

Effective Solution for the Traveling Salesman Problem on Randomly Distributed Data

Boris F. Melnikov Pavel A. Oganessian Bowen Liu

Department of Computational Mathematics and Cybernetics, MSU-BIT University, Shenzhen, China
Institute of Mathematics, Mechanics and Computer Science named after I.I. Vorovich, Southern Federal University, Rostov-on-Don, Russia

July 15, 2025

Traveling Salesman Problem: Statement

The Traveling Salesman Problem (TSP) is a classic optimization problem in computer science and mathematics.

Formal Definition

Given a set of N cities and the distances between each pair of cities, find the shortest possible route that visits each city exactly once and returns to the starting city.

Key characteristics:

- The route must visit every city exactly once.
- The total distance (or cost) of the route should be minimized.
- The problem is NP-hard; exact solutions are computationally expensive for large N .

Well-Known Versions of the TSP

Variants of the Traveling Salesman Problem:

- **Symmetric TSP**
Distance from city A to B is the same as from B to A.
- **Asymmetric TSP**
Distances between cities may differ depending on direction.
- **Metric TSP**
Distances satisfy the triangle inequality.
- **Euclidean TSP**
Cities are points in the plane; distances are Euclidean.
- **Multiple TSP (mTSP)**
Multiple salesmen, each with their own route.
- **Time-Dependent TSP**
Travel costs depend on time of day or other factors.
- **Prize-Collecting TSP**
Not all cities must be visited; each has a prize or penalty.

Applications: Genome Sequencing and Random TSP

The Traveling Salesman Problem is widely used in computational biology, especially in genome sequencing and assembly.

Genome Sequencing:

- TSP models the problem of finding the shortest superstring that contains all DNA fragments, helping reconstruct the original genome.
- Each fragment is treated as a "city," and the cost between fragments reflects their overlap or similarity.
- The optimal tour corresponds to the best order for assembling the genome.

Many biological datasets are noisy or lack strong structure, so random TSP instances are highly relevant for testing algorithms. Finding exact solutions on random data helps researchers benchmark algorithms and understand their true performance.

Random Non-Symmetric TSP: Problem Setup

Given:

- A set of n cities labeled $1, 2, \dots, n$.
- A distance (cost) matrix $D = [d_{ij}]$, where d_{ij} is the cost to travel from city i to city j .
- The matrix D is **non-symmetric**: $d_{ij} \neq d_{ji}$ in general.
- All d_{ij} are generated randomly.

Random Non-Symmetric TSP: Mathematical Formulation

Objective:

Find a permutation π of the cities (a tour) that minimizes the total travel cost:

$$\min_{\pi \in S_n} \left(\sum_{k=1}^n d_{\pi_k, \pi_{k+1}} \right)$$

where $\pi_{n+1} = \pi_1$ (the tour returns to the starting city).

Constraints:

- Each city is visited exactly once.
- The tour starts and ends at the same city.

The Traveling Salesman Problem (TSP) as a Hamiltonian Cycle:

In graph theory, the TSP can be formulated as finding a Hamiltonian cycle of minimal total weight in a weighted directed graph.

- Each city corresponds to a vertex in the graph.
- Each possible route between cities is represented by a directed edge with weight d_{ij} .
- A **Hamiltonian cycle** is a cycle that visits every vertex exactly once and returns to the starting vertex.
- The objective is to find the Hamiltonian cycle with the smallest possible total weight.

Greedy Algorithms for TSP: Main Principles

The main idea is to build the route step by step, always choosing what **looks best at the moment**.

Typically, you start from any city. At each step, you look at all cities you haven't visited yet and pick the one that's closest (or cheapest to reach). You add that city to your route and repeat the process. This continues until every city has been visited, and finally, you return to where you started.

Greedy algorithms are fast and easy to implement. They often give decent results, especially for small problems or when the cities are spread out in a regular way. However, they don't guarantee the best possible route and can sometimes miss much shorter solutions.

Popular Greedy Strategies

There are several well-known greedy approaches for the TSP:

Nearest Neighbor: Always go to the nearest unvisited city.

Cheapest Insertion: At each step, insert the city that adds the least extra distance to your route.

Farthest Insertion: Sometimes, it's better to add the city that's farthest away, to avoid leaving tough connections for last.

Each of these strategies has its own strengths and weaknesses, and the quality of the solution can vary depending on the specific arrangement of cities.

Branch and Bound for TSP

The main idea is to systematically explore all possible tours, but to avoid unnecessary work by "pruning" routes that cannot possibly lead to the best solution.

- The algorithm builds a search tree, where each node represents a partial tour.
- At each step, it calculates a lower bound on the cost of completing the tour from the current node.
- If this lower bound is greater than the cost of the best solution found so far, the branch is abandoned ("pruned").
- Otherwise, the algorithm continues to expand the branch, exploring further possible cities.
- The process continues until all promising branches have been explored.

