TSP Approximation Algorithm

Greedy Nearest-Neighbor Heuristic

1. Algorithm Strategy

The Traveling Salesman Problem (TSP) is NP-hard: finding the optimal tour of n cities requires checking up to (n-1)!/2 possible tours in the worst case. For practical instances, we use approximation algorithms that run in polynomial time and provide reasonably good solutions.

Greedy Nearest-Neighbor Heuristic: This is an anytime algorithm that constructs a tour incrementally by always moving to the nearest unvisited node from the current position.

Algorithm Pseudocode:

```
function GreedyNearestNeighbor(graph, start_node):
    visited ← {start_node}
    tour ← [start_node]
    current ← start_node

    while |visited| < n:
        unvisited ← all nodes - visited
        nearest ← argmin {distance(current, node) for node in
unvisited}
        tour.append(nearest)
        visited.add(nearest)
        current ← nearest

    tour.append(start_node)   // return to start
    return tour
```

2. Algorithm Properties

Type: Greedy, Anytime, Deterministic

Start Node: Algorithm starts from an arbitrary node (typically first in input order).

Optimality: Does NOT guarantee optimal tour. Often produces tours 25-30% worse than optimal.

Completeness: Always produces a valid complete tour (if graph is connected).

Determinism: Given the same graph and start node, always produces the same tour.

3. Runtime Complexity Analysis

Time Complexity: $O(n^2)$

Detailed breakdown for a graph with n nodes and m edges:

1. Main Loop: Executes n-1 times (once per node after start)

Each iteration: Find nearest unvisited node

2. Per Iteration Cost: O(n)

Check all remaining unvisited nodes: O(n)

For each unvisited node, find minimum edge weight: O(degree)

In worst case: O(m/n) average per node, O(n) worst case (complete graph)

3. Total Cost: $(n-1) \times O(n) = O(n^2)$

Space Complexity: O(n)

- Store visited set: O(n)
- Store adjacency list: O(n + m)
- Store tour: O(n)

4. Polynomial Time Guarantee

The algorithm runs in polynomial time $O(n^2)$, which is essential for TSP approximation:

Why Polynomial Time Matters:

- NP-hard problems like TSP have no known polynomial-time exact solutions (unless P=NP)
- Approximation algorithms must run in polynomial time to be practical
- $O(n^2)$ is very efficient: even for n=10,000 nodes, we compute ≈ 100 million operations
- Compare to brute force: $(n-1)!/2 \approx 10^{30,000}$ operations for n=10,000

Practical Performance:

- For n=100: ~10,000 operations (microseconds)
- For n=1,000: ~1,000,000 operations (milliseconds)
- For n=10,000: ~100,000,000 operations (seconds)
- For n=100,000: ~10,000,000,000 operations (seconds to minutes, depending on hardware)

5. Approximation Quality & Bounds

Approximation Ratio: The ratio of greedy tour cost to optimal tour cost is unbounded in general (e.g., star graph: greedy may be 2× optimal). However, for Euclidean TSP, empirical performance is good.

Known Results:

- Greedy is O(log n)-approximation for metric TSP (satisfies triangle inequality)
- For general graphs: no constant-factor approximation exists (unless P=NP)
- Empirical: typically 1.2–1.5× optimal on random Euclidean instances

When Greedy Fails:

The greedy heuristic makes locally optimal choices that can lead to globally suboptimal tours:

Example (Star Graph):

- Hub A connected to spokes B, C, D (cost 1 each)
- Outer edges B-C, C-D, B-D all cost 100
- Greedy: A→B→C→D→A = 1+100+100+1 = 202
- Optimal: Same (unavoidable in star; greedy happens to tie here)

Better improvement: Use 2-opt local search or other meta-heuristics after greedy construction.

6. Comparison to Other Approximation Methods

Method	Time Complexity	Approximation Ratio	Notes
Nearest Neighbor	$O(n^2)$	Unbounded	Simple, fast, practical
Christofides	$O(n^3)$	1.5×	Complex, better quality
2-opt Improvement	$O(n^2)$ per swap	Variable	Local search refinement
Genetic Algorithm	$O(n^2)$ per generation	Unbounded	Metaheuristic, often good
Branch & Bound	$O(2^n)$ worst	Optimal	Exact but exponential

7. Conclusion

The greedy nearest-neighbor heuristic is a practical, polynomial-time approximation algorithm for TSP:

- **Efficiency:** $O(n^2)$ runtime makes it applicable to instances with hundreds of thousands of nodes
- **Simplicity:** Easy to implement and understand
- **Anytime Algorithm:** Returns a valid tour at any point of execution (though typically runs to completion)
- **Quality Trade-off:** Sacrifices optimality for speed; good for many practical applications

For applications requiring better solutions, combine greedy with local search (2-opt, 3-opt) or use more sophisticated algorithms (Christofides, LKH, genetic algorithms).