Travelling Salesman Problem
Project Defense

Team Members

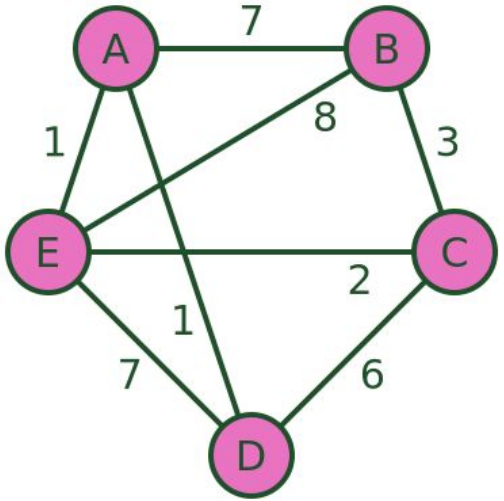
Patrick Barry
Buu Dinh Ha
Mahima Haridasan
Chelsy Mena
Serhii Vakulenko

What is the Travelling Salesman Problem?

The Travelling Salesman Problem (TSP) Definition

Given a list of cities and the distances between each pair of them, find the shortest possible route that visits each city exactly once and returns to the origin city.

Our goal is to find the most suitable algorithm to **approximate** a solution.



Algorithms Implemented

Algorithms Implemented

Team Member	Algorithm
Patrick Barry	Brute Force
	Ant Colony Optimisation
Buu Dinh Ha	Greedy
	Genetic Algorithm
Mahima Haridasan	Generic Branch and Bound
	Branch and Bound with Reduction Matrix and Edge Selection
Chelsy Mena	Dynamic Programming
	Markov Chain Monte Carlo
Serhii Vakulenko	Minimum Spanning Tree Approximation
	Lin-Kernighan

TSP Problems
and
Software Used

TSP Problems and Software Used

- TSP problems represented as a distance matrix.
  - Where each element (*i,j*) represents the Euclidean distance from city *i* to city *j*.
- Tests performed on:
  - Data from the University of Heidelberg, stored in the TSPLIB95 benchmark library.
  - Randomly generated TSP and asymmetric TSP (ATSP) instances built with own function.
- Variety of Python libraries used to implement and test our algorithms:
  - NumPy
  - TSPLIB95
  - Matplotlib

Experimental Setup

Experimental Setup

Functions used across the project

- **generate_tsp(n)** - Generates a random TSP problem of size n.
- **create_distance_matrix(problem)** - Given a problem (TSP or ATSP) in the standard TSPLIB95 format, returns the square matrix of n $\times$ n size with all distances.
- **Evaluation Metrics** - to compare the algorithms to each other, we measured the clock-time and deviation from optimal solution.
    - In the case of randomly generated problems, we measured the deviation from the best solution of that iteration.

1. Brute Force Implementation

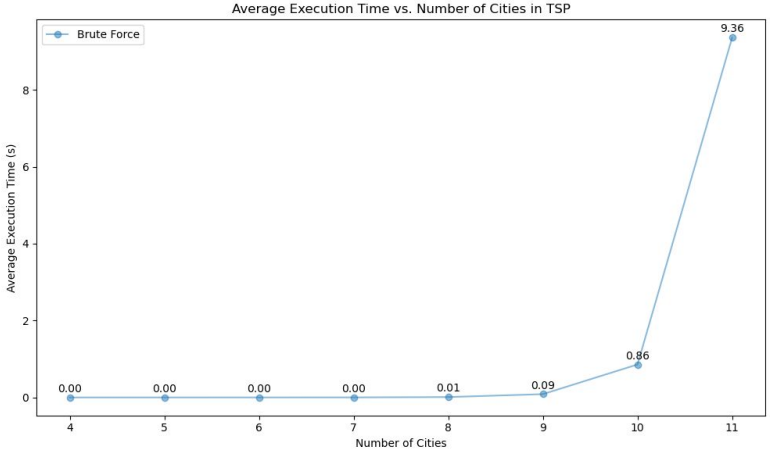
Brute Force

Advantages

- Simple to implement.
  - Checks every possible path.
- Guaranteed to find optimal solution.

Disadvantages

- Extremely slow
  - O(n!)
- Infeasible on problems of size 15 and more
  - Execution time of 5 days, then 80 days!



2. Ant Colony Optimisation

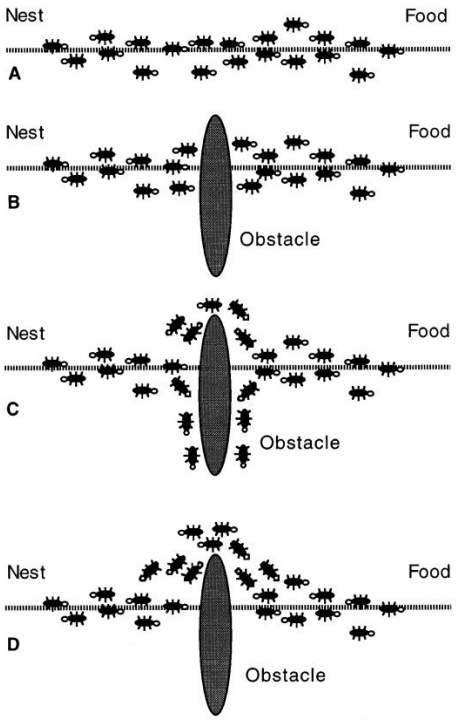
Ant Colony Optimisation

Basic Idea

- Ants find the shortest path to a food source using pheromone trails.
- The more ants traverse a path, the more pheromones are on that path.
- Each ant probabilistically follows a direction with a lot of pheromones.

Implementation

- Use pheromones and probabilistic choice for each edge in tour.
- Ant makes choice from combined distance and pheromone level, with random percentage.
- Shorter tours get more pheromones after each iteration.

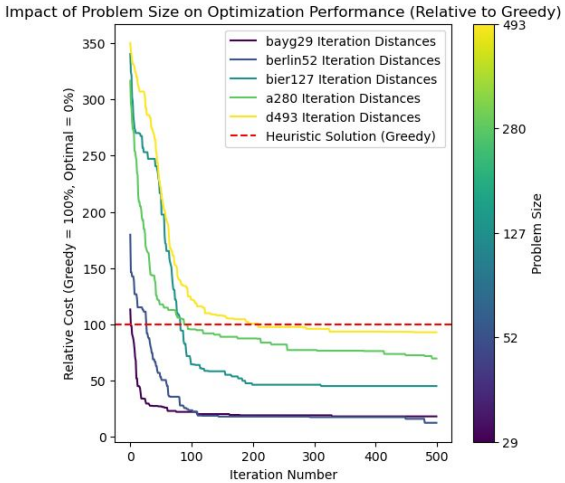


Visualisation from Dorigo et al. 1996.

Ant Colony Optimisation

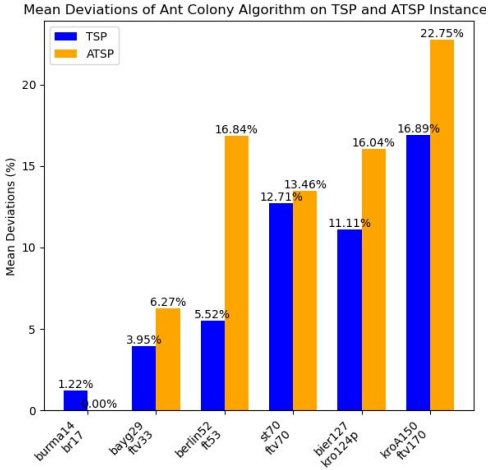
Advantages

- Good approximation of optimal solution.
- Use of probability decreases chance of premature convergence.
- Quickly betters greedy solutions.



Disadvantages

- Requires more iterations for larger problems.
  - $O(m \times T \times n^2)$
- Solution quality decreases as problem size increases.



3. Greedy

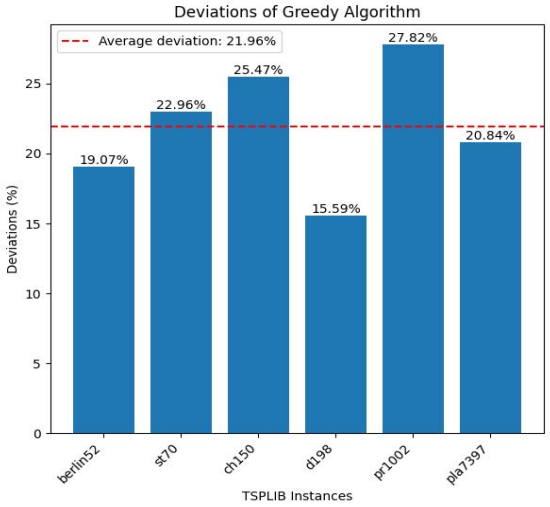
Greedy Approach

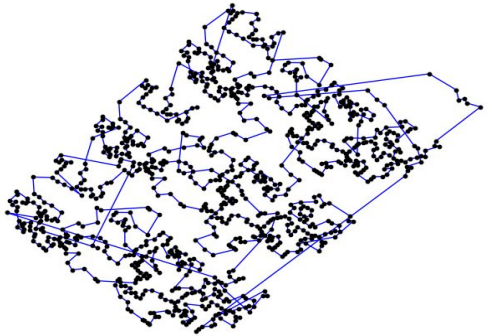
Advantages

- Simplicity and Intuitiveness
- Speed and Efficiency
- Useful for Approximate Solutions.

