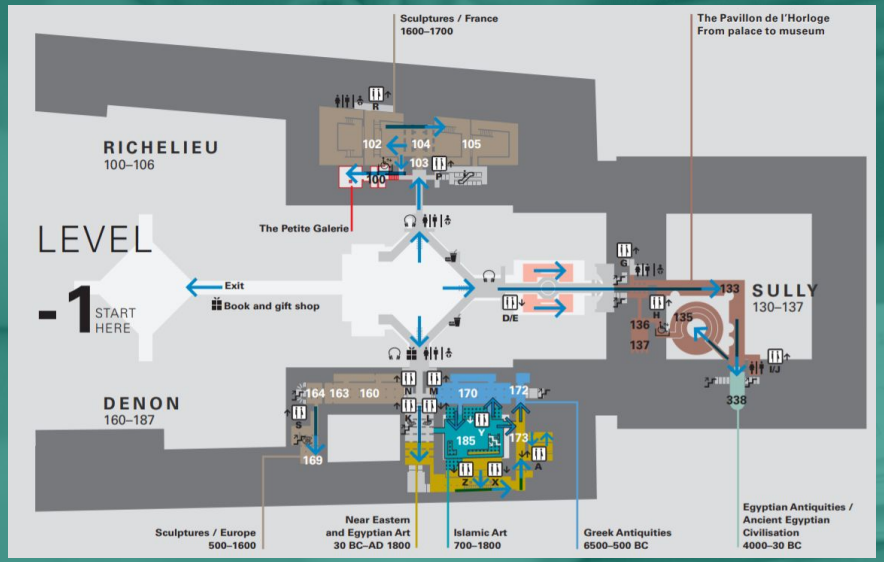
The Museum Problem

Édition du musée du Louvre

Team - Sic Mundus

Problem Statement



Optimize the route for a tourist visiting The Louvre Museum, such that the satisfaction level is maximised by visiting all/select exhibits in a single working day. With coinciding exit and entry points.

Design Variables

1. $(2 \times {}^{N}C_{2})$ - Path indicator variables

Problem 1

- This is the simplest version of the museum path optimization problem, which requires the tourist to visit all the exhibits located at lattice points separated by known distances.
- The objective is to find an optimized sequence of visiting the exhibits such that the total path length is minimised.

$$min \sum_{i} \sum_{j=1} c_{ij} y_{ij}$$

$$s.t \sum_{i<k} y_{ik} + \sum_{j>k} y_{kj} = 2, k \in V$$

$$\sum_{i} \sum_{j} y_{ij} \leq |S| - 1, S \subset V, 3 \leq |S| \leq n - 3$$

$$y_{ij} \in 0, 1, \forall [i, j] \in E$$

Problem 2

- Consider a model, where certain points cannot be reached by all of the points in the space of lattice points
  - Models exhibits in museum situated at different floors; can be accessed only from certain entry/exit points.
- Problem is asymmetric, represents paths which don't exist in both directions.
  - Models one-way routes, and/or routes with different departure and arrival rates.

$$min \sum_{i} \sum_{j=1} c_{ij} y_{ij}$$

$$s.t \sum_{j} y_{ij} = 1, i = 0, 1, \ldots, n-1$$

$$\sum_{i} y_{ij} = 1, j = 0, 1, \ldots, n-1$$

$$\sum_{i} \sum_{j} y_{ij} \leq |S| - 1, S \subset V, 2 \leq |S| \leq n-2$$

$$y_{ij} \in 0, 1, \forall [i, j] \in E$$

$$y_{ij} \in 0, \forall [i, j] \in E'$$

Problem 3

- Consider a model, where the satisfaction level of the tourist needs to maximised over a fixed interval of time
  - Models a tourist who prioritises visiting exhibits with higher popularity index in order to leave the museum at the end of the day with maximum satisfaction

Note: Satisfaction level based on popularity index of an exhibit is approximated to be a deterministic variable in this case, complexities can be added in the future by considering it to be a probabilistic variable.

$$max \sum_{i=1}^{n} s_i \sum_{j} y_{ij}$$

$$\sum_{i=1}^{n-1} y_{ij} = \sum_{k=2}^{n} y_{jk} \leq 1, s.t. j = 2, \ldots, n-1$$

$$y_{ij} \in 0, 1 \forall i, j \in V$$

$$\sum_{i} \sum_{j} y_{ij} \leq |S| - 1, S \subset V, 2 \leq |S| \leq n-2$$

$$\sum_{i} \sum_{j} (\frac{c_{ij}}{v} + \tau) y_{ij} \leq T_0$$

Optimization Results

Link to GitHub Repository: Museum-Path-Optimization

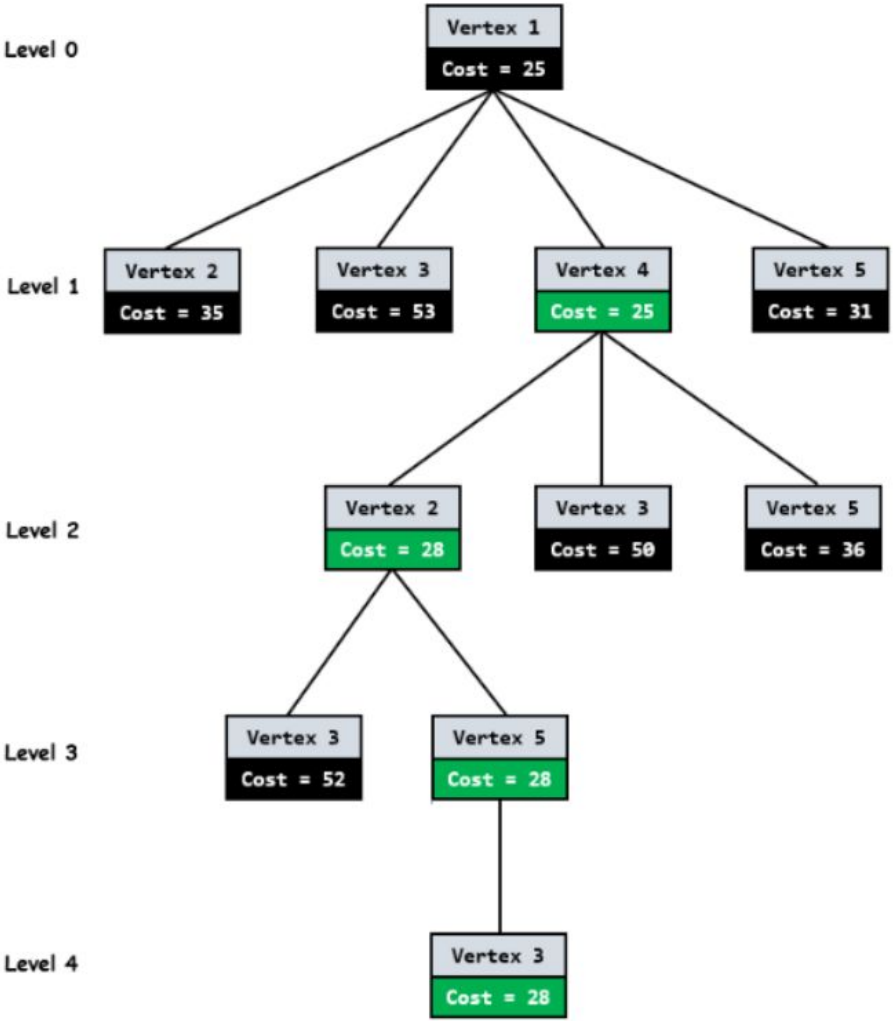
Code Speedup

- **Numba Library**
  - Just-in-Time (JIT) compiler package was used
  - The compiled machine byte-code provides a significant speedup
  - **Branch and Bound**
    - Speedup of **2.1X**
  - **Simulated Annealing**
    - Speedup of **3.2X**

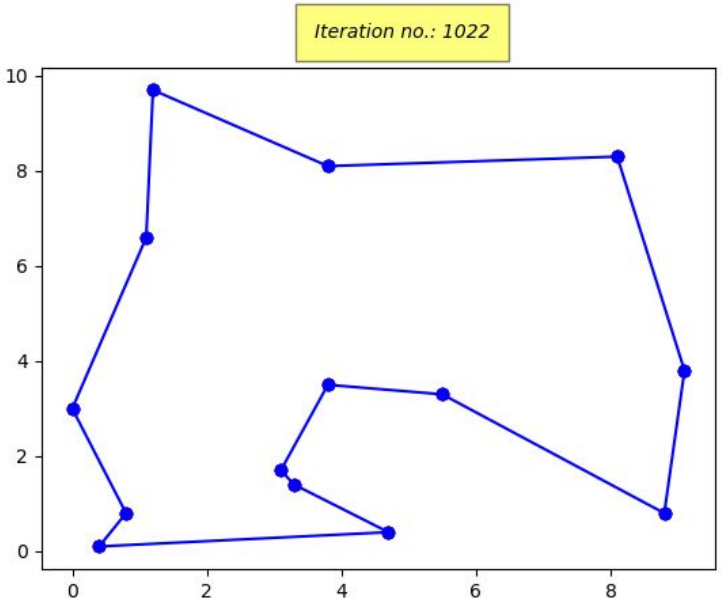
Branch and Bound

Branch and Bound

```
==================================================
Solver Summary:
==================================================

Run on: 18/03/2021 16:35:32 | PC: LAPTOP-L1T2T565
Test case name: Manual input - 5 nodes

Solved in: 0.004 s
Number of reduction function calls: 17

The optimal tour is:
0 --> 2
2 --> 3
3 --> 1
1 --> 4
4 --> 0

Total cost is 34.0

==================================================
```

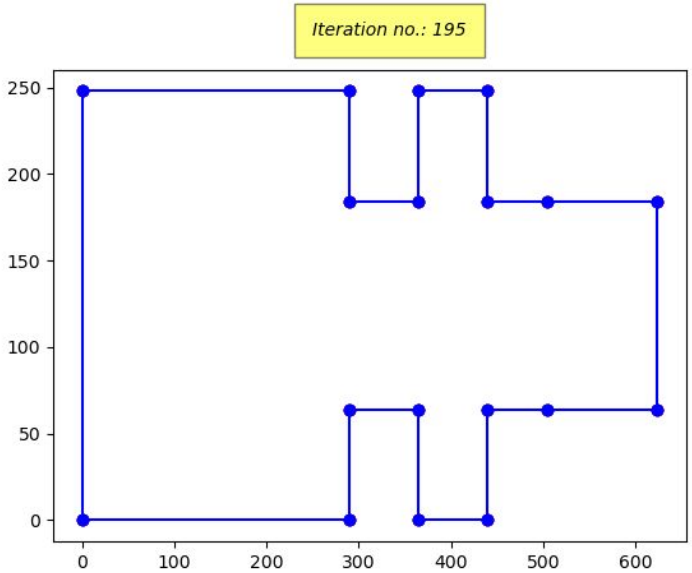


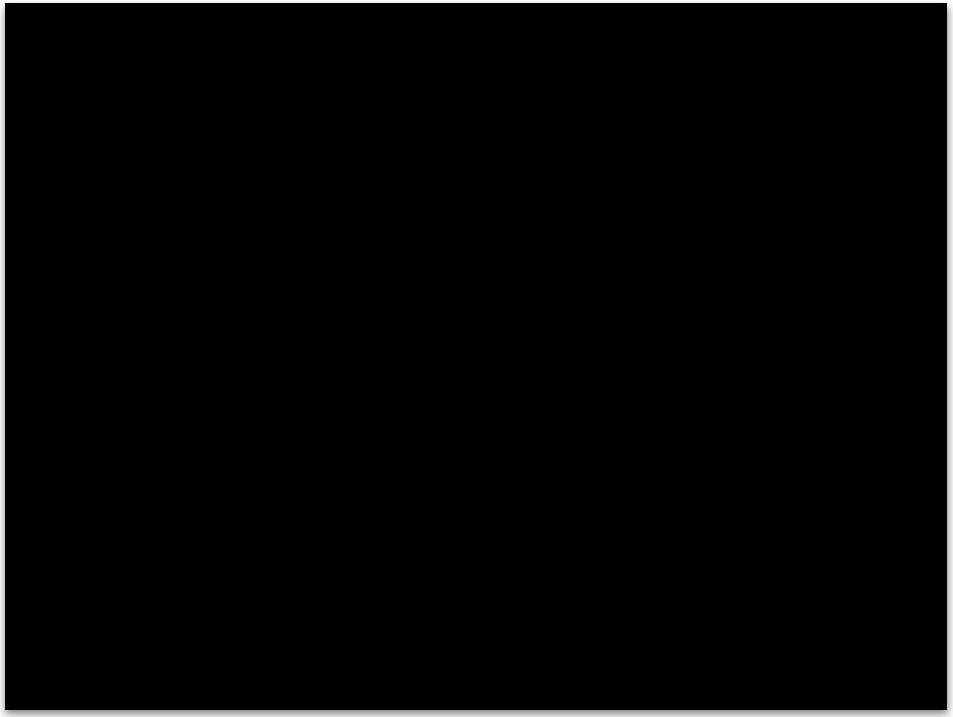
Problem 1
Symmetric connected points



Problem 2: Specified paths only

Applied to a basic floor plan of ‘The Louvre’





Standard Test Case:

