Technical Report: Travelling Salesman Problem Analysis

Comparative Analysis of Hill Climbing and Genetic Algorithms

Author: Hassan Osman

Table of Contents

1. Introduction

This technical report documents the implementation and comparative analysis of two optimization algorithms for solving the Travelling Salesman Problem (TSP): Hill Climbing with Random Restarts and Genetic Algorithm (GA). The study evaluates their performance on Euclidean TSP instances ranging from 10 to 50 cities.

1.1 Problem Definition

The Travelling Salesman Problem (TSP) is a classic NP-hard combinatorial optimization problem where the objective is to find the shortest possible route that visits each city exactly once and returns to the origin city.

1.2 Objectives

- Implement Hill Climbing with Random Restarts algorithm
- Implement Genetic Algorithm for TSP
- Compare solution quality and computational efficiency
- Analyze scalability with increasing problem size
- Provide practical recommendations for TSP optimization

2. Implementation Details

2.1 Dataset Specifications

Feature	Description	Value
Dataset	50 cities coordinates	cities.csv
Distance Metric	Distance calculation method	Euclidean distance
Coordinates	City locations	(x, y) pairs
Total Cities	Complete dataset size	50 cities

2.2 Hill Climbing with Random Restarts

Algorithm Components

- **Initial Solution:** Random permutation of city indices
- **Neighborhood:** All possible pair swaps ($O(n^2)$)
- **Search Strategy:** Steepest ascent hill climbing
- **Restart Mechanism:** 40 random restarts

- **Termination:** No improvement for 1000 iterations

```
def hill_climb_with_restarts(coords, restarts=40): best_route = None
best_distance = float('inf') histories = [] for _ in range(restarts):
start = random_route(len(coords)) route, distance, history =
hill_climb(start, coords) histories.append(history) if distance <
best_distance: best_distance = distance best_route = route return
best_route, best_distance, histories
```

2.3 Genetic Algorithm

Algorithm Parameters

Population Size

120

Generations

250

Crossover Rate

90%

Mutation Rate

3%

Genetic Operators

- **Order Crossover (OX):** Preserves ordering from parents
- **Swap Mutation:** Random position swapping
- **Tournament Selection:** k=3 competitive selection
- **Elitism:** Preserves best solution across generations

3. Experimental Setup

3.1 Test Scenarios

Problem Size	Trials per Algorithm	Description
10 cities	5 trials	Small-scale evaluation
20 cities	5 trials	Medium-scale testing
30 cities	5 trials	Increased complexity
40 cities	5 trials	Large problem testing
50 cities	5 trials	Full dataset evaluation

3.2 Performance Metrics

Solution Quality

Total Route Distance

Lower values indicate better solutions

Computational Time

Execution Seconds

Algorithm runtime efficiency

Consistency

Solution Variance

Reliability across multiple runs

3.3 Environment Specifications

- **Language:** Python 3.x
- **Libraries:** NumPy, Pandas, Matplotlib, Seaborn
- **Random Seed:** Fixed at 42 for reproducibility
- **Hardware:** Standard computing environment
- **Visualization:** Seaborn with whitegrid style

4. Results Analysis

4.1 Performance Summary by City Size

City Size	Algorithm	Average Distance	Average Time (s)	Performance Notes
10 Cities	Hill Climbing	0.00	0.044	Perfect solutions, 10x faster than GA
	Genetic Algorithm	0.00	0.411	Perfect solutions, slower execution

City Size	Algorithm	Average Distance	Average Time (s)	Performance Notes
20 Cities	Hill Climbing	0.00	0.779	Perfect solutions, maintains accuracy
	Genetic Algorithm	57.19	0.633	Variable performance (0-138 distance)
30 Cities	Hill Climbing	0.00	5.188	Perfect solutions, performance gap emerges
	Genetic Algorithm	793.78	1.087	Poor performance, significant gap
40 Cities	Hill Climbing	0.00	15.298	Perfect solutions despite complexity
	Genetic Algorithm	1265.82	1.133	Performance degradation continues
50 Cities	Hill Climbing	0.00	38.396	Perfect solutions, exponential time growth
	Genetic Algorithm	1737.29	1.463	~1750 units worse than HC solutions

4.2 Key Performance Findings

Solution Quality Winner

Hill Climbing

Perfect solutions across all problem sizes

Speed Winner (Small)

Hill Climbing

10-20 cities: Faster execution

Speed Winner (Large)

Genetic Algorithm

30-50 cities: Faster execution

Consistency

Hill Climbing

Perfect reliability vs GA variability

5. Algorithm Performance Characteristics

5.1 Hill Climbing with Random Restarts

Strengths:

- Excellent solution quality for Euclidean TSP
- Deterministic behavior with fixed random seed
- Simple implementation and easy debugging
- Perfect solution reliability across all tested sizes
- Effective restart strategy prevents local optima

Weaknesses:

- Exponential time complexity ($O(n^3)$ neighborhood evaluation)
- Computationally expensive for large problems

- Complete neighborhood evaluation required
- May not scale well beyond 50 cities
- Memory intensive for large neighborhood storage

5.2 Genetic Algorithm

Strengths:

- Constant time complexity relative to problem size
- Parallel exploration capability
- Robust to problem representation changes
- Better scalability potential
- Population-based approach prevents early convergence

Weaknesses:

- Poor solution quality on Euclidean TSP
- High parameter sensitivity
- Premature convergence issues
- Order crossover may not preserve good building blocks
- Requires careful parameter tuning