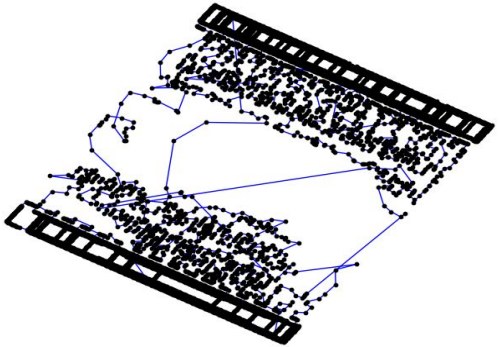
Disadvantages

- Suboptimal Solutions
- No Global View of the Problem
- Sensitivity to Poor Starting Choices





pr1002

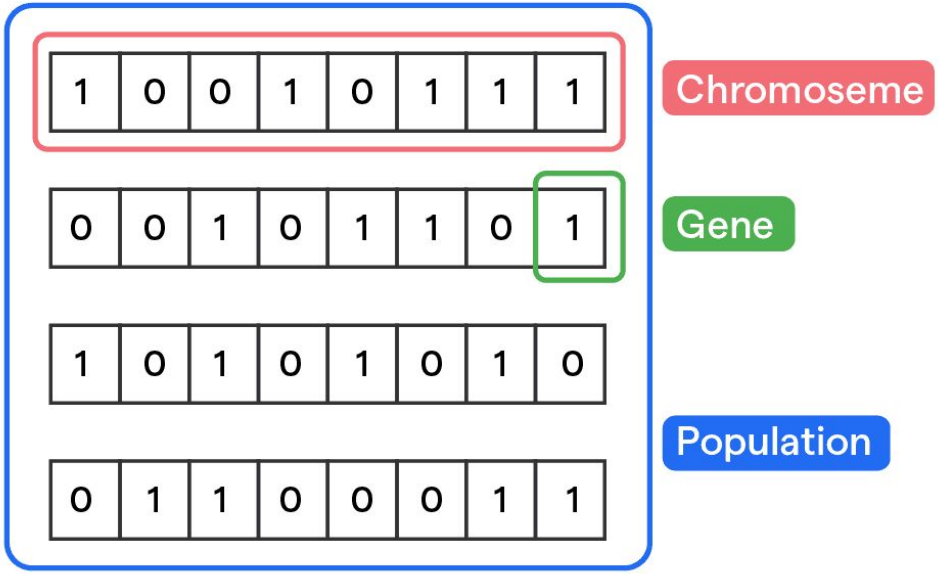


pla7397

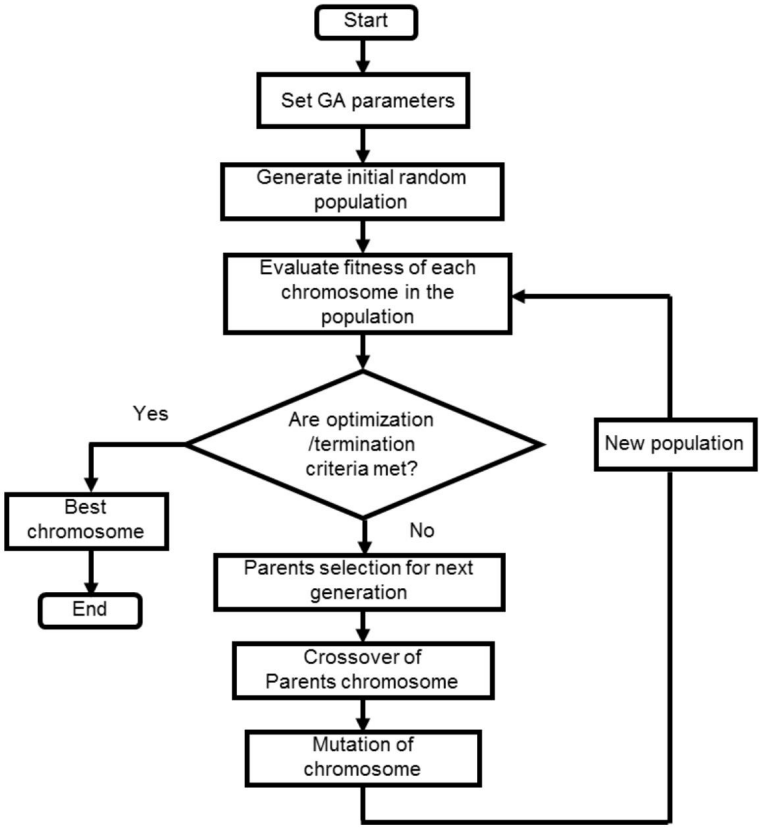
4. Genetic

Genetic Algorithm

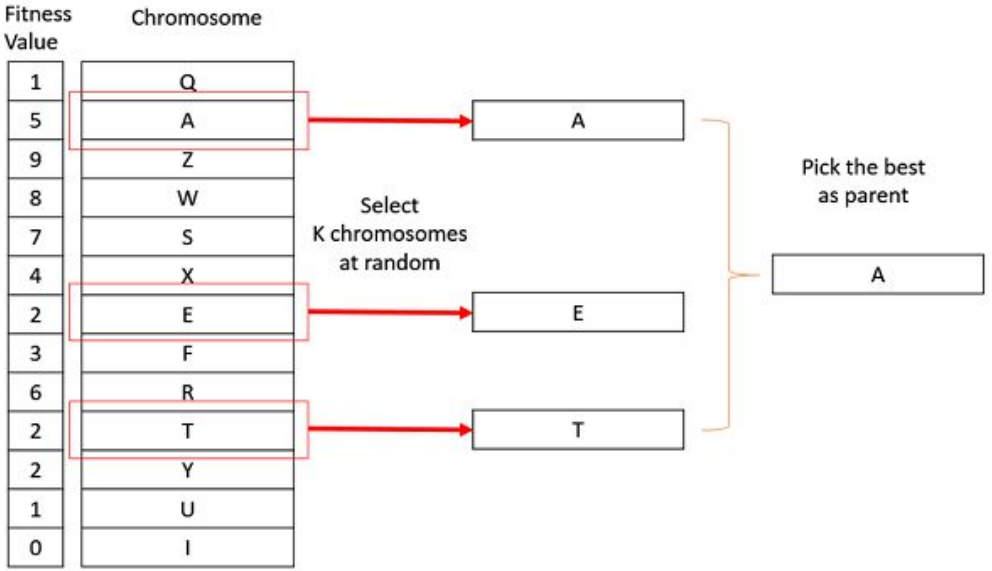
- Heuristic approach
- ‘Survival of the fittest’ principle
- Each chromosome represents a path



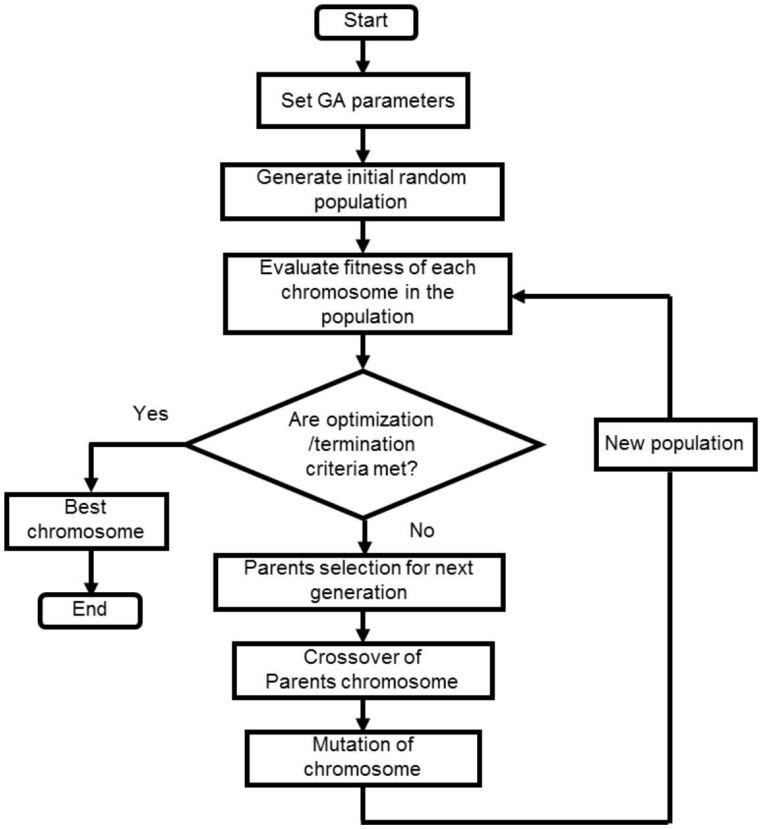
Genetic Algorithm



Tournament selection

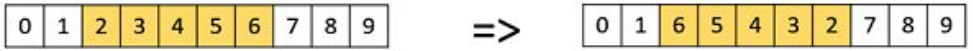


Genetic Algorithm



Order Crossover (OX1)

Reverse Mutation



5. Branch and Bound

Definition of Branch and Bound

- The approach consists of algorithms that follows the general rule of dividing into sub-problem(branching) and the use of a function to prune off sub-optimal cases(bounding).

