem is controlled via parameters such as maximum search time or the number of heuristic iterations.

Since each cluster contains significantly fewer cities than the original problem, solving the TSP for each cluster individually is computationally less intensive. This approach allows us to find solutions that are closer to optimal for each cluster compared to attempting to solve the entire problem at once. Smaller problem sizes reduce the solution space, enabling the solver to explore possibilities more thoroughly within a reasonable time frame. Once we have obtained the routes for each cluster, we proceed to merge them into a single, unified tour. The merging process is guided by specific algorithms designed to combine the partial solutions effectively.

3.3 Merging Clusters

At this stage, we have already partitioned the original TSP problem into clusters and obtained a solution for each individual cluster. The next step is to merge these cluster-level solutions into a single, unified tour that solves the original TSP. The key challenge at this point is determining the order in which the clusters should be merged, as the merging procedure works on two clusters at a time.

To address this, we propose two distinct strategies for selecting which clusters to merge at each step. These strategies guide the merging process, and their application can lead to different results, particularly when dealing with four or more clusters. The detailed descriptions of these strategies can be found in Appendix A.

The first strategy selects the nearest cluster to the newly formed cluster from the previous merging step. In this approach, we progressively expand the solution outward from a single starting point, as each newly merged cluster is joined with its closest neighbor. This method allows for a more localized growth of the solution, potentially reducing the overall distance traveled as the tour expands incrementally from a central area.

The second strategy, on the other hand, simply selects the two clusters that are closest to each other at each step, regardless of whether they were part of a previous merge or not. This results in a more distributed merging process, where clusters from different regions are combined at different stages. The solution in this case is constructed by connecting the closest clusters across the entire problem space, leading to a more global approach.

As a result, the first strategy can be seen as gradually expanding the solution from one area, while the second strategy simultaneously incorporates various regions. These differing approaches lead to variations in the accuracy of the final solution, which will be evident in the results section.

We compare three different cluster merging procedures, one of which is our own modification based on Sigal's algorithm. This modification was developed to address the lack of detailed merging procedures in the original work, improving its applicability to our problem.

3.3.1 Modified Sigal's Algorithm

This algorithm iteratively merges clusters by selecting edges for removal and reconnecting clusters without introducing crossing edges. The merging process continues until all clusters are combined into a single cycle.

Given k cycles $C_1, C_2, \dots, C_k$, the algorithm repeatedly selects two neighboring clusters to merge. For two cycles C_p and C_q, where:

$$C_p = (i_1^p, \dots, i_{r_p}^p), \quad C_q = (i_1^q, \dots, i_{r_q}^q), \tag{5}$$

we aim to merge them by identifying the best pair of edges from each cycle to replace. Specifically, we select edges (i_s^p, i_{s+1}^p) from cycle C_p and (i_t^q, i_{t+1}^q) from cycle C_q. The goal is to minimize the following expression:

$$\min \left(d(i_t^q, i_s^p) + d(i_{t+1}^q, i_{s+1}^p) - d(i_t^q, i_{t+1}^q) - d(i_s^p, i_{s+1}^p) \right), \tag{6}$$

where $d(i, j)$ is the distance between cities i and j.

Once the optimal edges are found, the edges (i_s^p, i_{s+1}^p) and (i_t^q, i_{t+1}^q) are removed from the respective cycles, and new edges (i_s^p, i_t^q) and (i_{s+1}^p, i_{t+1}^q) are added, effectively merging the two cycles into a new cycle C_p'.

This process of merging two cycles is repeated until all clusters are combined into one final cycle:

$$C_{final} = C_1' \cup \ldots \cup C_k', \tag{7}$$

where C_{final} represents the final solution, encompassing all cities from the original problem.

The complexity of this algorithm is $O(k \cdot (\frac{n}{k})^2) = O(\frac{n^2}{k})$ where n is the number of cities and k is the number of clusters, since we have to perform the merging procedure with k clusters, each step reducing their number by one and at the same time to search all pairs of edges from different clusters, and they are on average $\frac{n}{k}$.