Burma14

Running time for the algorithm is ~ 9.25 sec!

```
================================================
Solver Summary:
================================================

Run on: 18/03/2021 16:31:11 | PC: LAPTOP-L1T2T565
Test case name: burma14

Solved in: 9.248 s
Number of reduction function calls: 12985

The optimal tour is:
0 --> 9
9 --> 8
8 --> 10
10 --> 7
7 --> 12
12 --> 6
6 --> 11
11 --> 5
5 --> 4
4 --> 3
3 --> 2
2 --> 13
13 --> 1
1 --> 0

Total cost is 3323.0

================================================
```

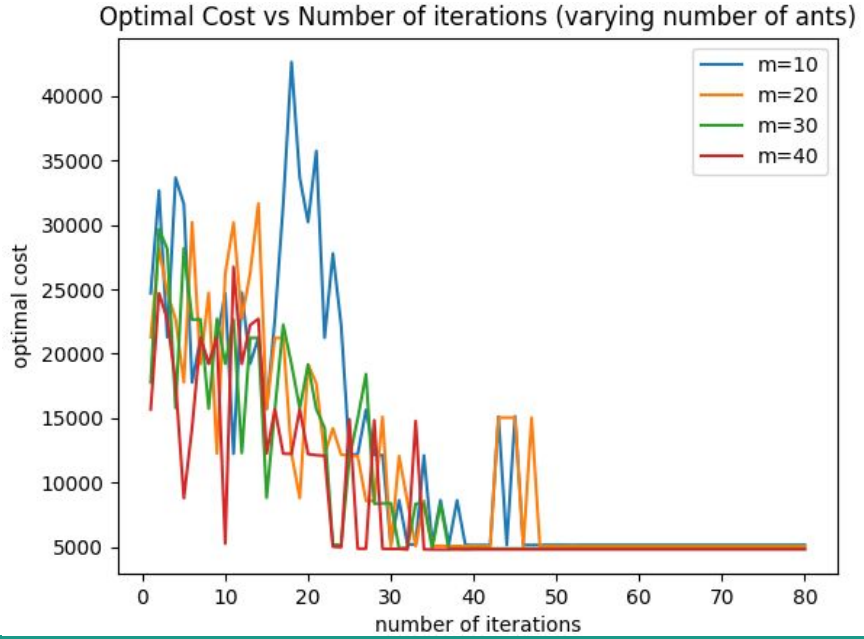
Branch and Bound

- Exponential time complexity **O(n!)**
- The algorithm gives results faster for sparse cost matrices.
- On applying it to Problem 3 it gets transformed to Greedy algorithm.
  So its implementation is not possible here.