- The time complexity of this approach greatly depends on the implementation of the algorithm as we can use this technique on various algorithms to reduce their complexity.

- In the best case the complexity will be linear and in the worst case the complexity will be exponential;but in general case for TSP is typically **exponential** because it explores a large number of possible combinations of cities.

Branch and Bound

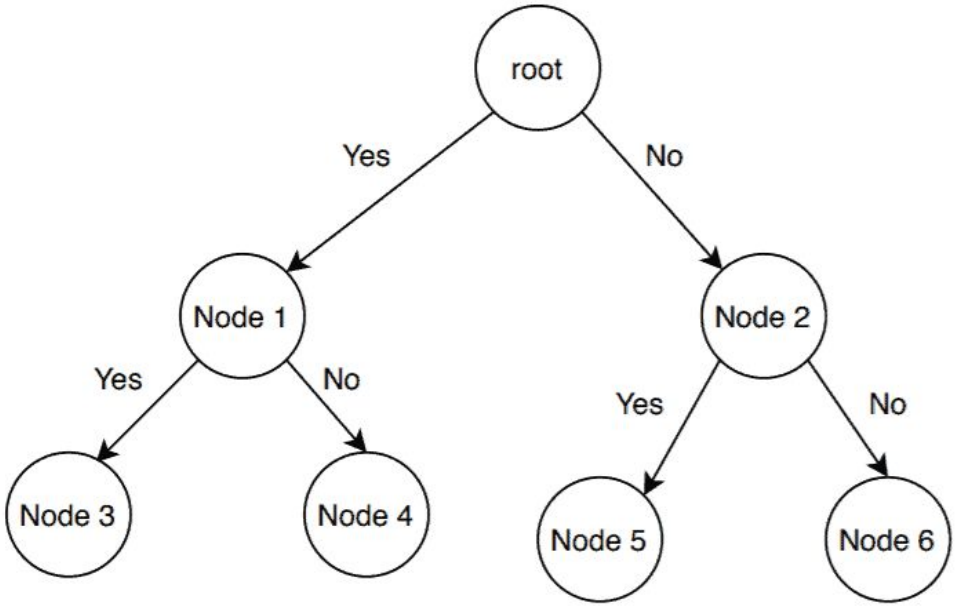
Views TSP as state-space tree

○ Node = city, child = connected city to current city

● Chooses next city by picking which one guarantees optimal solution

● Constructing state-space tree has time complexity = O(n!)

○ Worst case for branch and bound



6. Branch and Bound
with
Reduction Matrix and Edge Selection

∞	27	43	16	30	26
7	∞	16	1	30	25
20	13	∞	35	5	0
21	16	25	∞	18	18
12	46	27	48	∞	5
23	5	5	9	5	∞

Table 4.1: Cost matrix for 6-City traveling salesman problem

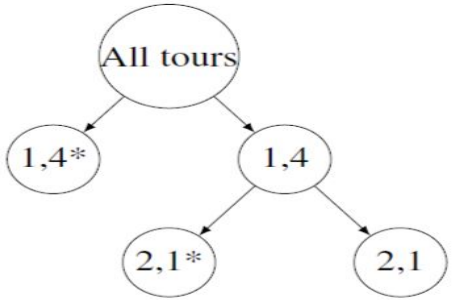


Figure 4.2: Start tree for the traveling salesman problem.

∞	11	27	0	14	10
1	∞	15	0	29	24
15	13	∞	35	5	0
0	0	9	∞	2	2
2	41	22	43	∞	0
13	0	0	4	0	∞

Table 4.2: Cost matrix after row and column reduction

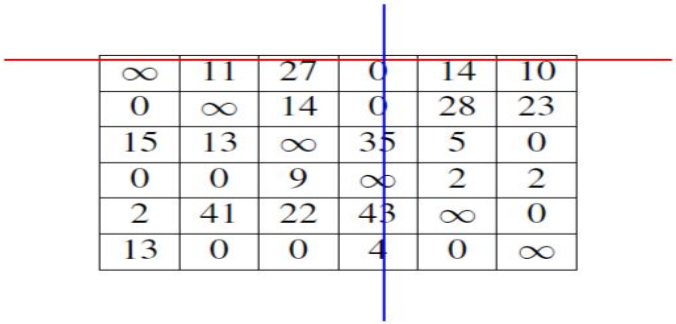
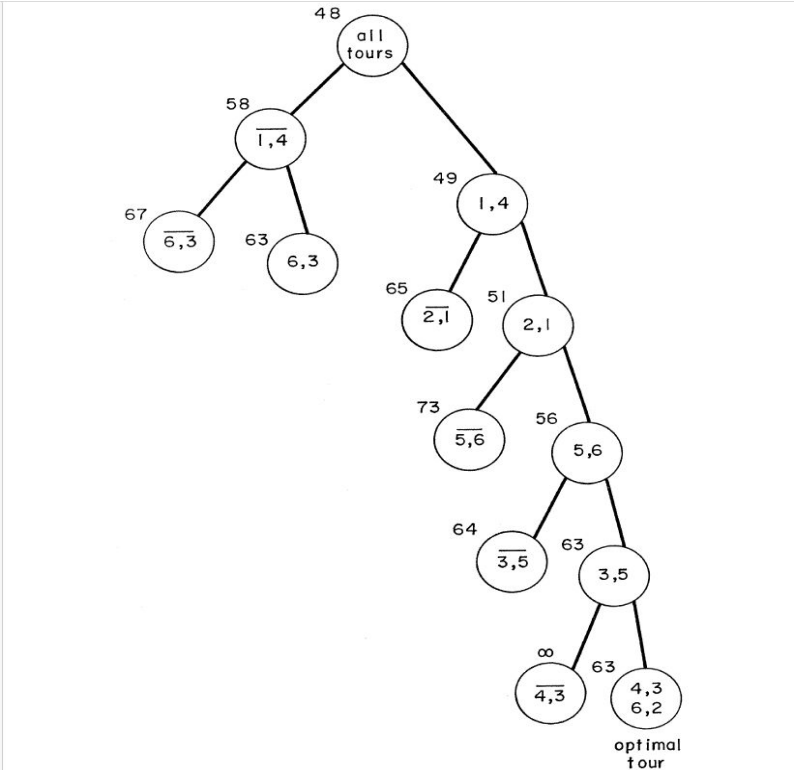