3.3.2 Mid-Edge Merging Algorithm

This algorithm merges clusters by finding the shortest distance between the midpoints of edges from different clusters. Instead of directly comparing city-to-city distances, it considers the midpoints of edges to guide the merging process.

Given two cycles C_p and C_q, we first calculate the midpoints of each edge:

$$m_s^p = \frac{i_s^p + i_{s+1}^p}{2}, \quad m_t^q = \frac{i_t^q + i_{t+1}^q}{2}, \tag{8}$$

where m_s^p is the midpoint of the edge (i_s^p, i_{s+1}^p) in cycle C_p and m_t^q is the midpoint of the edge (i_t^q, i_{t+1}^q) in cycle C_q.

The algorithm identifies the pair of edges (one from each cluster) whose midpoints are closest by minimizing the distance between midpoints:

$$\min d(m_s^p, m_t^q), \tag{9}$$

where $d(m_s^p, m_t^q)$ is the distance between the midpoints.

Once the nearest midpoints are found, the corresponding edges (i_s^p, i_{s+1}^p) and (i_t^q, i_{t+1}^q) are removed, and new edges (i_s^p, i_t^q) and (i_{s+1}^p, i_{t+1}^q) are added. This effectively merges the two clusters into a new cycle.

The process is repeated until all clusters are merged into a single cycle, representing the final solution.

The complexity of this algorithm is the same as that of the modified Sigal algorithm.

3.3.3 Centroid-Based Merging Algorithm

This algorithm focuses on merging clusters based on the distances between their centroids and the nearest cities within each cluster. A centroid represents the geometric center of a cluster and provides a way to approximate the cluster's location on the plane.

To merge clusters, the algorithm first computes the centroid of each cluster. For a cluster C_i, its centroid is calculated as:

$$\text{centroid}(C_i) = \left(\frac{\sum_{j=1}^{r_i} x_j}{r_i}, \frac{\sum_{j=1}^{r_i} y_j}{r_i} \right),$$

where r_i is the number of cities in cluster C_i, and (x_j, y_j) are the coordinates of city j in the cluster.

The algorithm begins by selecting the first cluster, C_1, and finding the nearest neighboring cluster, C_c, based on the distance between their centroids:

$$C_c = \arg\min d(\text{centroid}(C_1), \text{centroid}(C_i)), \tag{10}$$

where $i \in \{2, \ldots, k\}$ and $d(\cdot, \cdot)$ represents the distance between centroids.

Once the nearest cluster C_c is identified, the algorithm finds the nearest pair of cities between clusters C_1 and C_c, denoted as $nodeC1$ (from C_1) and $nodeC2$ (from C_c). These cities are then connected by a new edge, and the clusters are merged by adjusting the edges between them to form a new combined cycle.

The process is repeated with the newly formed cluster, which continues to grow by merging with the nearest remaining cluster until only one cycle remains, encompassing all cities.

The complexity of this algorithm is $O(n + \frac{n^2}{k})$ since we also need to compute the centroids of the clusters.

4 Numerical Experiments

The following instances from the TSPLIB dataset were selected for the numerical experiments: berlin52.tsp, bier127.tsp, ch130.tsp, ch150.tsp, eil51.tsp, eil76.tsp, eil101.tsp, pr76.tsp, pr144.tsp, pr439.tsp, and rat575.tsp. They vary in the number of vertices, indicated by their names, density, topology, and complexity of finding optimal solutions. Some instances, such as those starting with 'ch', are based on real-world geographical maps.

The following numerical experiments were conducted to test the effectiveness of cycle merging algorithms:

1. Clustering instances using either k-means or affinity propagation. For k-means, the number of clusters is manually set and ranges from 4 to 10. For affinity propagation, the algorithm itself determines the required number of clusters.
2. Solving the traveling salesman problem for each cluster using OR-Tools, with time limits set to 1 second and 10 seconds.
3. Merging cycles of solutions from the clustered traveling salesman problems using three different merging algorithms. Strategies 1 and 2 are employed to select the next cluster for merging, and various initial clusters are considered for centroid connections.