Ant Colony Optimization

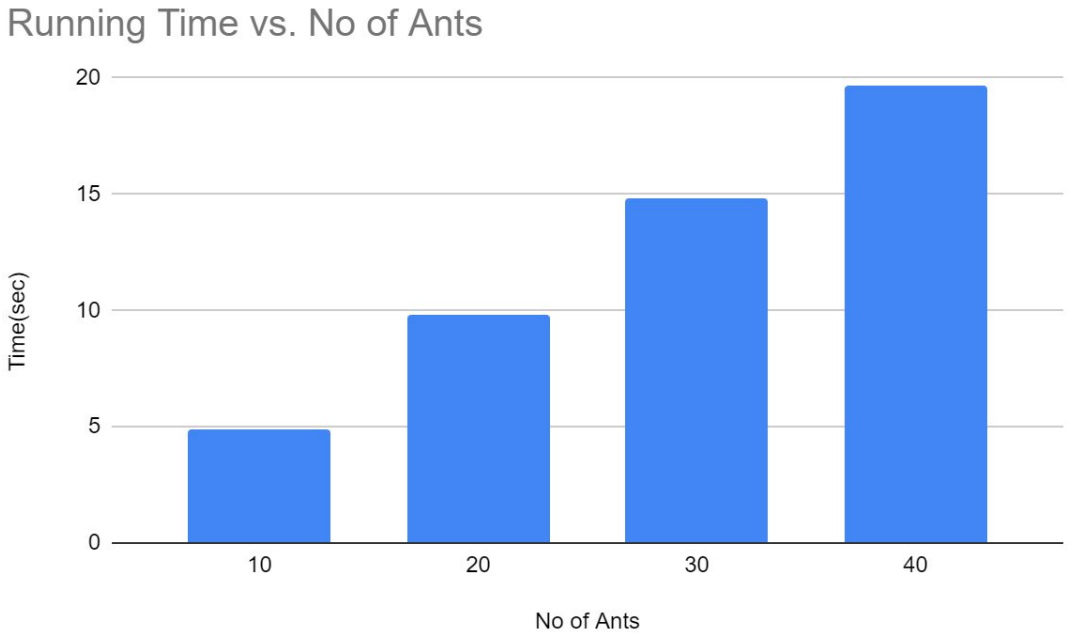
Ant-Colony Optimization

Problem 1 - Symmetric Problem



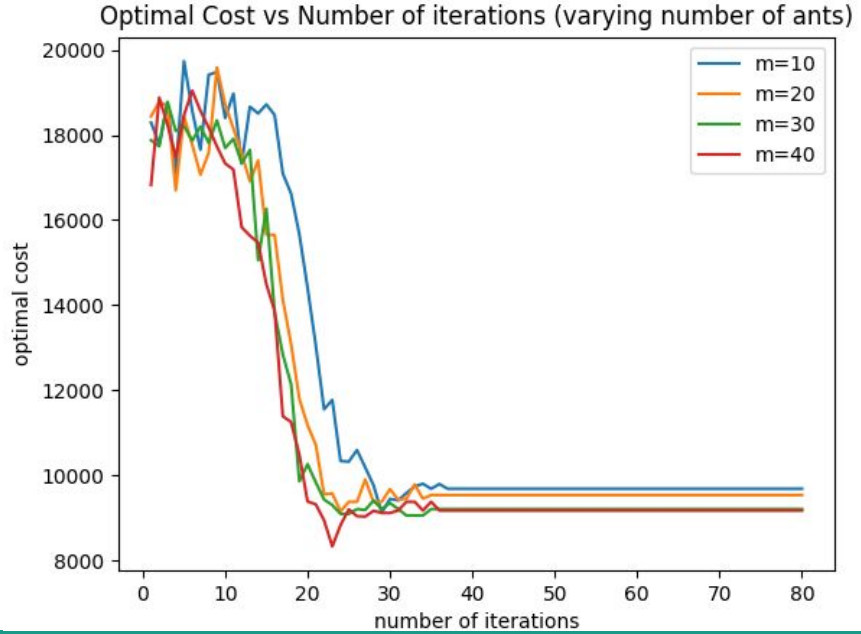
Ant-Colony Optimization

Problem 1- Symmetric Problem



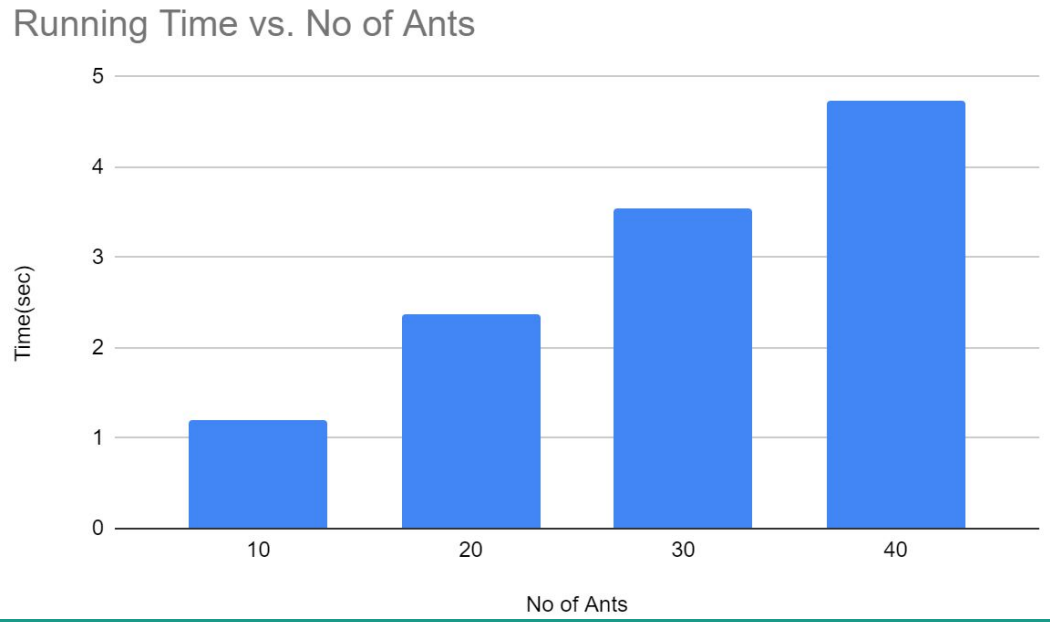
Ant-Colony Optimization

Problem 2 - Asymmetric Problem



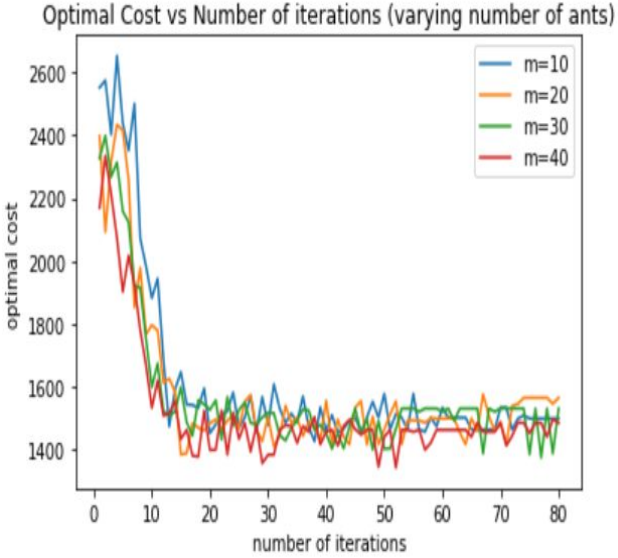
Ant-Colony Optimization

Problem 2 - Asymmetric Problem

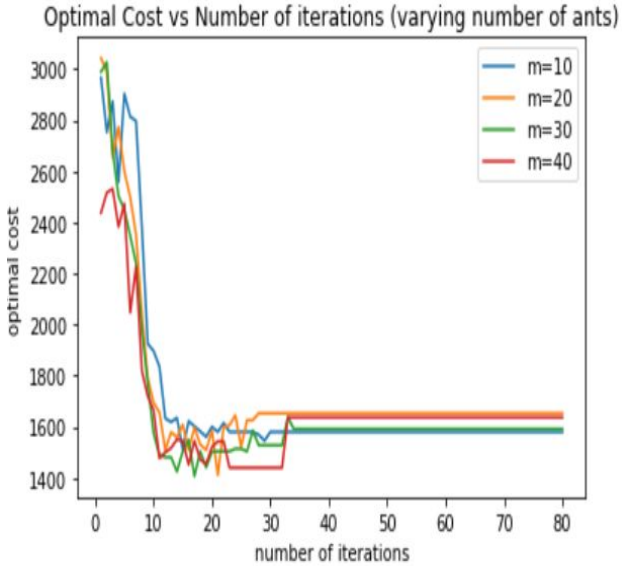


Ant-Colony Optimization

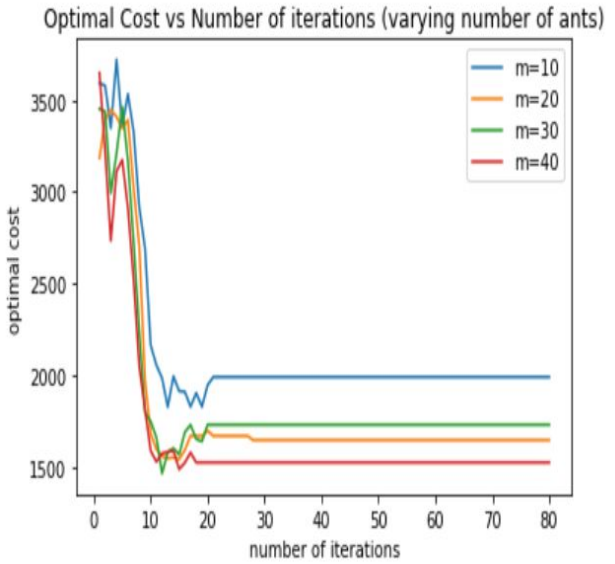
α = 2, □ = 1



α = 1.5, □ = 1.5

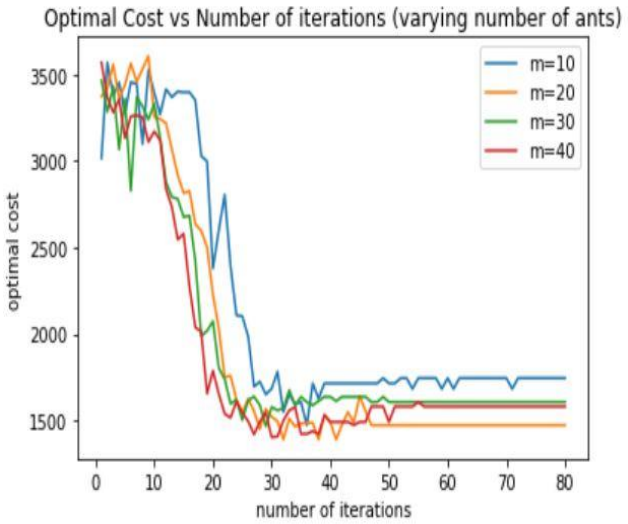


α = 1, □ = 2

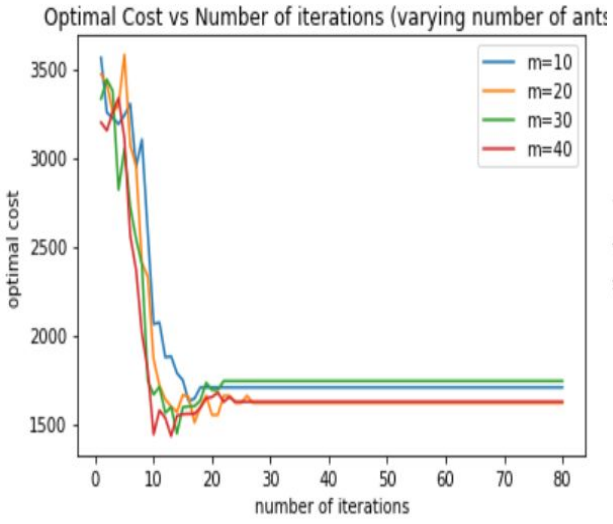


Ant-Colony Optimization

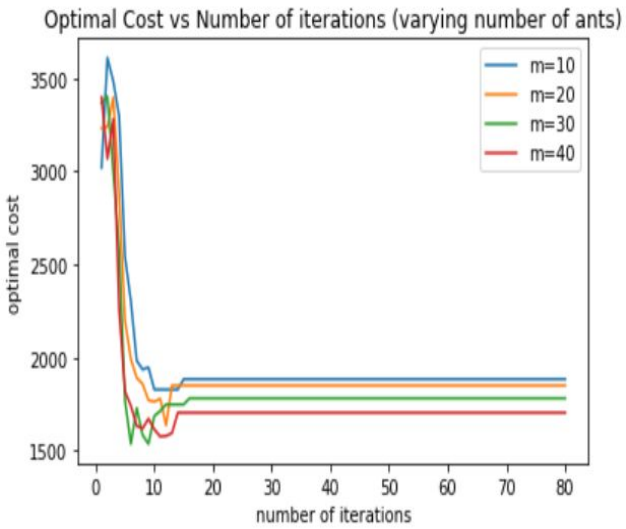
e = 0.2



e = 0.5

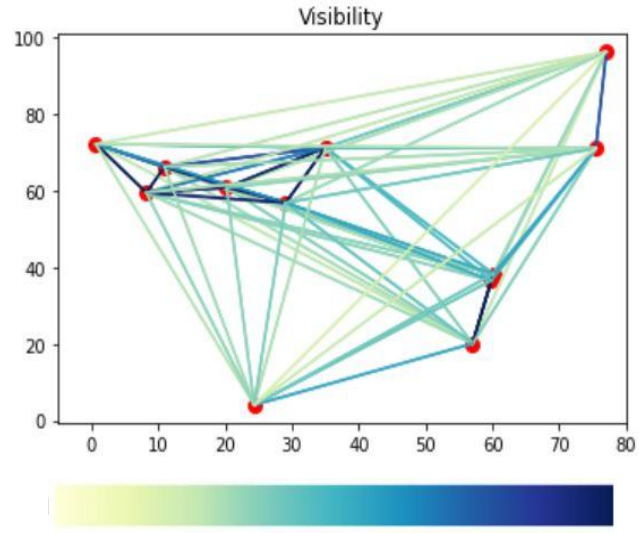


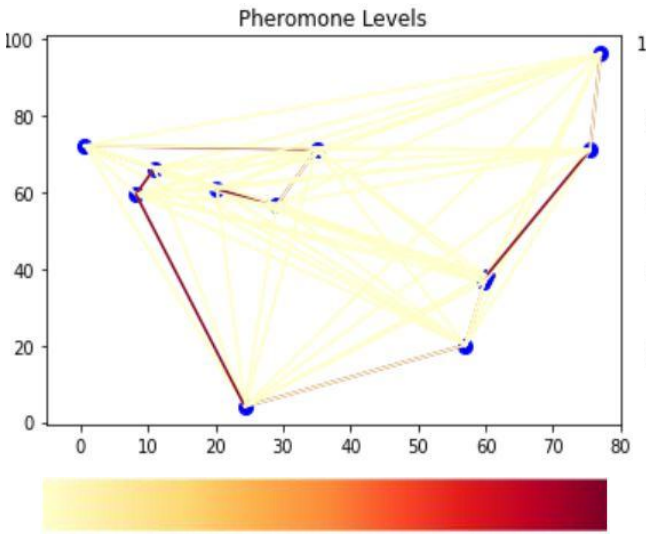
e = 0.8

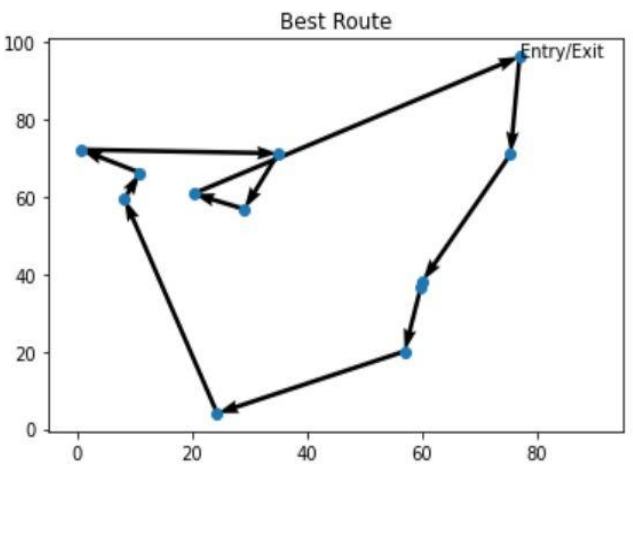


Ant-Colony Optimization

No. of iterations = 80, No. of ants = 10, No. of exhibits = 12

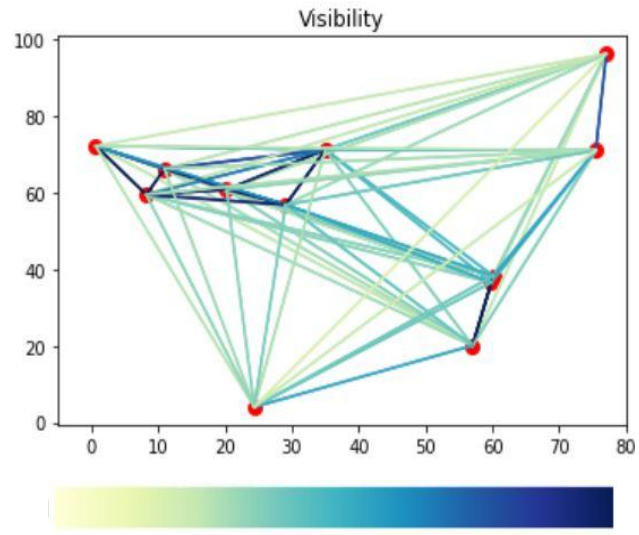


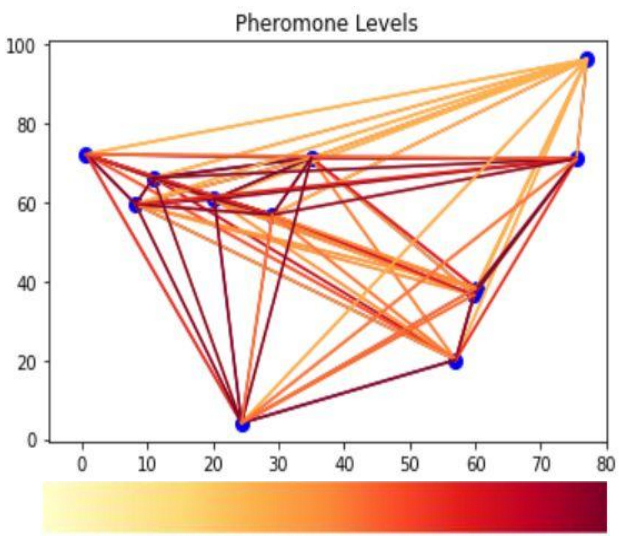


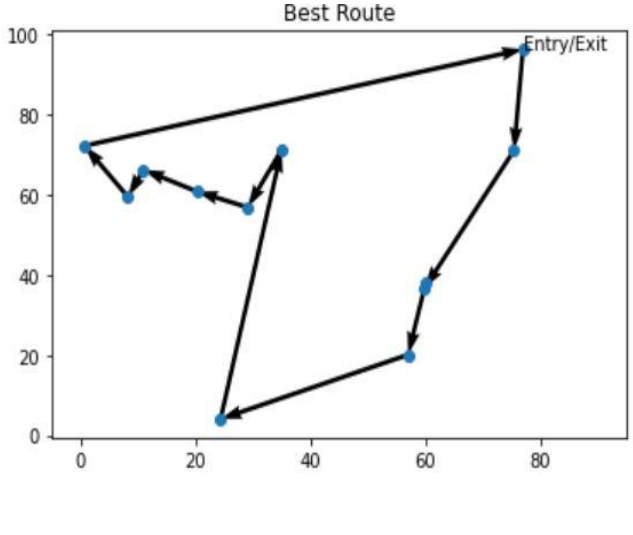


Ant-Colony Optimization

No. of iterations = 25, No. of ants = 10, No. of exhibits = 12



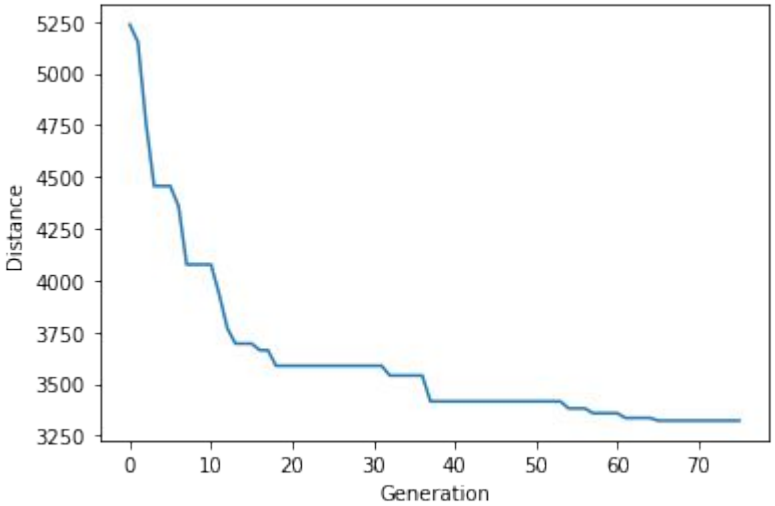




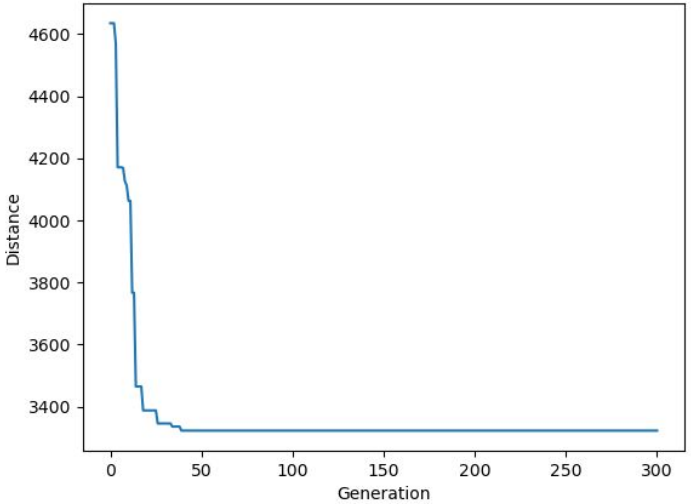
Genetic Algorithm

Genetic Algorithm

- Problem 1 - Symmetric Case



TSPLIB: Burma14
Optimal Cost: 3323
(150 Popsize, 20 Elitesize, 0.002 Mutrate)



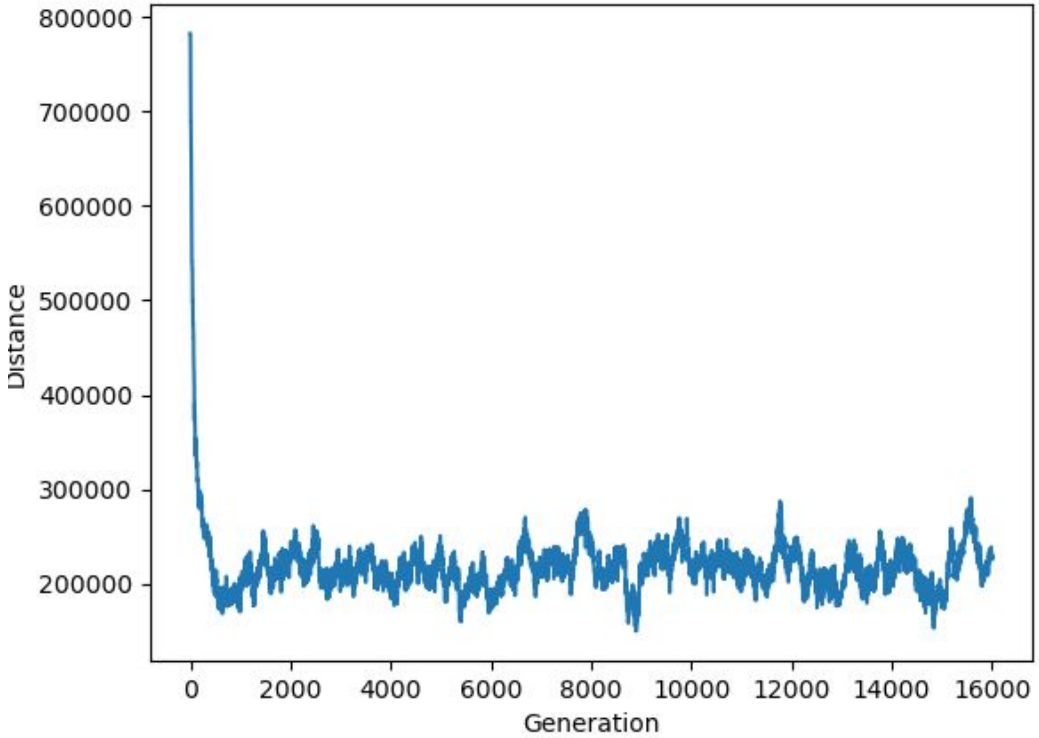
TSPLIB: Att48
Optimal Cost: 1142
(150 Popsize, 20 Elitesize, 0.002 Mutrate)