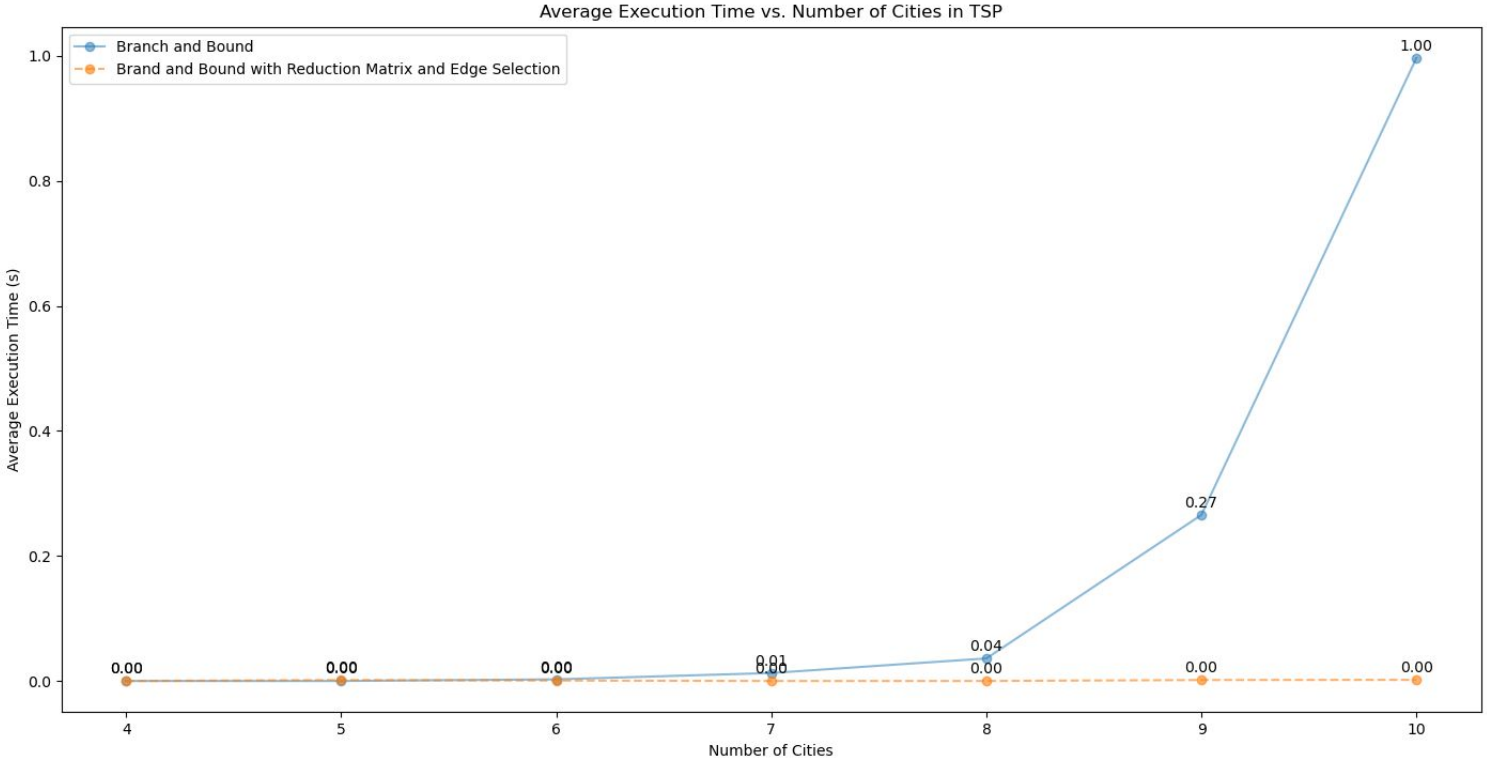
Figure 4.3: Matrix after deletion of row 1 and column 4.



Time Complexity: **O(N!.N)** (due to the exponential growth in the number of nodes and the cost calculation per node).

Space Complexity: **O(N!.N2)**(due to the storage of nodes in the priority queue and the space required for each node's matrix and path).

Runtime Comparison between BB & Reduction Matrix



7. Dynamic Programming

Dynamic Programming Approach

How does it work?

We take a top-down approach, where we take each possible subset of cities as a subproblem and finding the optimum route for it ensures a global optimum as we add another city to it.

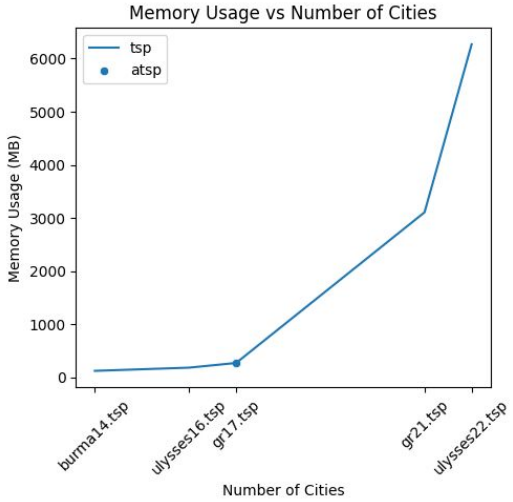
Advantages

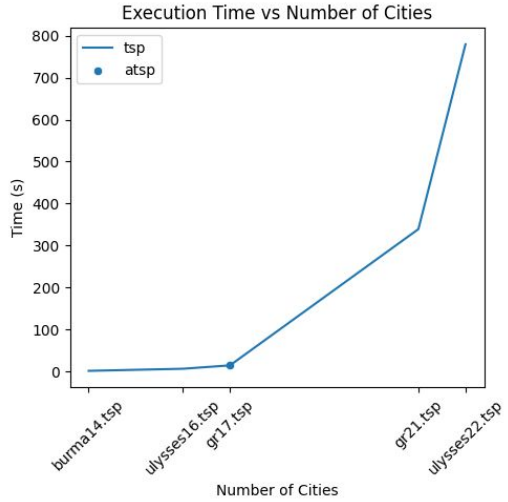
Gets the Optimum Cost and Route

Disadvantages

Takes O(n2 · 2n) time

Requires O(2n) space





Anything with more than 22 cities halts due to a Memory Error

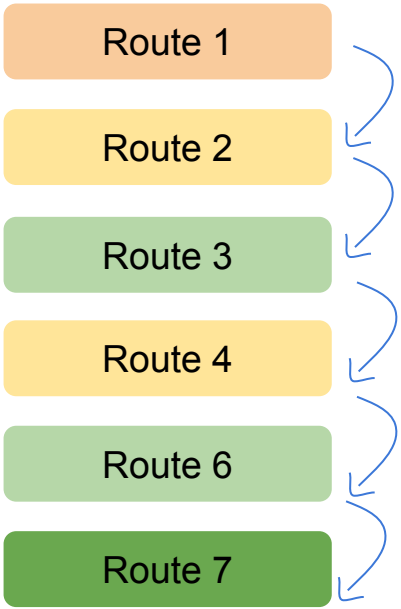
8. Markov Chain Monte Carlo

Markov Chain Monte Carlo

How does it work?

We combine a few concepts to traverse the solution space in a efficient but thorough way.

Markov Chain of Tours



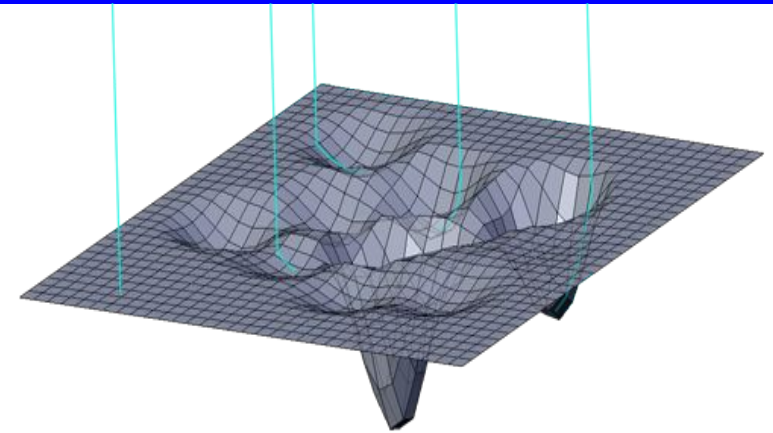
Monte Carlo Arbitrary Jump

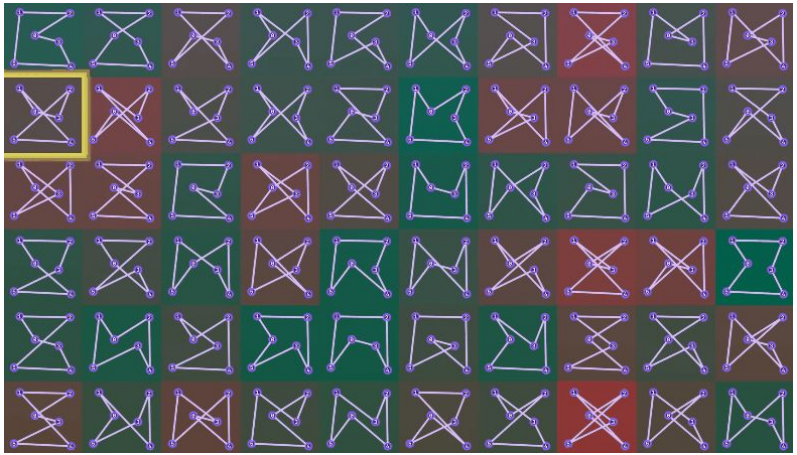
Transform the route you currently have

A favorable change is always accepted

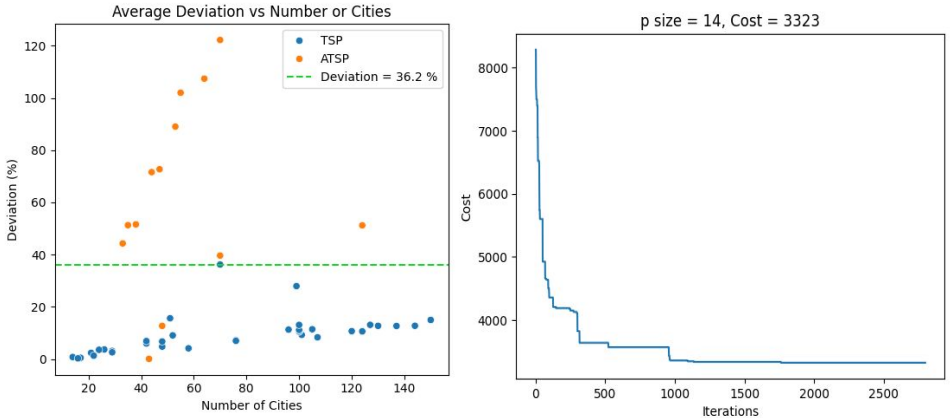
A non favorable one is only accepted with probability