5.3 Complexity Analysis

Algorithm	Time Complexity	Space Complexity	Scalability
Hill Climbing	$O(\text{restarts} \times \text{iterations} \times n^2)$	$O(n^2)$	Poor (exponential growth)

Algorithm	Time Complexity	Space Complexity	Scalability
Genetic Algorithm	O(generations × population × n)	O(population × n)	Good (linear components)

6. Discussion

6.1 Why Hill Climbing Performs Better

- **Smooth Search Landscape:** Euclidean distances create a continuous optimization surface well-suited for local search
- **Swap Neighborhood:** Well-matched to permutation problems like TSP
- **Effective Restart Strategy:** 40 random restarts effectively escape local optima
- **Deterministic Exploration:** Complete neighborhood evaluation ensures thorough search
- **Dataset Characteristics:** The specific 50-city dataset favors local search approaches

6.2 Genetic Algorithm Performance Issues

- **Suboptimal Operators:** Order crossover may not preserve good TSP subtours
- **Parameter Sensitivity:** Fixed rates (crossover=0.9, mutation=0.03) may not be optimal
- **Population Diversity:** Premature convergence to suboptimal solutions
- **Fitness Landscape:** GA struggles with deceptive fitness landscapes
- **No Local Search:** Pure GA without local optimization operators

6.3 Practical Implications

Small Problems (10-20)

Both algorithms viable, Hill Climbing preferred for accuracy

Medium Problems (30-50)

Hill Climbing recommended despite longer runtime

Large Problems (>50)

Neither algorithm suitable; need advanced methods

7. Recommendations

7.1 Algorithm Selection Guidelines

Primary Choice: Hill Climbing with Random Restarts

- When solution quality is paramount
- For problems with 50 cities or fewer
- When computational time is not critical
- For reproducible, deterministic results

Alternative Approach: GA with Improved Configuration

- When computational time is limited
- For exploratory analysis of large problems
- When parallel computing resources are available
- As part of a hybrid algorithm framework

Hybrid Strategy: Memetic Algorithm

- Combine GA for global exploration
- Use Hill Climbing for local optimization

- Leverage strengths of both approaches
- Apply to medium and large-scale problems

7.2 Parameter Optimization

Hill Climbing Improvements

- **Dynamic Restarts:** Adaptive restart strategy based on stagnation
- **Variable Neighborhood:** Adjust neighborhood size during search
- **Adaptive Termination:** Dynamic iteration limits
- **Parallel Restarts:** Distribute restarts across CPU cores

Genetic Algorithm Improvements

- **Population Size:** Adaptive population sizing
- **Mutation Rates:** Dynamic mutation based on diversity
- **Advanced Operators:** Edge recombination, inversion mutation
- **Local Search:** Incorporate 2-opt or 3-opt local optimization

7.3 Future Work

- **Hybrid Algorithms:** Implement memetic algorithms combining GA and local search
- **Advanced Operators:** Test inversion mutation, scramble mutation, and other TSP-specific operators
- **Parallel Implementation:** Distribute restarts or population evaluation across multiple processors
- **Meta-heuristics:** Implement Simulated Annealing, Ant Colony Optimization, or Particle Swarm Optimization
- **Benchmarking:** Test on standard TSPLIB instances for comparison
- **Real-world Applications:** Apply to logistics, vehicle routing, and circuit design problems

8. Conclusion

This comprehensive analysis demonstrates that for the specific Euclidean TSP instance with 10-50 cities, Hill Climbing with Random Restarts significantly outperforms the implemented Genetic Algorithm in terms of solution quality, achieving perfect solutions across all tested problem sizes.

Solution Quality

Hill Climbing Wins

Perfect solutions vs degraded GA performance

Small Problems

Hill Climbing Wins

Both accurate, HC 10x faster

Large Problems

GA Faster but Inaccurate

Time vs accuracy trade-off

While Hill Climbing shows exponential time growth with problem size, its perfect solution reliability makes it the preferred choice for Euclidean TSP problems of moderate scale (≤50 cities). The Genetic Algorithm implementation, despite its poor performance here, remains valuable for larger problems where Hill Climbing's time complexity becomes prohibitive.

Final Recommendation: For this specific TSP instance and similar Euclidean problems, use Hill Climbing with Random Restarts for optimal solutions, with future work focusing on hybrid approaches for scalability.

9. Technical Specifications

9.1 Code Quality Assessment

- **Modularity:** Well-structured functions with clear responsibilities
- **Documentation:** Minimal but functional inline comments
- **Visualization:** Comprehensive plotting capabilities using Seaborn and Matplotlib
- **Reproducibility:** Fixed random seed ensures consistent experimental results
- **Error Handling:** Basic implementation with room for improvement

9.2 Limitations

- **Dataset Specificity:** Results may not generalize to all TSP instances or problem types
- **Implementation Simplicity:** Basic versions of both algorithms without advanced optimizations
- **Hardware Constraints:** Single-threaded execution limits performance
- **Parameter Tuning:** Fixed parameters not optimized for all problem sizes
- **Memory Usage:** Complete neighborhood evaluation can be memory intensive

9.3 Computational Requirements

Resource	Hill Climbing	Genetic Algorithm
Memory	$O(n^2)$ distance matrix	O(population × n)
Time (Small)	Fast (0.044-0.779s)	Moderate (0.411-0.633s)
Time (Large)	Slow (5.188-38.396s)	Fast (1.087-1.463s)
CPU Usage	Single-threaded	Parallelizable