Genetic Algorithm

- High Mutation Rate

TSPLIB: Bier127
Calculated Final Cost: 214995
Calculated Min Cost: 17243
Global Optimum Cost: 118282

Parameters:
Popsize- 200
Elitesize- 300
Mutation rate- 0.02
Generations- 16000

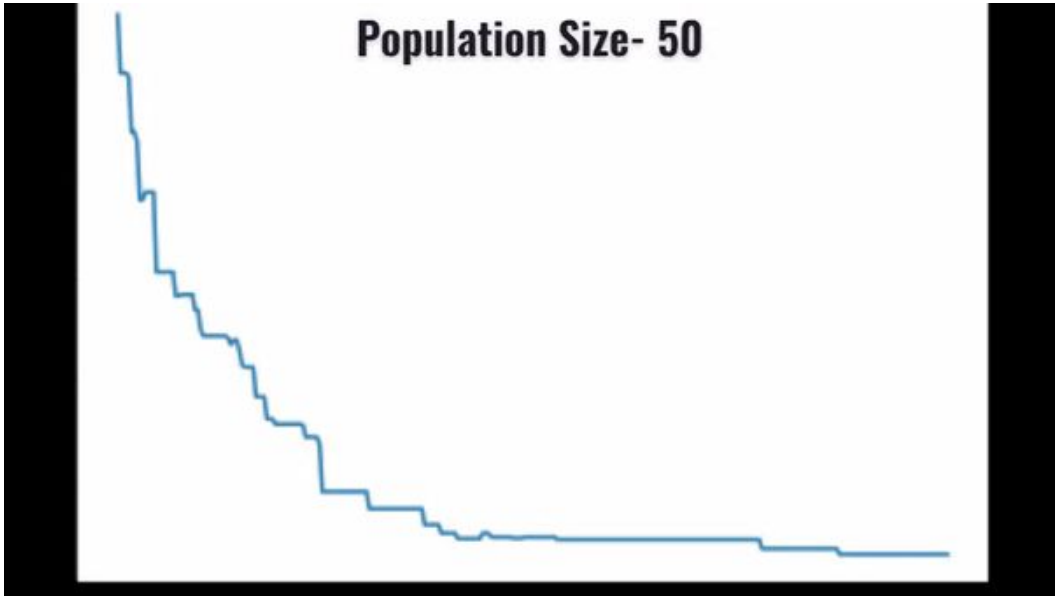


Genetic Algorithm

TSPLIB: Bays29
Fixed Parameters:
(Elitesize- 20, mr- 0.001)
Generations- 300

Effect of Varying Population Size

Population Size	Cost	Run- time (sec)
50	2127	6
100	2161	27
150	2026	62
200	2213	110
250	2162	125

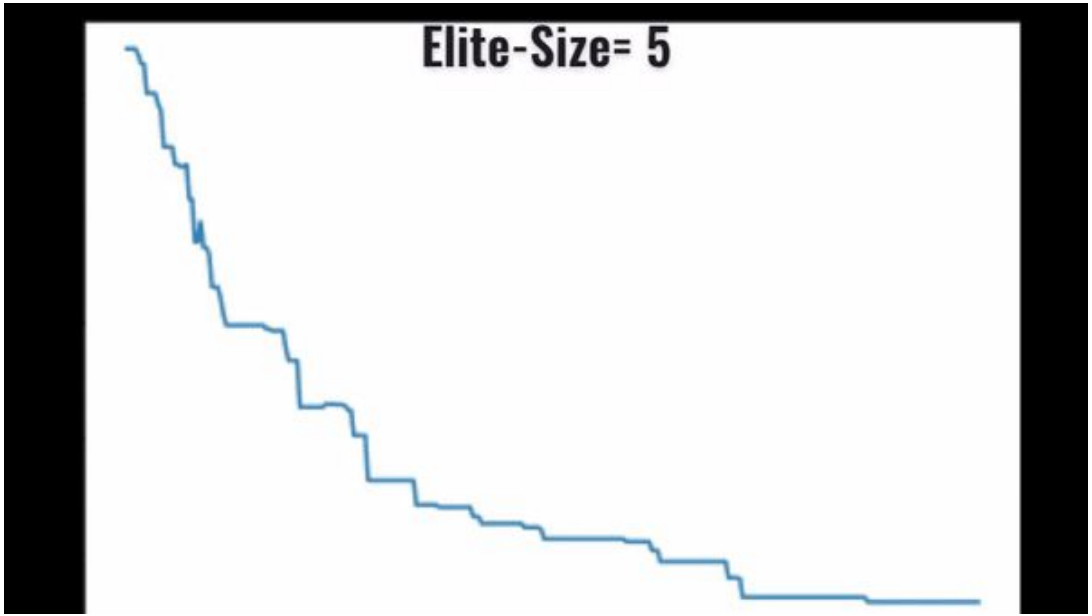


Genetic Algorithm

TSPLIB: Bays29
Fixed Parameters:
(Popsize- 150, mr- 0.001)
Generations- 300

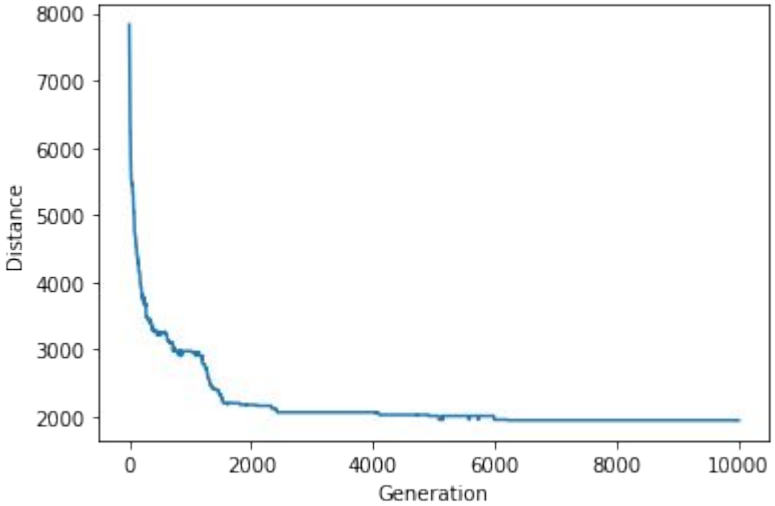
Effect of Varying Elite Size

Elite Size	Cost	Run- time
5	2285	68
10	2084	76
15	2144	62
30	2068	65
50	2131	48
75	2072	37

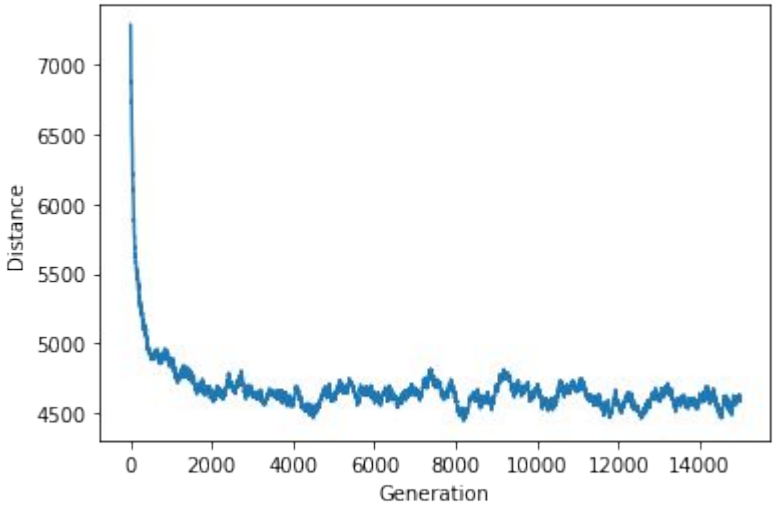


Genetic Algorithm

- Problem 2 - Asymmetric Case



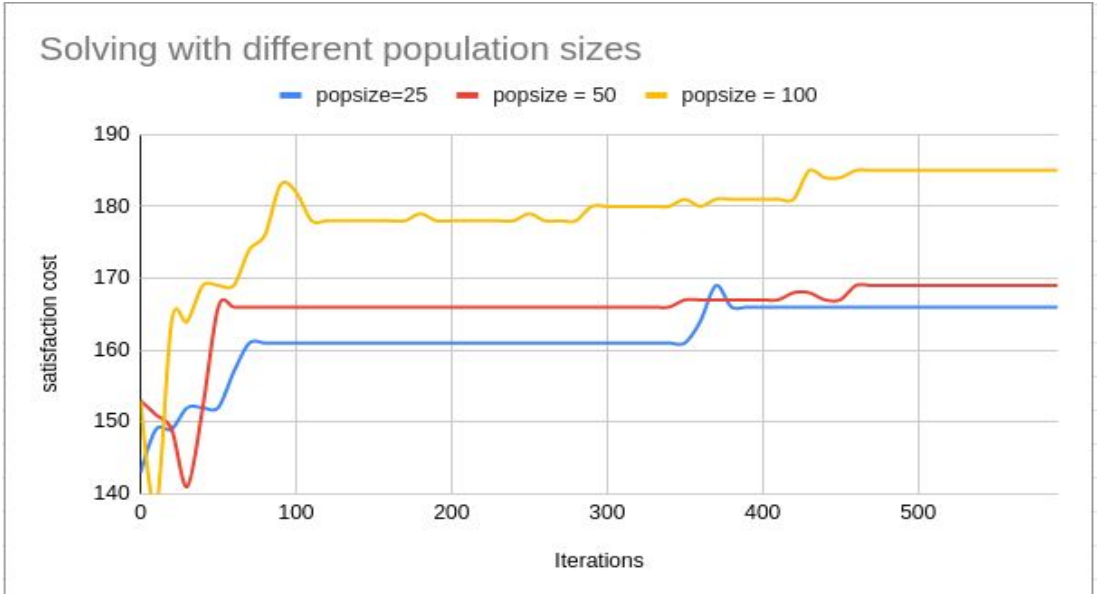
TSPLIB: ftv64
Optimal Cost: 1945
(150 Popsize, 25 Elitesize, 0.002 Mutate)



TSPLIB: rbg407
Optimal Cost: 4619
(150 Popsize, 25 Elitesize, 0.002 Mutrate)

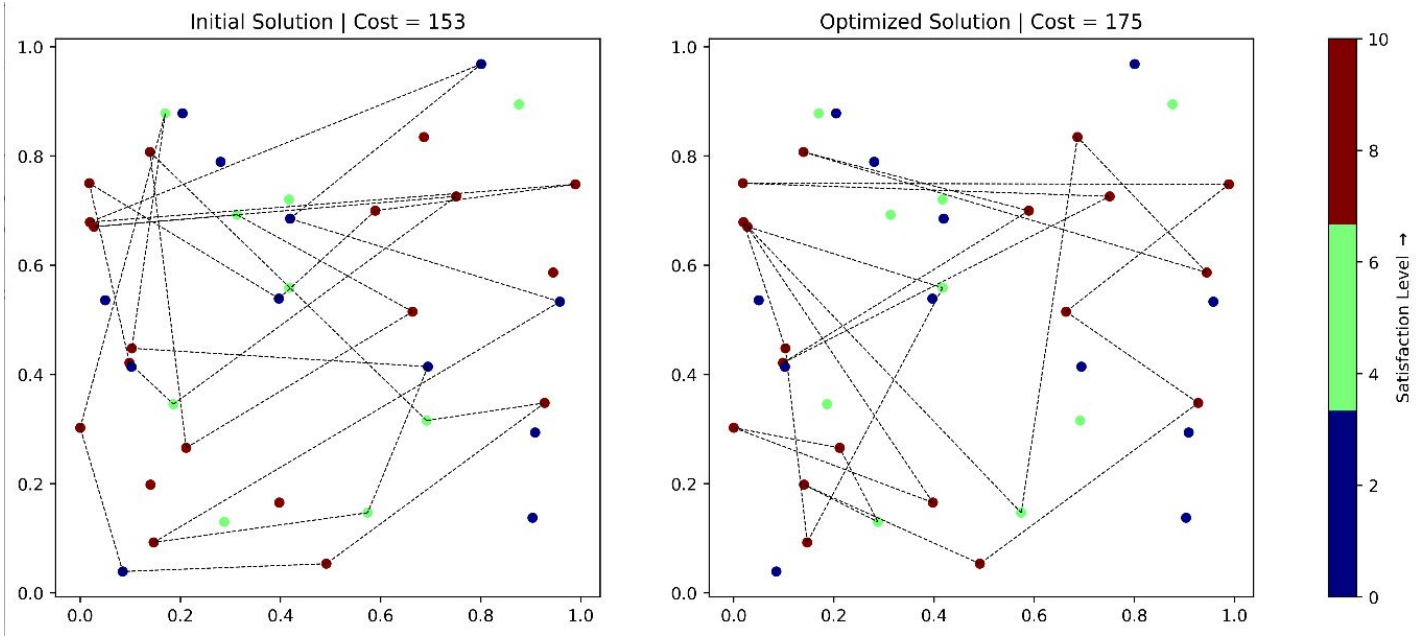
Genetic Algorithm

- Problem 3 - Effect of population size



Genetic Algorithm

- Problem 3 - T_{max} = 20 | N = 40



Simulated Annealing

Simulated Annealing

- **Penalized Objective Function**
  - ‘T’ includes time taken in travelling and the time spent at each exhibit
  - ‘F’ gets activated on violation of constraint
  - Objective function for constrained optimization with penalty coefficient ‘λ’

$$T = \sum_i \sum_j \left(\frac{c_{ij}}{v} + \tau \right) y_{ij}$$

$$F = \begin{Bmatrix} 0, & \text{for } T \leq T_{max} \\ e^{T - T_{max}}, & \text{for } T > T_{max} \end{Bmatrix}$$

$$\max \left(\sum_{i=1}^{n} s_i \sum_j y_{ij} - \lambda(T + F) \right)$$

Simulated Annealing

- Problem 3 - Satisfaction Index
  - Solution Representation

Nodes being visited: `[1, 2, 3, 4, 5]`
Nodes not visited: `[9, 8, 7]`
Representation: `[1, 2, 3, 4, 5 | 9, 8, 7]`
Actual input:

- Single Array
- Location of bar

`nodes=[1, 2, 3, 4, 5, 9, 8, 7]`
`loc_bar=5` *(= Number of elements in array of nodes being visited)*

Simulated Annealing

- Problem 3 - Satisfaction Index
  - Functions:

    - **`ApplySwap()`**: Swap 2 elements in the array of nodes being visited

      Eg: `nodes=[1, 2, 3, 4, 5 | 9, 8, 7] → nodes=[4, 2, 3, 1, 5 | 9, 8, 7]`
    - **`ApplyShake():`** Swap 1 element from the array of nodes being visited and 1 from the array of nodes not being visited

      Eg: `nodes=[1, 2, 3, 4, 5 | 9, 8, 7] → nodes=[1, 2, 7, 4, 5 | 9, 8, 3]`
    - **`ModifyNodes():`** Change location of bar (Increase or decrease number of nodes being visited), i.e., (+1/-1 with probability k)

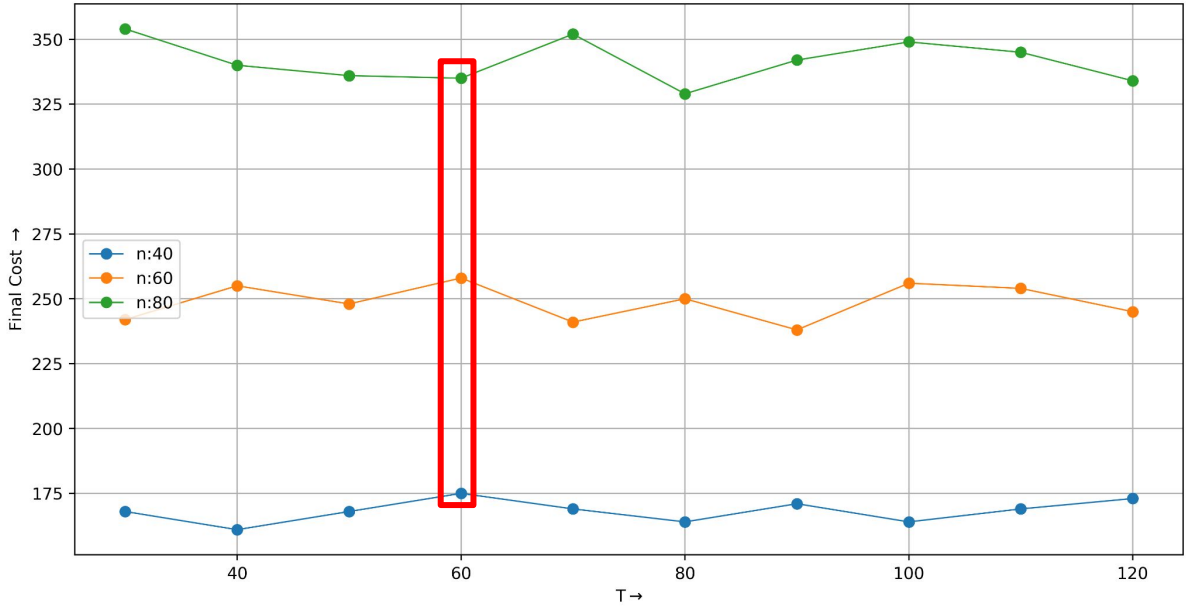
      Eg: `loc_bar=5 → loc_bar=6`

Simulated Annealing

Parameter Tuning

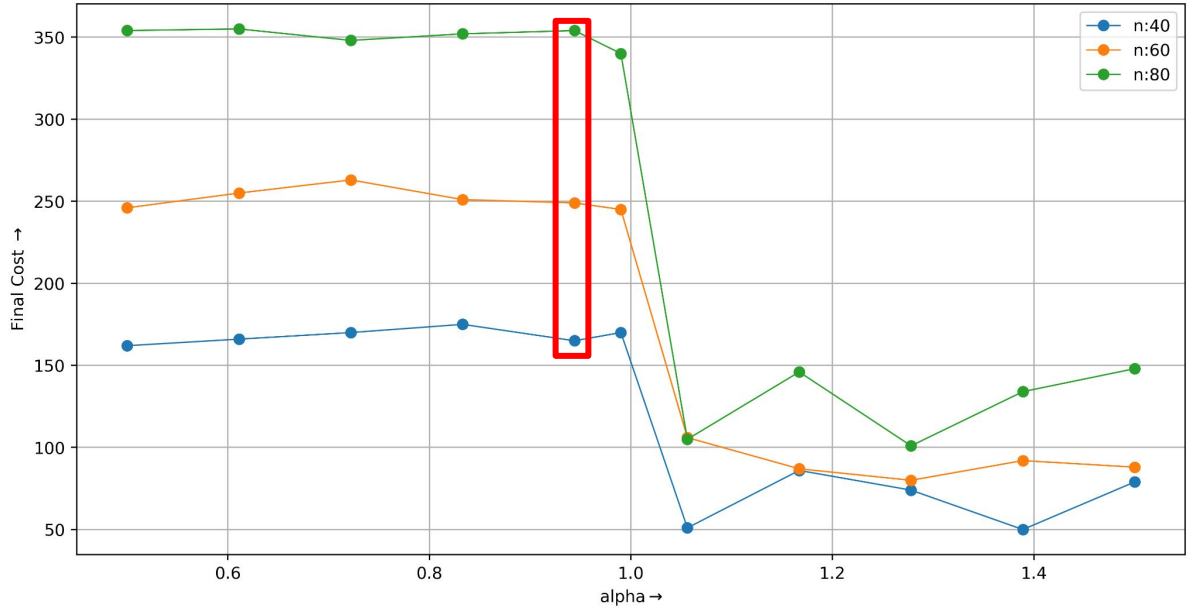
Simulated Annealing

- **Initial Temperature (T)** : Tuned value = 60



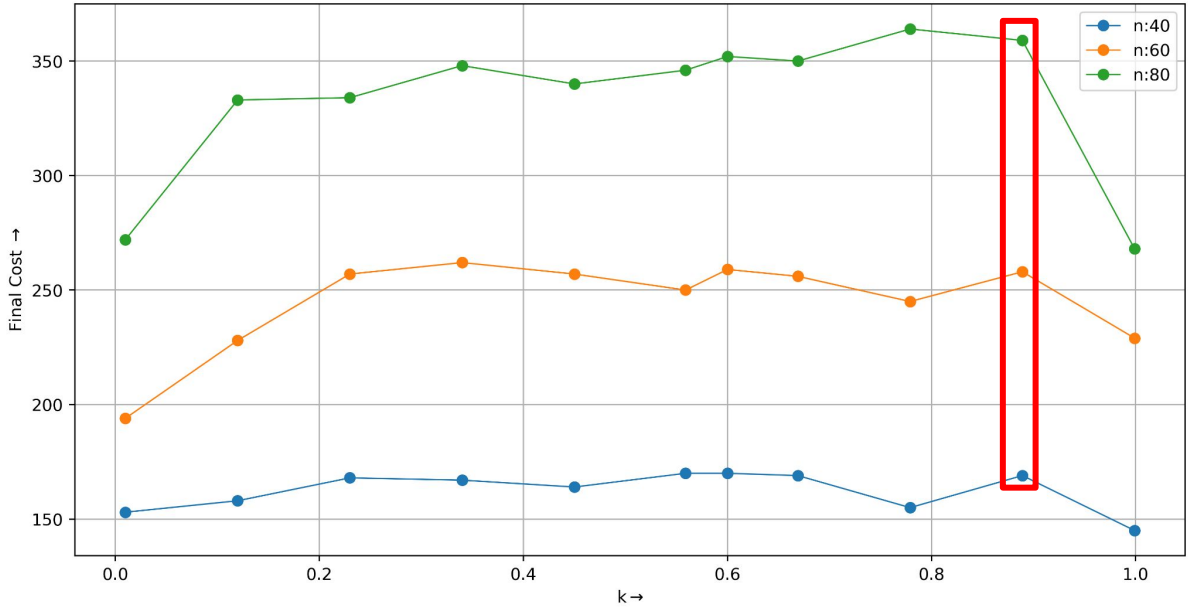
Simulated Annealing

- **Cooling Factor (α)** : Tuned value = 0.944



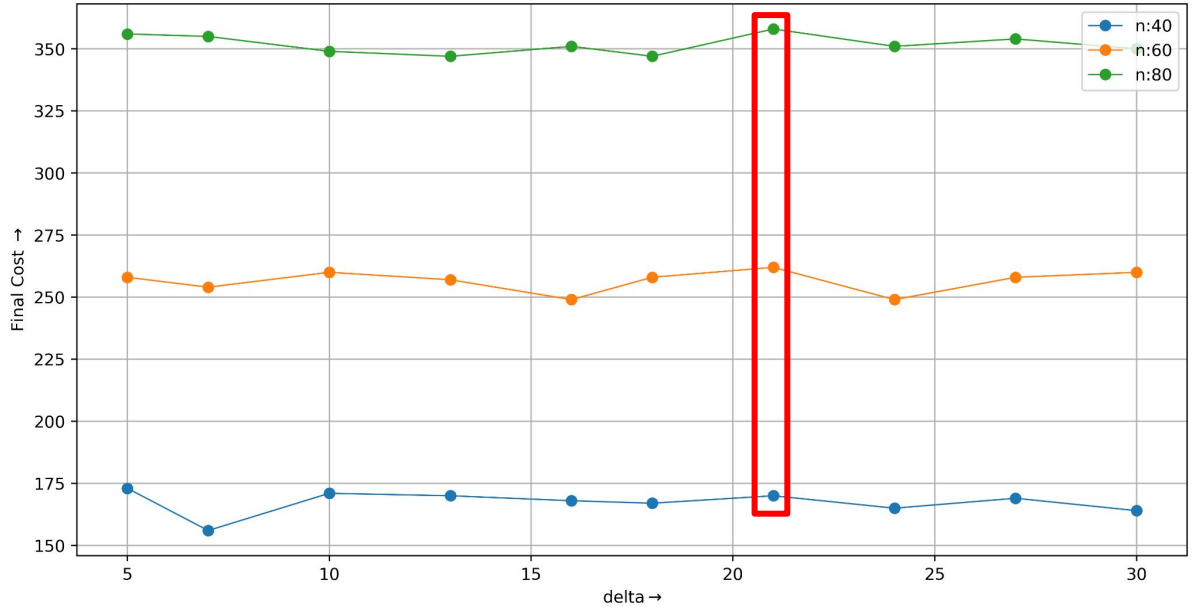
Simulated Annealing

- **Probability of increasing visited exhibits (k)** : Tuned value = 0.889



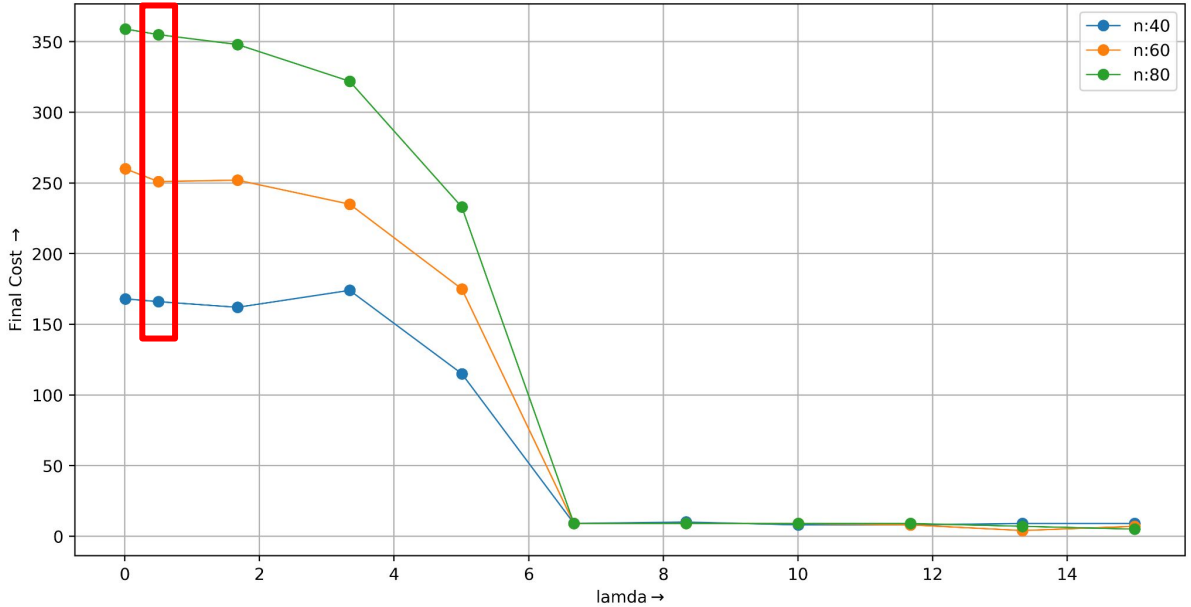
Simulated Annealing

- **# of iterations after which solution is shaken (δ)** : Tuned value = 21



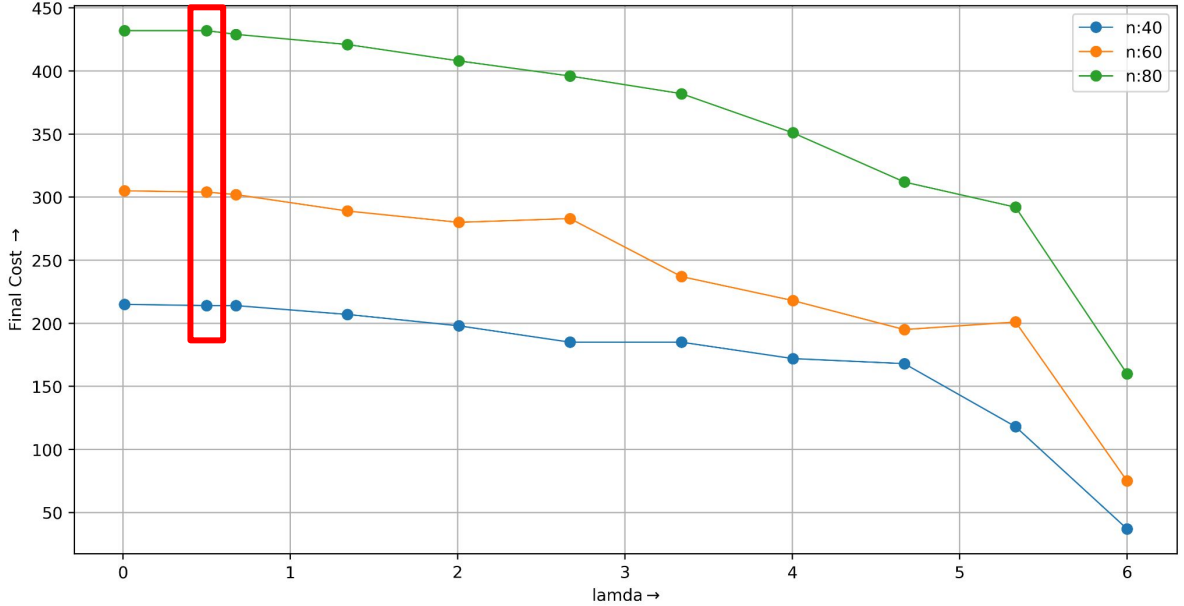
Simulated Annealing

- **Penalty coefficient (λ) - Constrained** : Tuned value = 0.5



Simulated Annealing

- **Penalty coefficient (λ) - Unconstrained** : Tuned value = 0.5