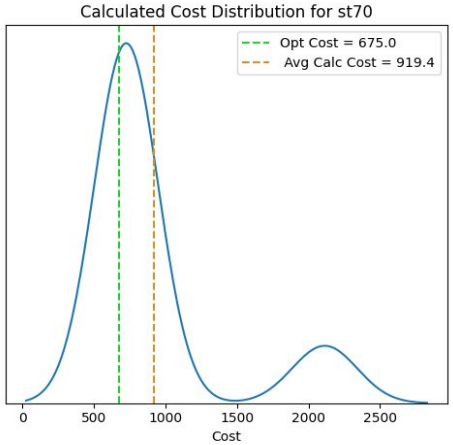
$$random\_value \leq e^{-\frac{\Delta}{T}}$$

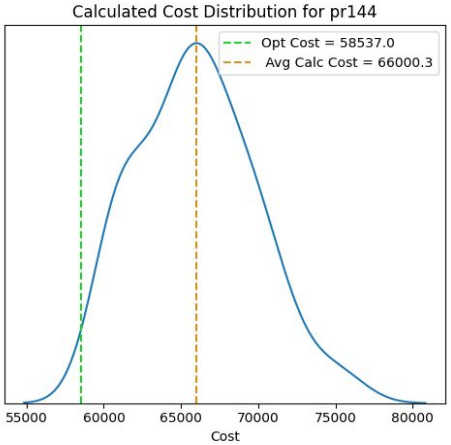




Markov Chain Monte Carlo







Advantages

- ★ Good approximation to the optimum cost and route, even in big problems
- ★ Complexity can be independent of the number of cities.
- ★ Small storage needs, best route and cost, and current route and cost.

Disadvantages

- ★ Inherits the weaknesses of the local search procedure
- ★ Trade-off between reproducibility of the result and runtime

9. Minimum Spanning Tree Approximation

Minimum Spanning Tree - Prim’s Algorithm

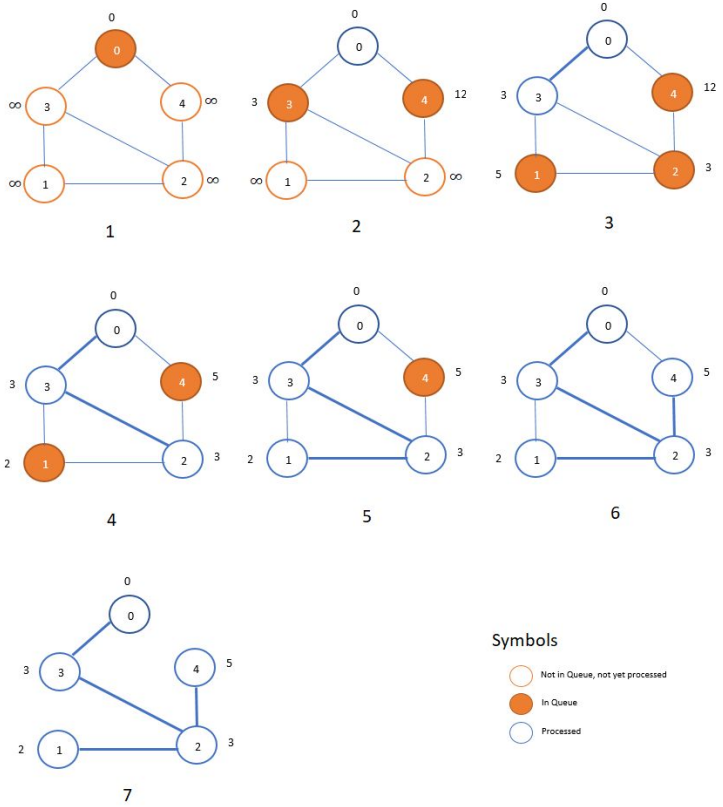
Prim's Algorithm

Purpose: To find a **Minimum Spanning Tree** in a weighted graph.

How It Works

1. **Select first city as starting point**
2. **Find and add shortest edge to any unvisited city**
3. **Repeat** until all vertices are included.

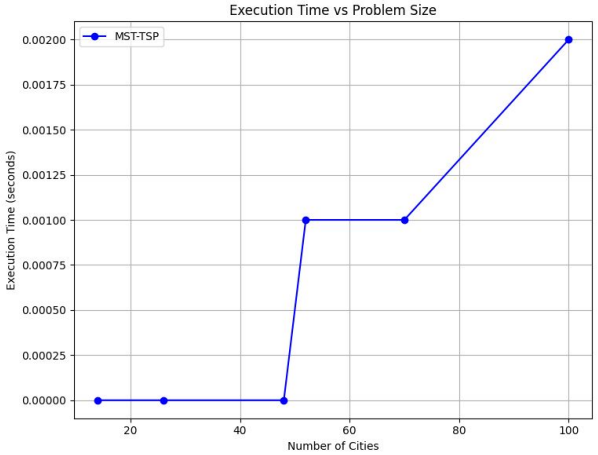
Once the **Minimum Spanning Tree (MST)** is built, the next step is to create a valid **TSP tour**. Since MST connects all the vertices with minimal edge weights but does not provide a valid TSP cycle directly, we need to adapt it by performing a **DFS (Depth-First Search)** traversal on the MST

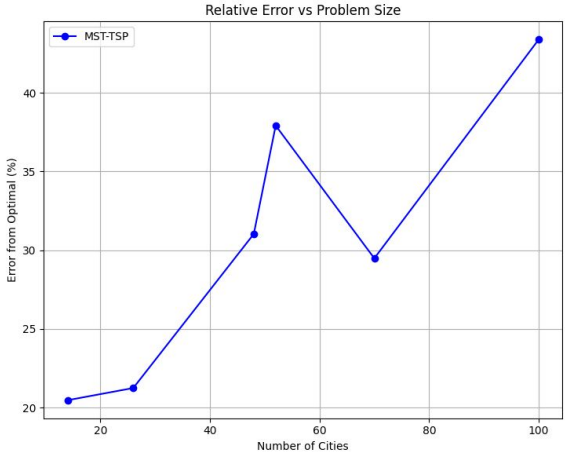


Minimum Spanning Tree - Prim's Algorithm

➔ The resulting TSP tour may **not always be optimal**.

➔ The generated TSP tour's weight is guaranteed to be at most **twice the MST cost** in the worst case.





Time complexity:

- MST Construction: $O(n^2)$
- TSP Tour Construction: $O(n)$
- Route Distance Calculation: $O(n)$

10. Lin-Kernighan

Lin-Kernighan (*partial case of 2-opt*)

Simplified version of Lin-Kernighan that repeatedly applies 2-opt moves while using tabu search and random restarts to avoid getting stuck in local optima.