Implementation: Efficient Search Tree

Each node in the tree represents a partial tour, storing only the current path, the set of visited cities, and the accumulated cost so far:

$$\text{Node} = (\text{path}, \text{cost})$$

where path is the sequence of cities visited, cost is the sum of d_{ij} for the current partial tour.

Child nodes are generated only for feasible extensions, i.e., for each unvisited city j , a new node is created by appending j to the path and updating the cost:

$$\text{cost}_{\text{new}} = \text{cost}_{\text{current}} + d_{i,j}$$

where i is the last city in the current path.

Bounding Techniques

To make Branch and Bound practical for larger TSP instances, we use following bounding strategies.

- Lower bounds are computed using cost matrix reduction and other heuristics.
- The implementation leverages precomputed shortest paths and triangle inequality to tighten bounds.

This approach allows the algorithm to skip huge portions of the search space, focusing only on promising solutions.

For our experiments, distance matrices for the TSP were generated using pseudo-random number generators.

- Matrices of size 49×49 and 99×99 were created.
- Distances were chosen uniformly from 0 to 999.
- Diagonal elements were set to a very large value to prevent self-loops.

Randomness Check

To ensure the data is truly random and uniformly distributed, a chi-squared goodness-of-fit test was performed.

- The expected frequency for each possible value is $E = \frac{N}{1000}$, where N is the total number of generated values.
- The test statistic is calculated as:

$$\chi^2 = \sum_{i=0}^{999} \frac{(O_i - E)^2}{E}$$

where O_i is the observed frequency of value i .

- The resulting p -value was compared to a threshold (0.05) to confirm uniformity.

This validation ensures that the generated data does not introduce bias into the algorithm's performance evaluation.

Numerical Results: Control Parameters

The following control parameters were used to tune the algorithm and manage computational resources:

Parameter Name	Description	Reference Value
Max problem dimension	Used to effectively pre-allocate memory for matrices	99
Max subproblem queue size	Upper limit for stored partial solutions during branching	10,000
Max branching iterations	Cap on steps to prevent infinite loops	100,000
Full brute-force threshold	Subproblems with less nodes use exhaustive search	2
Priority threshold	Subproblems with nodes jump the queue	6

Control parameters for the Branch and Bound algorithm

Numerical Results: Algorithm Comparison

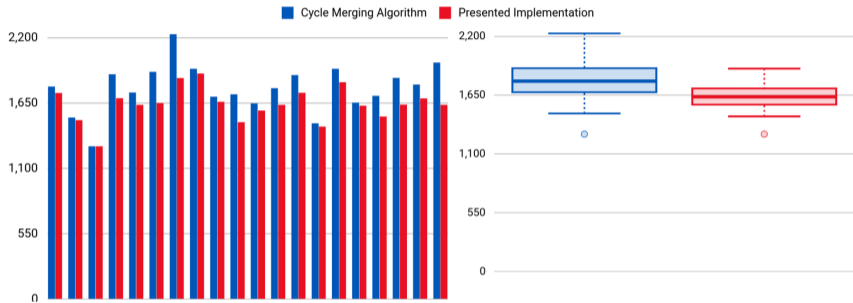











Figure: Comparison of the cost of the shortest obtained Hamiltonian cycle for both algorithms on 20 random problems with 99 nodes (left) and average, maximum, and minimum cost (right).

References (1/2)

-  Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.
-  Gutin, G., & Punnen, A. P. (Eds.) (2006). *The Traveling Salesman Problem and Its Variations*. Springer.
-  Cook, W. J. (2012). *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press.
-  Gimadi, E. & Tsidulko, Oxana. (2020). Asymptotically Optimal Algorithms for the Prize-Collecting Traveling Salesman Problem on Random Inputs. *10.1007/978-3-030-38629-0_16*.
-  Huang, Zhouchun & Zheng, Qipeng & Pasiliao, Eduardo & Boginski, Vladimir & Zhang, Tao. (2019). A cutting plane method for risk-constrained traveling salesman problem with random arc costs. *Journal of Global Optimization*, 74, 839-859. *10.1007/s10898-018-0708-0*.

References (2/2)

-  Melnikov B., Chaikovskii D. Pseudogeometric Version of the Traveling Salesman Problem, Its Application in Quantum Physics Models and Some Heuristic Algorithms for Its Solution (2024). *Springer Proceedings in Mathematics and Statistics*, 446, pp. 391-401.
-  Melnikov, B., Radionov, A., Moseev, A., Melnikova, E. Some specific heuristics for situation clustering problems. *ICSOFIT 2006 1st International Conference on Software and Data Technologies Proceedings*, 2006, 2, pp. 272-279.
-  Melnikov, B.F., Melnikova, E.A., Pivneva, S.V., Dudnikov, V.A., Davydova, E.V. Geometric and game approaches for some discrete optimization problems. *Ceur Workshop Proceedings*, 2018, 2212, pp. 312-321.
-  Leonova Yu. Parallel implementation of the cycle merging algorithm for solving the traveling salesman problem. <https://github.com/YuliyaLeonova/CycleMergingAlgorithm.git> (accessed: 10.05.2025).

Thank You for Your Attention!