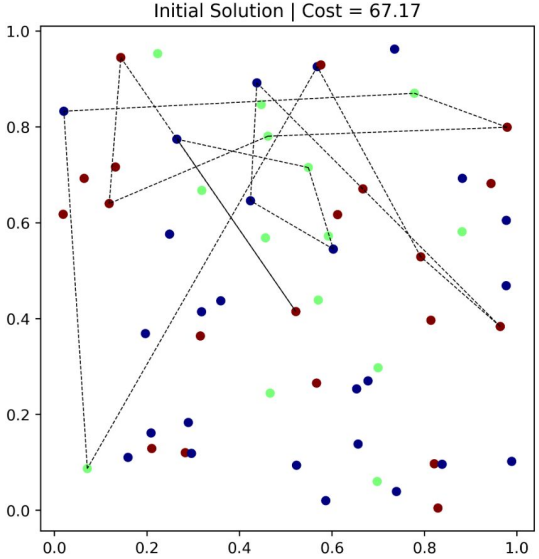


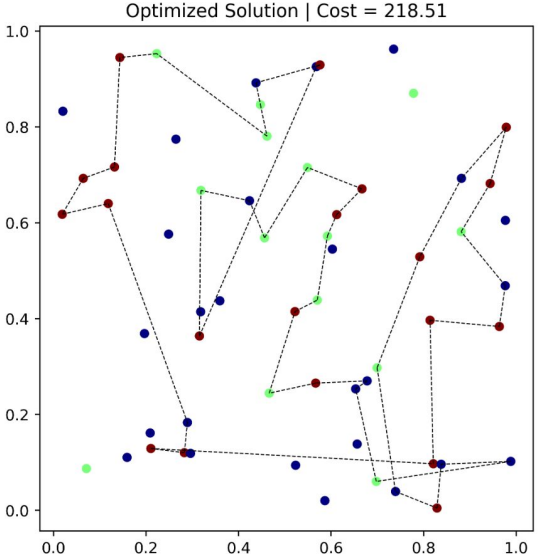
Simulated Annealing

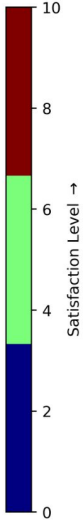
- **Observations**
  - **Concerns**
    - Purely unconstrained problem:
      - Travel path → Mess!
    - If $\mathbf{T_{max}}$ **> 0.5(# of exhibits)** → All exhibits are visited → Messy solution!
  - **Modification**
    1. Run algorithm to optimise the cost function **- Unconstrained**
    2. Run simulated annealing algorithm again - **Minimise only travel distance**
       a. Swap two randomly picked exhibits → **With 0.75 probability**
       b. Swap two randomly picked consecutive exhibits → **With 0.25 probability**
  - **Conclusion**
    - Both concerns are satisfied!

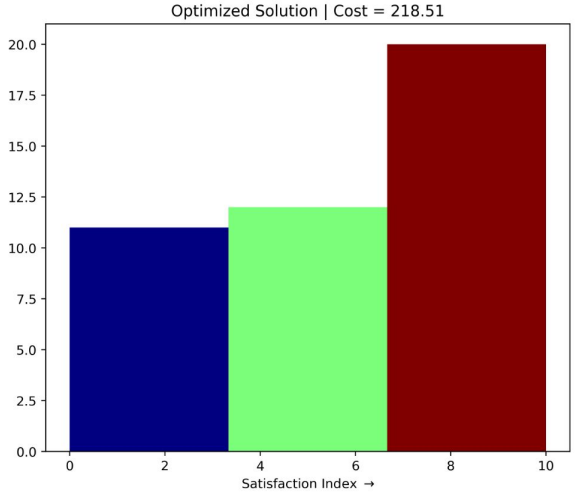
Simulated Annealing

- **Penalized Cost Function + Constrained :**
  - (Total travel time = 18.0 | T_{max}= 18.0 | RT = 32.61 s | N = 60)



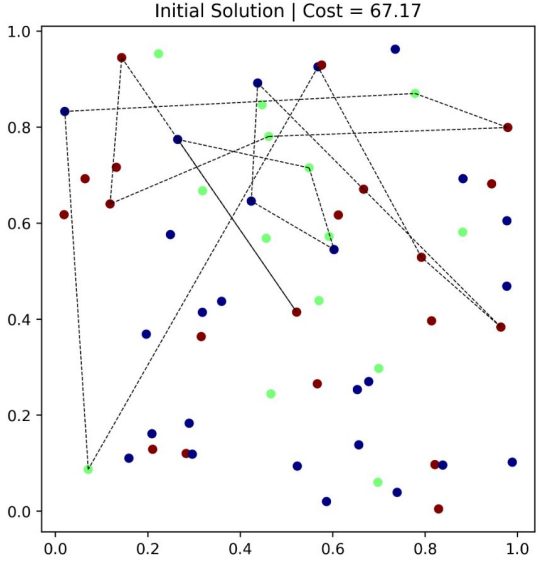


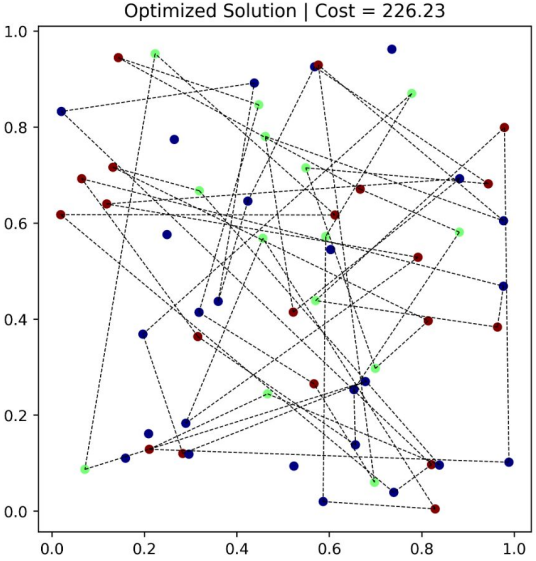


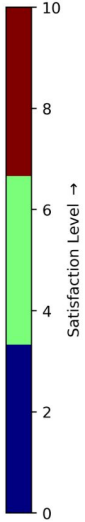


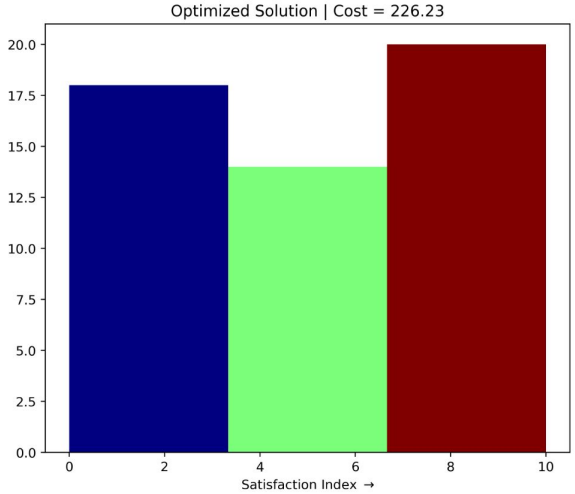
Simulated Annealing

- **Penalized Cost Function + Constrained :**
  - (Total travel time = 39.5 | T_{max}= 40 | RT = 48.1 s | N = 60)



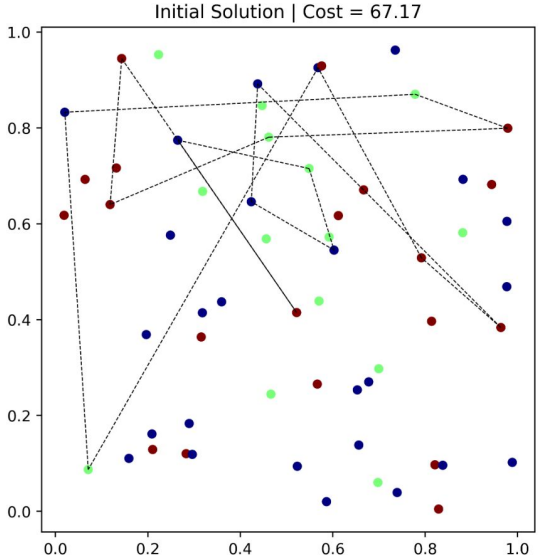


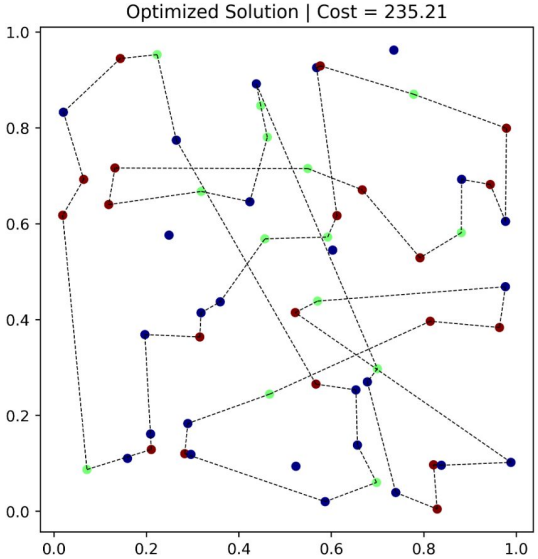


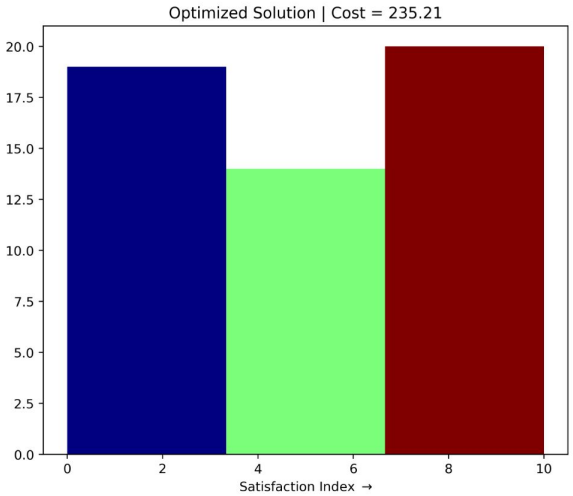


Simulated Annealing

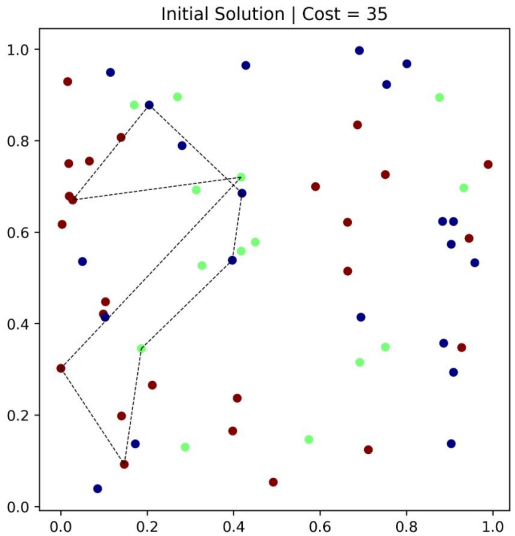
- **Penalized Cost Function + Unconstrained + Distance Minimization :**
  - (Total travel time = 22.6 | T_{max}= 40 | RT = 25.8 s | N = 60)