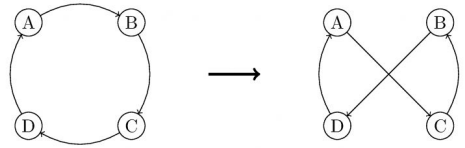
Start –> Create random initial tour

Main Loop - **k** (**try to improve current tour** up to **n** times)

current_tour

- **Check all possible edge pairs**
- **Skip adjacent and tabu edges**
- **Calculate gain for each swap**
- **Return best possible swap**

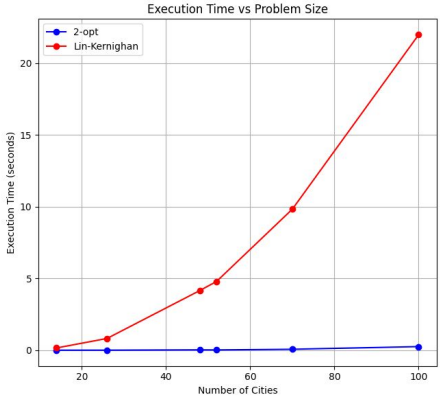
current_tour

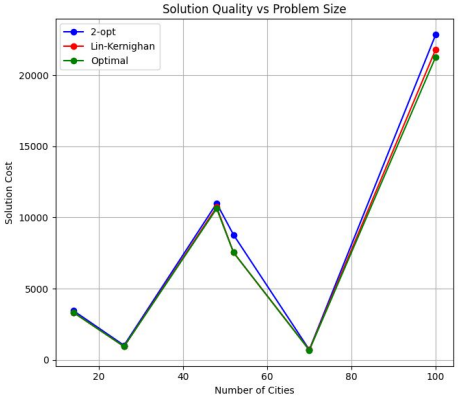


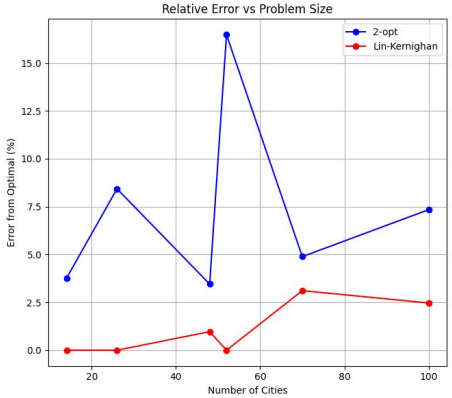
2-opt

10% chance to create new random tour

Lin-Kernighan (*partial case of 2-opt*)







Time complexity:

Overall: $\mathcal{O}(N^2 \times I)$

- N = number of cities
- I = number of iterations (1000 in my case)

Advantages	Disadvantages
Simple Implementation	Performance Issues
Flexibility	Solution Quality
Memory Efficient	Limited Moves

Results Analysis

Results Analysis

We chose to split the algorithms into three groups to enable easier comparison.

- Slow Algorithms
  - Brute Force (BF), both branch and bound algorithms (BB), and dynamic programming (DP).
- Medium Group - Good Approximate Solutions
  - Ant Colony Optimisation (ACO), Genetic, Lin-Kernighan (LK), Markov Chain.
- Fast Group - Poor solutions but very quick
  - Greedy, minimum spanning tree (MST) approximation.

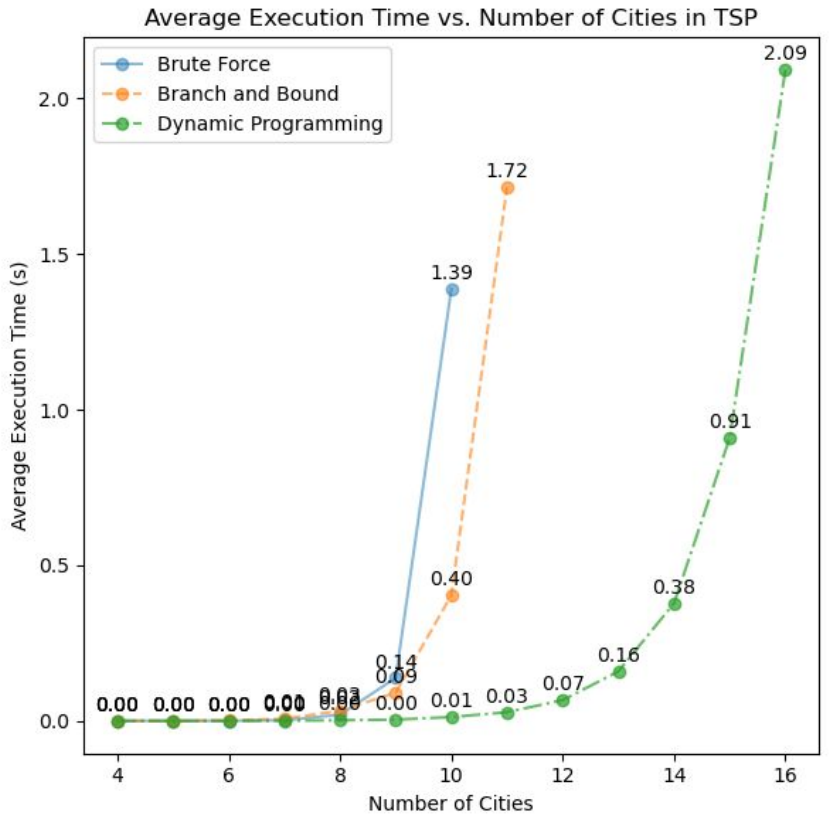
We took the mean deviation and execution time of 30 randomly generated problems to compare the algorithms.

Results Analysis - Slow (BF, BB, DP)

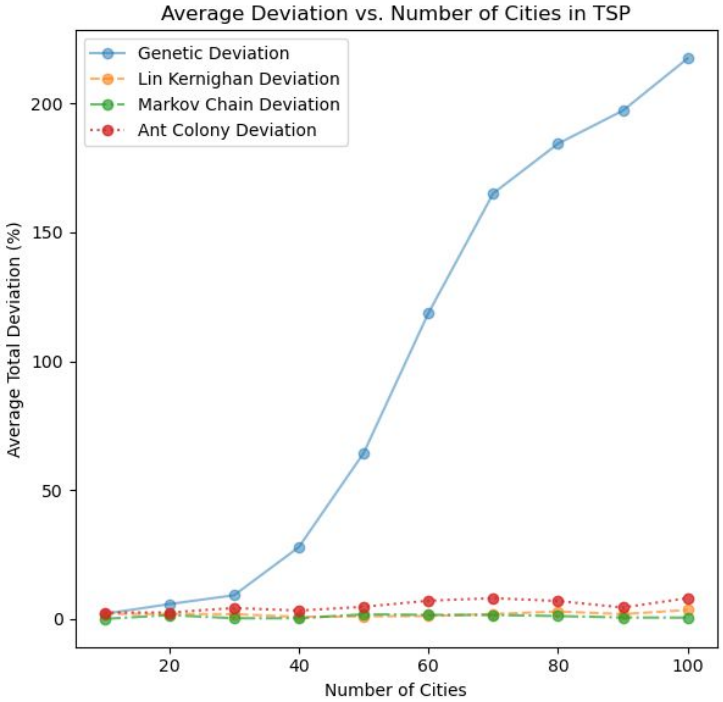
All algorithms guaranteed to find optimal cost

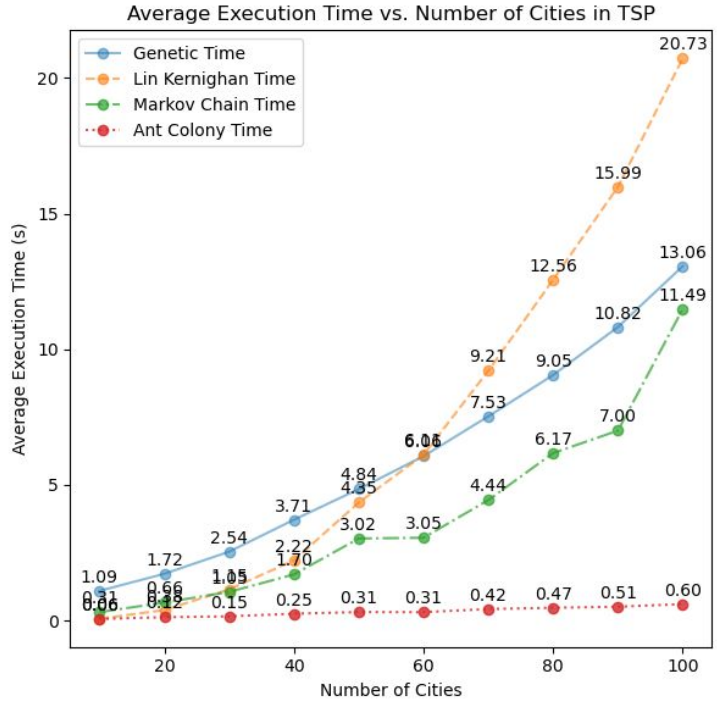
Compare execution time

- ★ The order of the algorithms is as expected
- ★ Proves that these algorithms are infeasible to use above problems of size 16 or 17.