4.1 Comparison of Combining Algorithms

Three different methods of combining cycles were compared: ***connection_centroids*** (connecting two clusters via the nearest centroids); ***connection_midpoints*** (connecting two clusters at the nearest midpoints on their edges); ***connection_sigal*** (connecting two clusters according to the modified Sigal method).

For all the different combinations of clustering methods, merging strategies, and the number of clusters, we constructed boxplots to showcase the final relative distances for each merging technique, as depicted in Figure 1.

$$\text{Relative Distance} = \frac{\text{Total Distance}}{\text{Accumulated Distance}}, \tag{11}$$

where the *Accumulated Distance* is the sum of the solutions for all clusters.

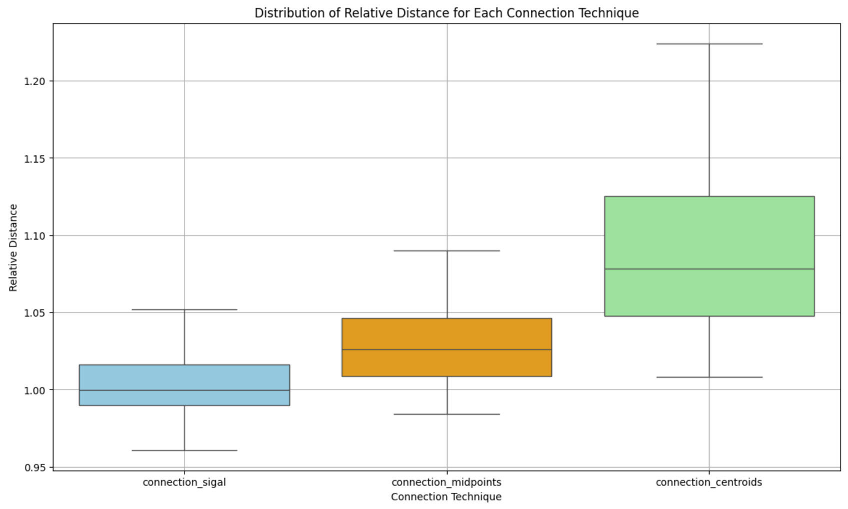

Figure 1: Variation in solution lengths depending on the merging method employed.

4.2 Investigation of the Impact of the Order of Cluster Solution Connections

This study explores how the final solution is influenced by the order of cluster connections. For the *connection_centroids* algorithm, the sequence is determined starting with the selection of the first cluster to merge. For other algorithms, either the first or second strategy is employed to identify the next pair of clusters to merge.

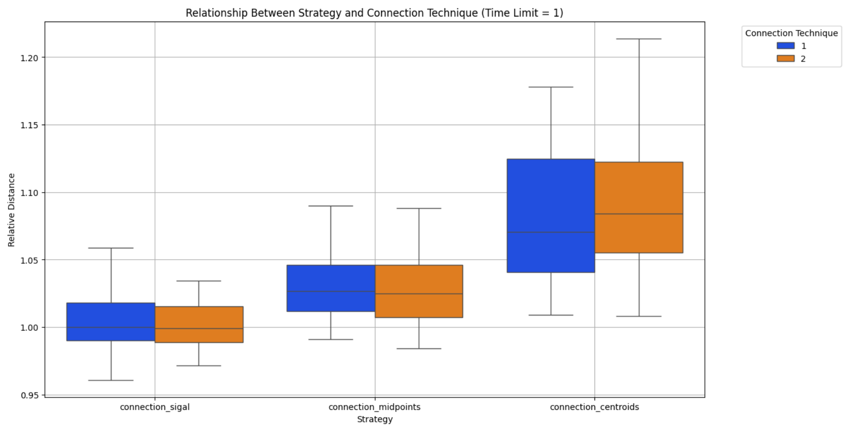

Figure 2: The relative length of the solutions depends on the merging method and the cluster connection order strategy (Time Limit = 1).

Figure 2 illustrates the distribution of relative solution lengths based on the merging algorithm and the order of connection.