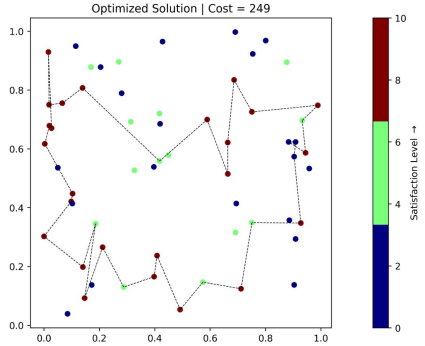




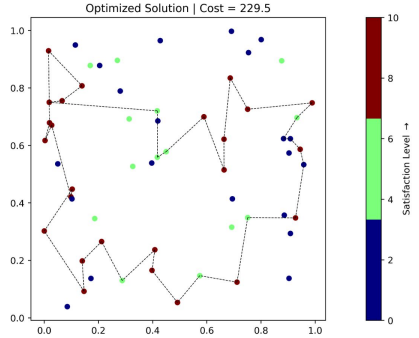


Comparing Implementations

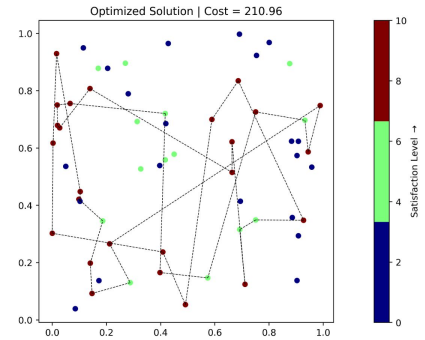




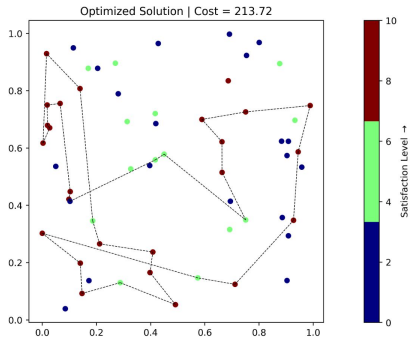
Constrained Optimization



Penalized Cost Function + Constrained



Penalized Cost Function + Unconstrained



Penalized Cost Function + Unconstrained + Distance Minimization

Comparing Implementations

N=60 , T_max = 14

	Constrained Optimization	Penalized Cost Function + Constrained	Penalized Cost Function + Unconstrained	Penalized Cost Function + Unconstrained + Distance Minimization
Run time	8.726	11.762	5.12	9.645
Number of cost function calls	110901	110063	100003	100003
Number of constraint function calls	200970	200968	1001	1001
Exhibits visited	34	35	33	31
Travel Time	13.9	14	17	13.6
Final Satisfaction	249	254	246	236

Performance Metrics

Performance Metrics

- Once the algorithms are implemented they can be compared using performance metrics
- The performance metrics to be used are Efficiency, Reliability and Quality of Solution
- Efficiency involves measuring number of fundamental evaluations, running time and memory consumption of the algorithms
- Reliability can be checked by considering aspects like success rate, number of constraint violations, if any, and effect of starting point choice.
- Quality of Solution is determined by measuring computational accuracy, time taken to find optimal value within a fixed range of the solution or accuracy of the solution after running for a certain number of iterations or a certain time.

Benchmark Cases - Symmetric

Test Case	Run Time (in sec)			
	Simulated Annealing	Ant Colony	Genetic (On-Cloud)	Branch and Bound
burma14	3.8	1.9	9	19.8
att48	7.4	5.4	22	$\longrightarrow \infty$ 😢
berlin52	7.6	6	21	
bier127	14.9	15.2	38	
brg180	19.9	24.3	56	

Benchmark Cases - Symmetric

Test Case	Final Cost				
	Simulated Annealing	Ant Colony	Genetic	Branch and Bound	Expected Optimum
burma14	3323	3323	3323	3323	*3323*
att48	11992	12600	13611	Hard Luck! 😢	*10628*
berlin52	9465	8500	9626		*7542*
bier127	180727	141513	343260		*118282*
brg180	4350	4690	315243		*1950*

Benchmark Cases - Asymmetric

Test Case	Run Time (in sec)		
	Simulated Annealing	Ant Colony	Genetic (On-Cloud)
br17	4.0	1.9	4
ft53	7.9	6.2	27
ftv64	9.1	8.5	25
ft70	10.0	9.2	29
rbg403	45.9	167.5	98

Benchmark Cases - Asymmetric

Test Case	Final Cost			
	Simulated Annealing	Ant Colony	Genetic	Expected Optimum
br17	39	89	39	*39*
ft53	8630	8969	10003	*6905*
ftv64	2436	2233	3395	*1839*
ft70	44497	45384	47147	*1950*
rbg403	3137	5179	4853	*2465*

Problem Statement - 3

# of Nodes	Initial Cost	Final Cost	
		Simulated Annealing	Genetic
40	27	214	185
60	35	305	247
80	129	431	358

Case (seed) = 1

Conclusion

Conclusion

Why do different algorithms lead to different optimal points?

- It is because heuristics algorithms involve random numbers to generate a solution and it does so in different ways. This randomness leads to the same algorithm not giving same optima over multiple runs

What are the main takeaways from the project?

- Different Variants of the problem statements can be solved
- Exact algorithms vs Heuristic algorithms
- Genetic Algorithm: O(g(nm + nm + n)) with g the number of generations, n the population size and m the size of nodes, this agrees with our results

Are all design variables important?

- Yes, all design variables are important in our case

Conclusion

If you had an opportunity to redo the project, what changes would you make?

- Would not have chosen Python - initialisation of variables is slow, it requires high memory allocation
- Use methods to speed up the computation like parallel processing

What recommendations, if any, would you provide to a potential user who is interested in using the algorithms for their purpose?

- Understand the nature of problem
  - Algorithms' implementations are problem statement specific
- Spend reasonable time and resources on parameter tuning

Thank You!

We would like to thank Prof. Abhijit Gogulapati for having given us this wonderful opportunity to work on this project, and for his invaluable inputs.

We would also like express our gratitude to Prof. Rajkumar S. Pant, for his insights on optimization algorithms, which proved to be of immense help in our tasks.

Finally, we would like to thank the teaching assistants, for their constant support!

Appendix

List of Symbols

$$y_{ij} = \left\{ \begin{array}{l} 1, \text{If path goes from exhibit } i \text{ to exhibit } j \\ 0, \text{otherwise} \end{array} \right\}$$

$$c_{ij} := \text{Penalty incurred for going from } i \text{ to } j$$

$$c_{ij} > 0$$

$$G := (V, \mathbb{E})$$

$$V := \text{Vertices}$$

$$\mathbb{E} := \{(x, y) | x, y \in V\}$$

$$S := \text{Set of all tours of } G$$

$$E := \text{Subset of permitted paths in } \mathbb{E}$$

$$E' := \text{Complement of E}$$

$$v = \text{Walking speed}$$

$$\tau = \text{Time spent at each exhibit}$$

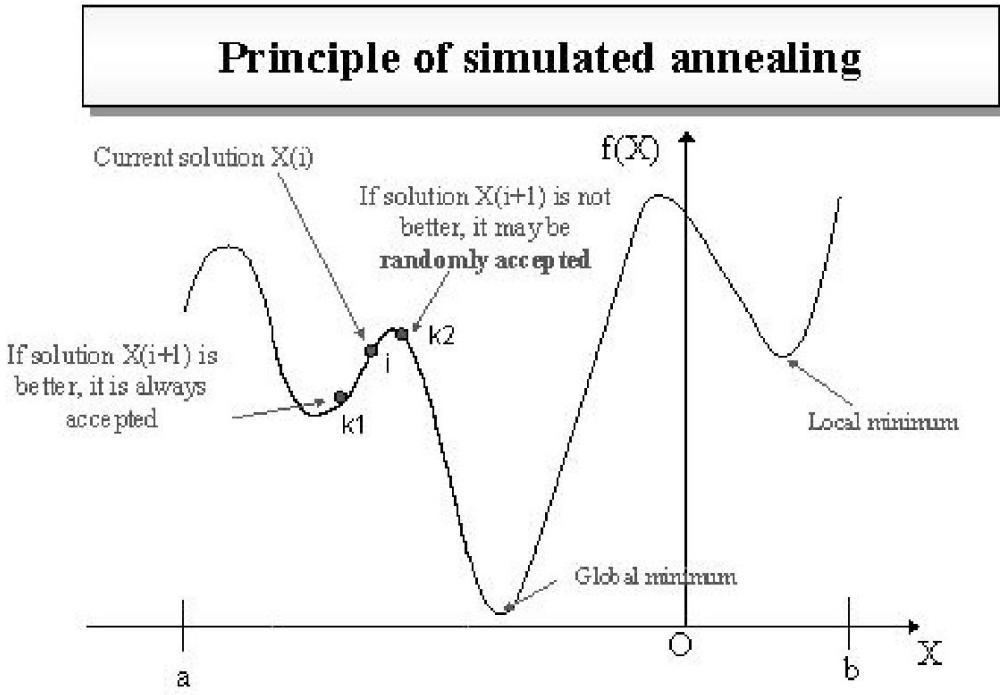
$$T_o = \text{Total available time}$$

Pseudo Codes

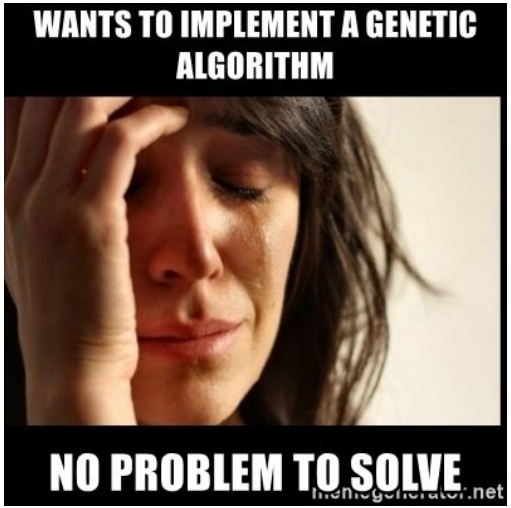
Simulated Annealing

Link to Pseudo Code.