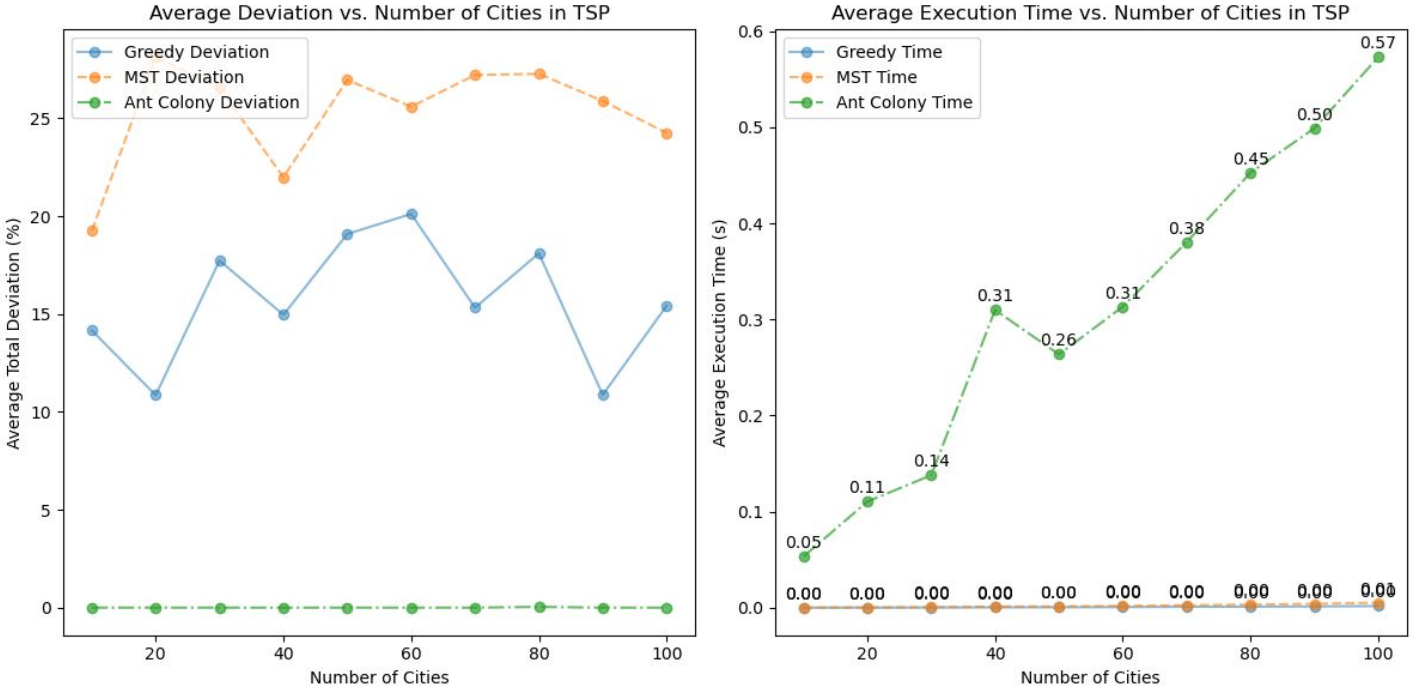
Results Analysis - Medium (ACO, Genetic, LK, Markov)





Results Analysis - Fast (Greedy, MST Approximation)

To provide a baseline, we took the best performing algorithm in terms of deviation and execution time from the previous group, ant colony optimisation.



Conclusion

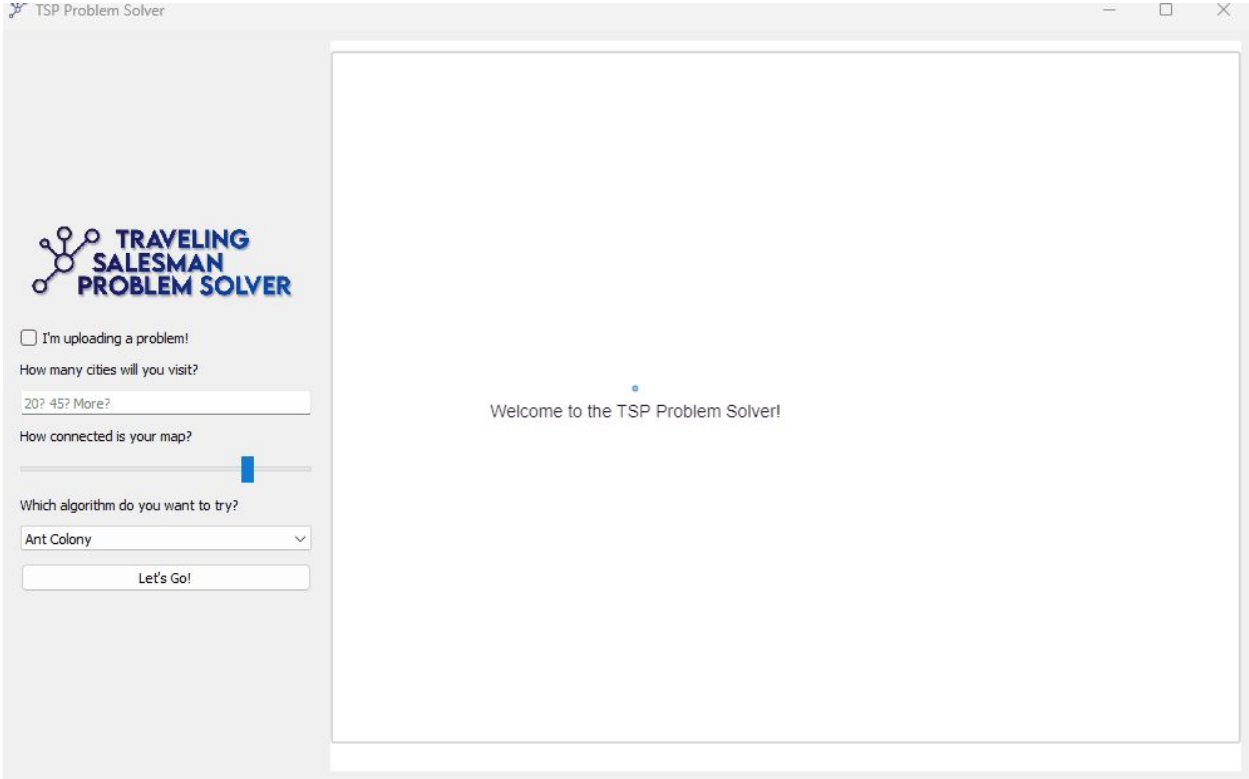
Conclusion

- We implemented various algorithms
  - Each algorithm has its use-cases
- There is a trade-off between efficiency and accuracy to be managed
- Greedy and MST approximations are useful to approximate the solution quickly
  - Used as a heuristic
- Algorithms like ACO, genetic, LK, and Markov chain are useful to get a solution closer to optimality in real-world cases.
- Brute Force, B+B and DP are useful for small problems to get the optimal solution.
  - And can be used to tune hyper-parameters of other algorithms.

GUI

Locally run interface to solve an instance of the TSPLIB library or generate a problem with a given size and sparsity.

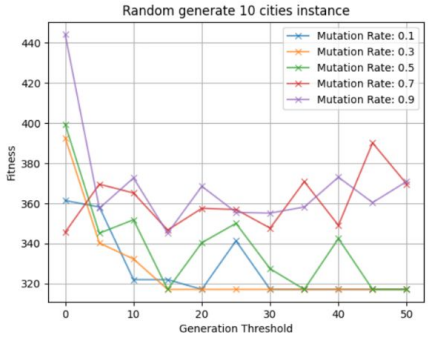
Allows visualization of chosen path and its cost and in particular cases shows links present in the optimum path and not in the one found.



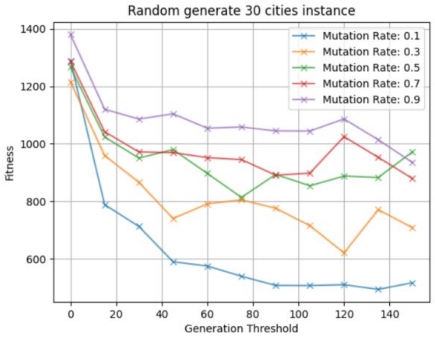
Tools Used: Python Libraries PyQt5 and pyvis; HTML and JavaScript

Thank you!
Any questions?