To assess how the solution quality during the subproblem solving step affects outcomes, a similar distribution is plotted for solutions with a time limit of 10 seconds, as shown in Figure 3.

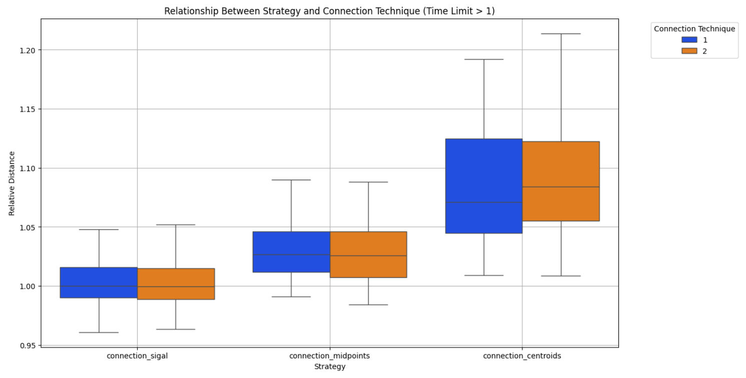

Figure 3: The relative length of the solutions depends on the merging method and the cluster connection order strategy (Time Limit = 10).

4.3 Comparison of the Effectiveness of the Solution Based on the Number of Clusters

This experiment investigates the relationship between merging techniques and the number of clusters used in k-means clustering. We examine how the cycle length, necessary to solve the problem, varies with the number of clusters into which the original set of vertices is divided. This analysis includes each merging method employed.

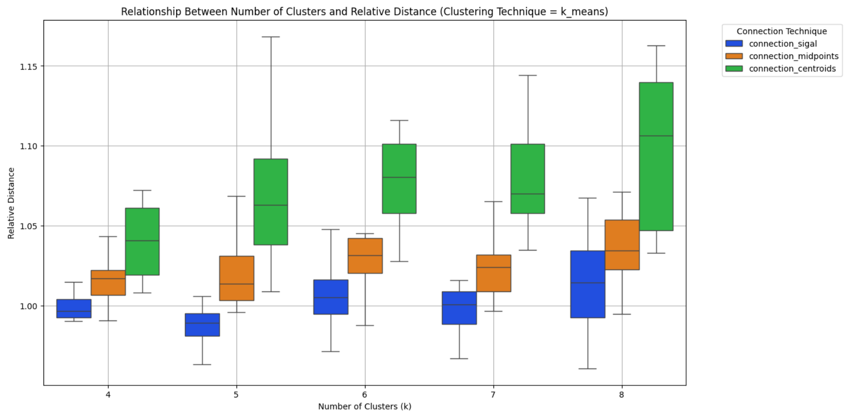

Figure 4: The relationship between the cycle length and the number of clusters, illustrates how segmentation affects solution paths across different merging techniques.

4.4 Comparison of the Impact of Errors in Cycle Construction on Clusters and Merging Results

This section presents an investigation into how the quality of solutions for subproblems affects the performance of cycle merging algorithms.

We measured the relative change in the total length of the solutions for the cycles and the variation in the relative length of the final solution upon modifying the Time Limit parameter. The results are depicted in Figure 5, illustrating the dependency for each merging algorithm.