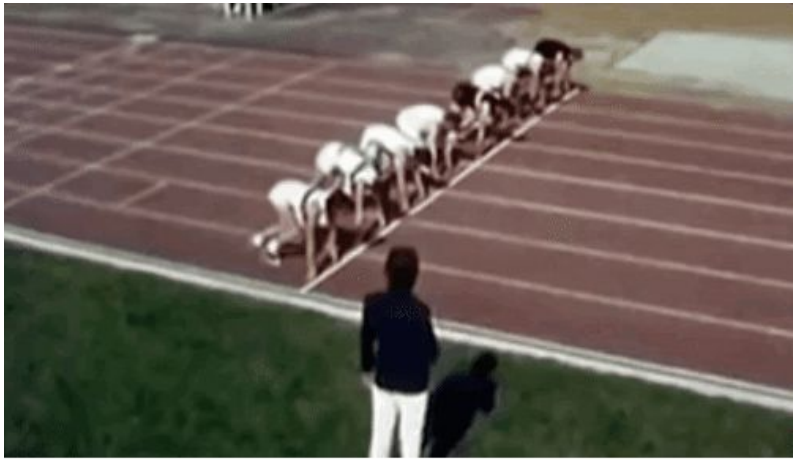
$$S_c = \{[1, 8, 2, 5], [9, 7, 10]\}$$
$$QueueList = [3, 4, 6]$$



Genetic

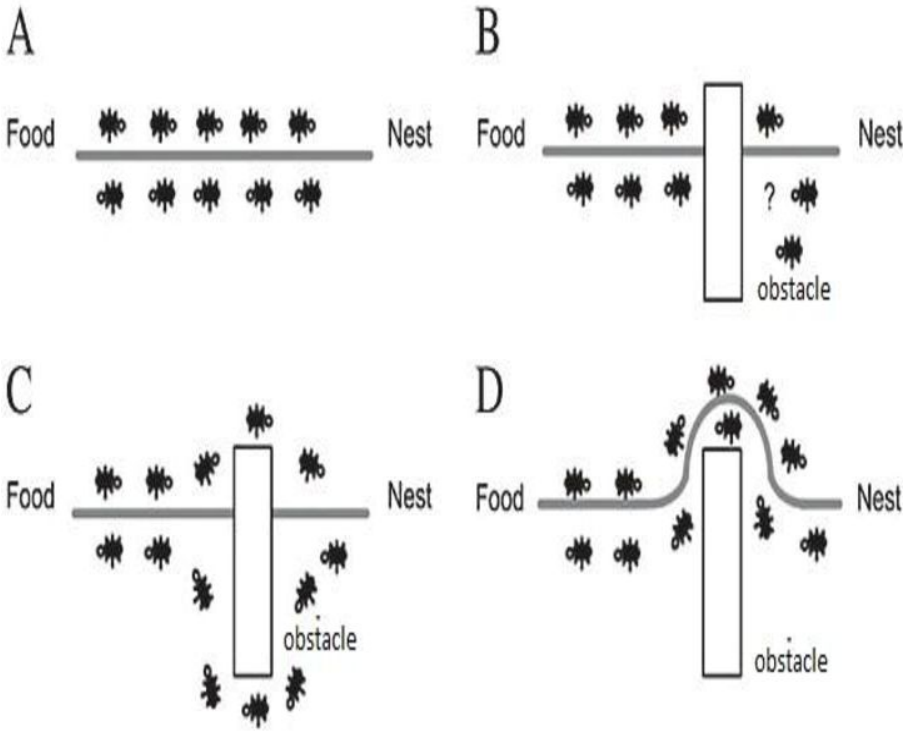


Link to Pseudo Code.



When you watch the first generation of your genetic algorithm

Ant Colony



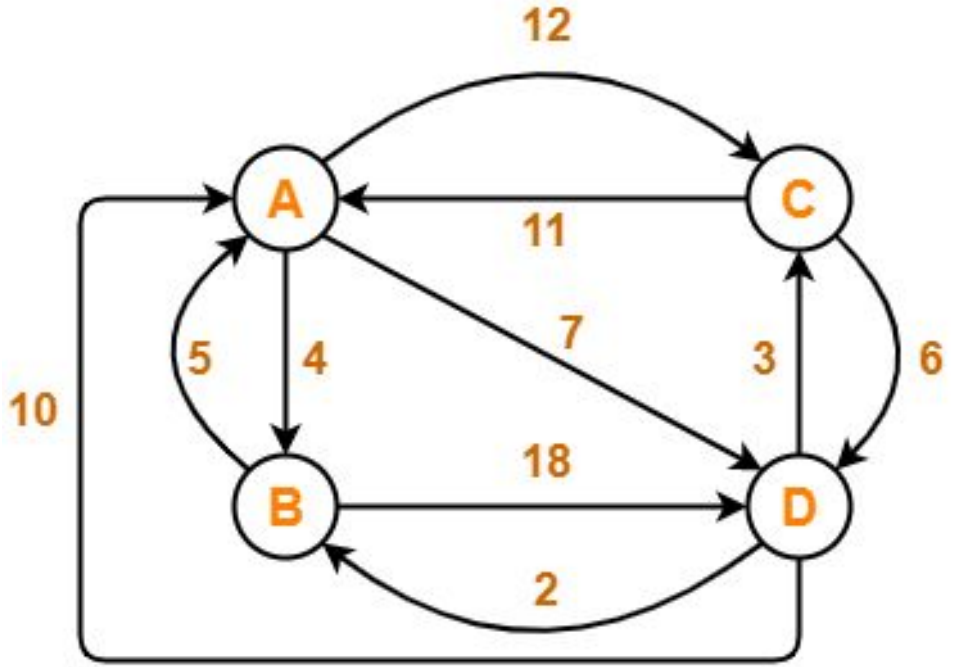
Link to Pseudo Code.

Plan of Action - Analysis

- Solve each problem separately using all the selected algorithms to understand the nuances of the problem and search algorithms
- Use Branch-and-Bound algorithms to obtain the global optimal solution.
- Compare the results with available literature and TSP libraries
- Solve the complete problem statement with all of the complexities

Branch-and-Bound

Link to Pseudo Code.

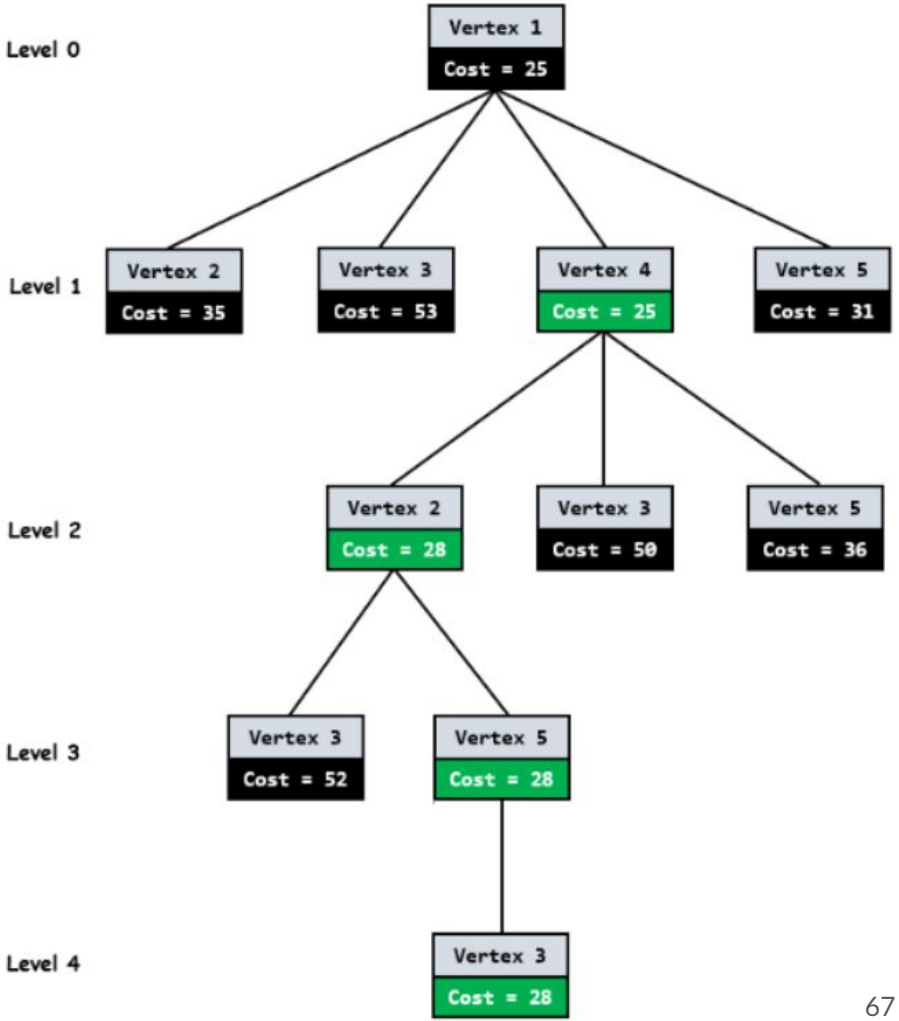


Matrix Reduction

INF	20	30	10	11
15	INF	16	4	2
3	5	INF	2	4
19	6	18	INF	3
16	4	7	16	INF

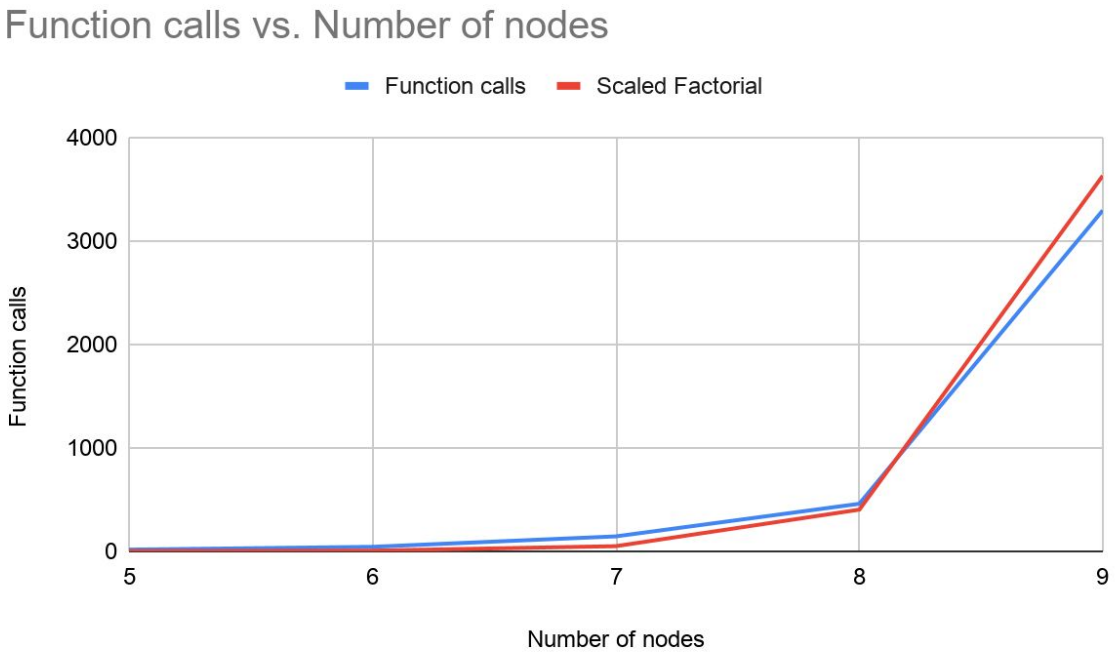
INF	10	20	0	1
13	INF	14	2	0
1	3	INF	0	2
16	3	15	INF	0
12	0	3	12	INF

INF	10	17	0	1
12	INF	11	2	0
0	3	INF	0	2
15	3	12	INF	0
11	0	0	12	INF



Branch and Bound computational cost

Number of nodes	Function calls	Scaled Factorial
5	17	1.68
6	44	7.2
7	145	50.4
8	460	403.2
9	3294	3628.8



References

References

Optimal walk around path in a museum to view all exhibits

- Optimal Museum Traversal Using Graph Theory
  - Explains basics of Hamiltonian path
- Warehouse Optimization - Algorithms For Picking Path Optimization
  - Gives a brief about all kinds of algorithms which can be employed for Path Optimization

Travelling Salesman Problem (TSP)

- Travelling salesman problem - Wikipedia
  - Explains the problem, formulations and constraints
  - Talks about the different algorithms as well
- Travelling Salesman Problem | Set 1 (Naive and Dynamic Programming)
  - There a lot of implementations of different algorithms for solving TSP in the Related Articles section
- Chapter 10 The Traveling Salesman Problem
  - Detailed paper talking about various solutions to TSP, and their analysis and their time complexity
- Particle swarm optimization for traveling salesman problem
- Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches

References

Problem Formulation

- Travelling Salesman Formulation
  - Traveling salesman problems
- School Bus Problem - This paper has a lot of variants
  - https://dspace.mit.edu/bitstream/handle/1721.1/5363/OR-078-78.pdf?sequence=1&isAllowed=y
- An Exact Algorithm for the Time-Constrained Traveling Salesman Problem
  - https://scihub.wikicn.top/10.1287/opre.31.5.938

Algorithms

- Dynamic Programming, Simulated Annealing, and 2-opt:
  - How to Solve Traveling Salesman Problem — A Comparative Analysis
- Genetic Algorithm, Simulated Annealing, Particle Swarm Optimization, Ant Colony Optimization, Bacteria Foraging Optimization, and Bee Colony Optimization
  - (PDF) Optimization Techniques for Solving Travelling Salesman Problem

References

Genetic Algorithm

- A Genetic Algorithm for Solving Travelling Salesman Problem
- A genetic algorithm for the orienteering problem - IEEE Conference Publication
- A genetic algorithm to design touristic routes in a bike sharing system | Request PDF

Simulated Annealing

- A simulated annealing heuristic for the team orienteering problem with time windows
- A simulated annealing heuristic for the multiconstraint team orienteering problem with multiple time windows
- Solving tourist trip planning problem via a simulated annealing algorithm

Ant Colony Algorithm

- (PDF) An ant colony approach to the orienteering problem
- Ant colony approach to the orienteering problem

Branch-and-Bound

- A Proposed Solution to Travelling Salesman Problem using Branch and Bound
- Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches
- Branch and Bound for TSP

TSPLIB Test Cases

- TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types.
- TSPLIB has test case problems of Symmetric and Asymmetric TSP Problems, with varying number of nodes, which will be used for performance evaluation of the implemented algorithms
- The data provided in TSBLIB includes coordinates of nodes or costs between the nodes and the best found cost for each test case. For some test cases the optimal path is also given.