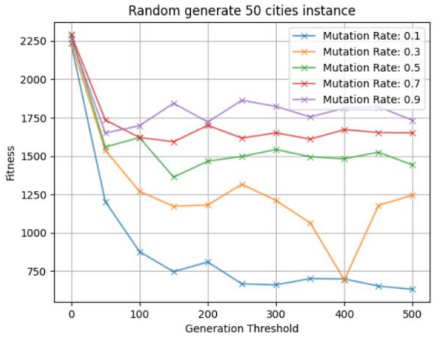
Appendix - Genetic Results



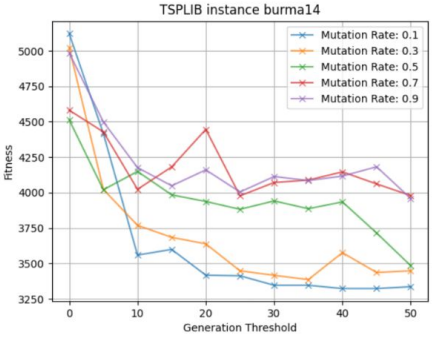
(a) 10 cities



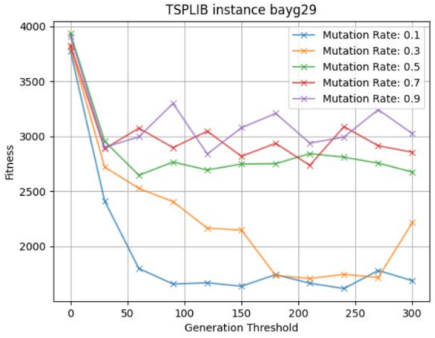
(b) 30 cities



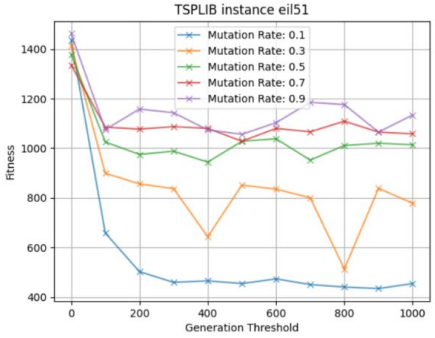
(c) 50 cities



(d) burma14



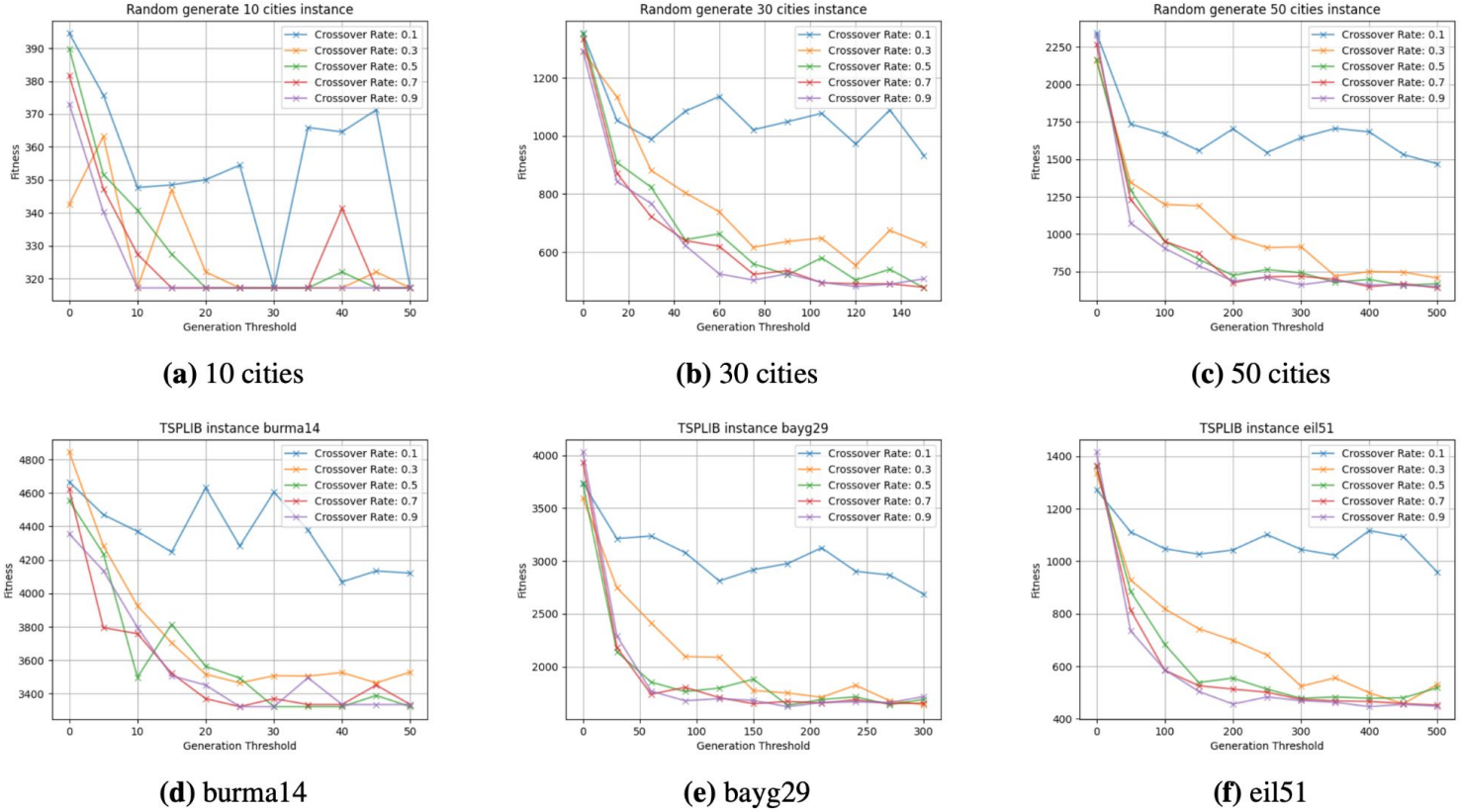
(e) bayg29



(f) eil51

Figure 0.12: Genetic performance with different mutation rate

Appendix - Genetic Results

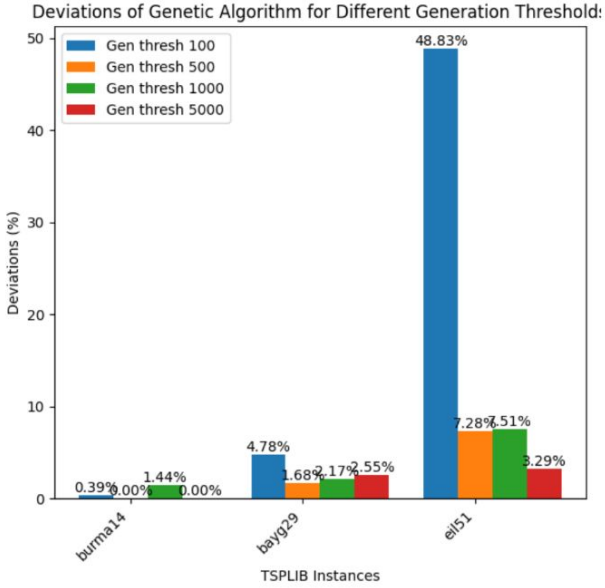


(a) 10 cities **(b)** 30 cities **(c)** 50 cities

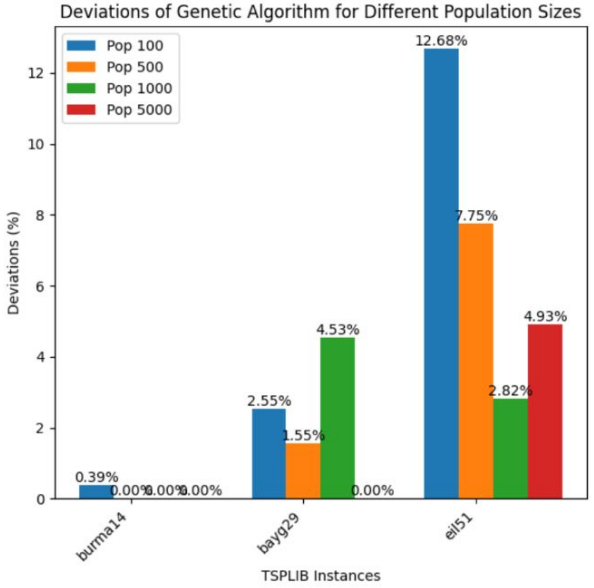
(d) burma14 **(e)** bayg29 **(f)** eil51

Figure 0.13: Genetic performance with different crossover rate

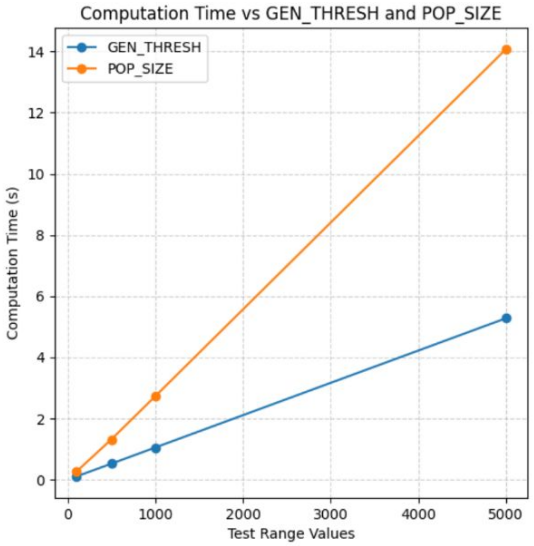
Appendix - Genetic Results



(a) vs Generation threshold



(b) vs Population size



(c) Computation Time

Figure 0.14: Deviation of genetic solution from optimal solution