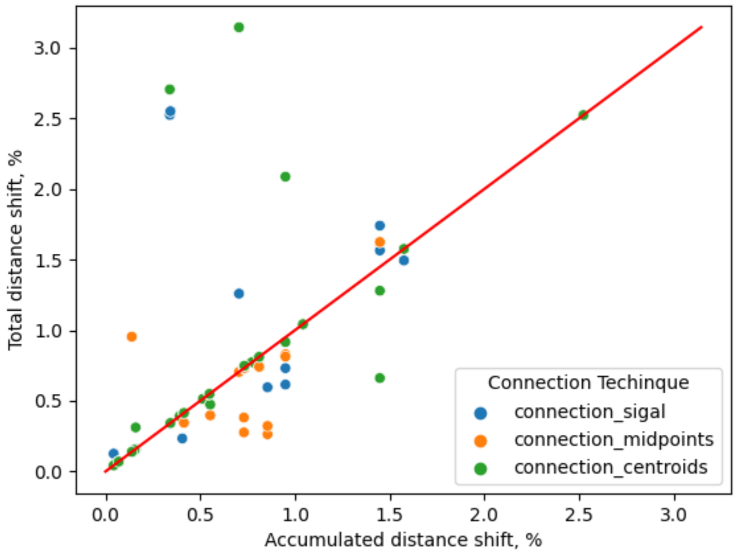

Figure 5: Impact of variations in time limits on the relative changes in the lengths of subproblem solutions and their influence on the overall merging outcome.

5 Conclusion

In this paper, we investigated a decomposition approach to solving the real-world ETSP. This approach involves dividing the problem into smaller subproblems, solving them independently, and then merging the solutions to obtain an optimal solution for the original problem. We proposed a new algorithm for merging the subsolutions based on the Segal algorithm. An experiment 4.1 comparing different merging algorithms showed that:

- modified Sigal merging algorithm has the lowest median of the final distance and a moderate range of values;
- The median value of the connection centroids is higher than that of other methods, and the range of values for the final distance is also wide;
- The connection midpoints produce a result that lies between the two methods and a range of relative final distance values that can be considered moderate.

The same results were obtained from the algorithms with different numbers of clusters in the experiment 4.3.

Based on the results, it can be concluded that the modified Sigal merging algorithm has an overall advantage over other methods of merging that were considered.

Experiment 4.2 demonstrated that all algorithms are dependent on the order in which the union is performed. However, as the accuracy of the subproblem solution improves, the difference between the solutions obtained from different orders becomes less significant.

The connection between the error and the stage of solving subproblems in the process of combining solutions has also been investigated in experiment 4.4. Improving the quality of a subproblem solution usually leads to a linear improvement in the final solution. The degree of this improvement is rarely influenced by the merging algorithm used.

The results obtained will allow further in-depth study of the decomposition approach to solving the TSP problem and will lead to its improvement.

Acknowledgments

The research was supported by RSF (project No. 22-71-10131)

References

Gustavo Erick Anaya Fuentes, Eva Selene Hernández Gress, Juan Carlos Seck Tuoh Mora, and Joselito Medina Marín. Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic. *PLoS One*, 13(8):e0201868, 2018.

Antonios Antoniadis, Krzysztof Fleszar, Ruben Hoeksma, and Kevin Schewior. A ptas for euclidean tsp with hyperplane neighborhoods. *ACM Transactions on Algorithms (TALG)*, 16(3):1–16, 2020.

Wesam Ashour, Riham Muqat, and Haneen Al-Talli. Optimization of traveling salesman problem based on adaptive affinity propagation and ant colony algorithms. *Int. J. Comput. Appl*, 181: 25–31, 2018.

Vladimir Deineko, Bettina Klinz, and Mengke Wang. 2-period balanced travelling salesman problem: a polynomially solvable case and heuristics. *arXiv preprint arXiv:2203.06090*, 2022.

Ahmad Fouad El-Samak and Wesam Ashour. Optimization of traveling salesman problem using affinity propagation clustering and genetic algorithm. *Journal of Artificial Intelligence and Soft Computing Research*, 5(4):239–245, 2015.

Alan Fuad Jahwar and Adnan Mohsin Abdulazeez. Meta-heuristic algorithms for k-means clustering: A review. *PalArch's Journal of Archaeology of Egypt/Egyptology*, 17(7):12002–12020, 2020.

Ameera Jaradat, Waed Diabat, et al. Solving traveling salesman problem using firefly algorithm and k-means clustering. In *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, pp. 586–589. IEEE, 2019.

BP Khachaturov, VE Veselovsky, A V Slots, S U Kaldybaev, YEZH Kaliev, A G Kovalenko, V M Montlevich, IK Segal, and RV Khachaturov. Combinatorial methods and algorithms for solving high-dimensional discrete optimization problems, 2000.

Erchong Liao and Changan Liu. A hierarchical algorithm based on density peaks clustering and ant colony optimization for traveling salesman problem. *Ieee Access*, 6:38921–38933, 2018.

Louis V Quintas and Fred Supnick. On some properties of shortest hamiltonian circuits. *The American Mathematical Monthly*, 72(9):977–980, 1965.

Vadim Romanuke. Traveling salesman problem parallelization by solving clustered subproblems. *Foundations of Computing and Decision Sciences*, 48(4):453–481, 2023.

Sarah K Schaumann, Abhishake Kundu, Juan C Pina-Pardo, Matthias Winkenbach, Ricardo A Gatica, Stephan M Wagner, and Timothy I Matis. The flying sidekick traveling salesman problem with multiple drops: A simple and effective heuristic approach. *arXiv preprint arXiv:2403.18091*, 2024.

Yannis Spyridis, Athanasios Gkelias, and Vasileios Argyriou. Variational quantum approach for the multiple traveling salesman problem optimisation. In *2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pp. 354–358. IEEE, 2023.

Chien-Yuan Tsai and Chuang-Cheng Chiu. A vns based hierarchical clustering method. In *International Conference on Computational Intelligence*. Citeseer, 2006.

A Appendix

A.1 Strategy 1: Sequential Merging with Newly Formed Cluster

The first strategy involves sequentially merging clusters with the newly formed cluster. The main steps are:

1. **Find the Closest Clusters**: Identify the two nearest clusters based on the distance between their centroids:
$$C_a, C_b = \arg \min_{C_i, C_j \in \{C_1, C_2, \ldots, C_k\}, i \neq j} d(\text{centroid}(C_i), \text{centroid}(C_j)), \tag{12}$$
where $d(\text{centroid}(C_i), \text{centroid}(C_j))$ is the distance between the centroids of clusters C_i and C_j.
2. **Merge Clusters**: Merge clusters C_a and C_b and recalculate the centroid of the newly formed cluster C_{new}:
$$\text{centroid}(C_{\text{new}}) = \frac{1}{|C_a| + |C_b|} \sum_{i \in C_a \cup C_b} i, \tag{13}$$
where $|C_a|$ and $|C_b|$ are the numbers of vertices in clusters C_a and C_b respectively.
3. **Find and Merge with the Next Nearest Cluster**: Identify the nearest cluster to C_{new} and merge it:
$$C_{\text{next}} = \arg \min_{C_i \in \{C_1, C_2, \ldots, C_k\} \setminus \{C_{\text{new}}\}} d(\text{centroid}(C_{\text{new}}), \text{centroid}(C_i)), \tag{14}$$
4. **Repeat the Process**: Continue merging the newly formed cluster with the nearest cluster until only one cluster remains.

This strategy ensures sequential expansion by progressively merging the nearest clusters.

A.2 Strategy 2: Merging Closest Clusters

The second strategy involves merging the two closest clusters regardless of their location. The steps are:

1. **Find the Closest Clusters**: Identify the two nearest clusters:
$$C_a, C_b = \arg \min_{C_i, C_j \in \{C_1, C_2, \ldots, C_k\}, i \neq j} d(\text{centroid}(C_i), \text{centroid}(C_j)), \tag{15}$$
2. **Merge Clusters**: Merge clusters C_a and C_b and recalculate the centroid of the newly formed cluster C_{new}:
$$\text{centroid}(C_{\text{new}}) = \frac{1}{|C_a| + |C_b|} \sum_{i \in C_a \cup C_b} i, \tag{16}$$
3. **Find and Merge the Next Closest Clusters**: Identify the next pair of closest clusters, considering the recalculated centroids:
$$C_x, C_y = \arg \min_{C_i, C_j \in \{C_1, C_2, \ldots, C_k\} \setminus \{C_{\text{new}}\}, i \neq j} d(\text{centroid}(C_i), \text{centroid}(C_j)), \tag{17}$$
4. **Repeat the Process**: Continue merging the two closest clusters until only one cluster remains.

This strategy allows merging to occur in different areas, potentially facilitating parallel operations across different regions.