AE - 755

Team Sic Mundus

Table of Contents

Resources

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

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
- Survey on Tourist trip design problems
    - https://www.researchgate.net/publication/271921760_A_survey_on_algorithmic_approaches_for_solving_tourist_trip_design_problems

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

Summary

Consider this reading as a basic to approach the problem

- MANAGING CROWDED MUSEUMS: VISITORS FLOW MEASUREMENT, ANALYSIS, MODELLING, AND OPTIMIZATION (We need to take the ideas from this paper to imagine the framework of the progress)

Few Takeaway Points:

- Our target problem statement: Managing a crowded museum with the visitors flow measures to optimize the time required to visit the exhibits. Can optimize the visitor flow also. Everything will depend on how much complexity we can handle.
- The problem requires a mixed approach of **Travelling Salesman Problem and Pedestrian Flow Model.** So it will be better if everyone has some idea about these two approaches.

Set of Variables that can be incorporated in the framework of the problem

- Average time spent for an exhibit
- Amount of people stopped for an exhibit
- Number of stops
- Use of stairs/escalators/lift
- Length of visits
- Movement pattern
- Busy hours
- Limit of maximum occupancy
- Regulate the entrance flows
- Setting a maximal duration
- Ticket price
- Festive Season

We can add or neglect a few variables according to our progress and time/resource constraints.

- We can think of this approach
    - A route navigation algorithm for pedestrian simulation based on grid potential field
- This is a general discussion on pedestrian flow model
    - Pedestrian Flow Models
- This is the kind of example we need to implement in our optimization
    - Analyzing and optimizing pedestrian flow through a topological network based on M/G/C/C and network flow approaches

- A MESHFREE PARTICLE METHOD FOR A VISION-BASED MICROSCOPIC PEDESTRIAN MODEL. This includes mathematical formulation of a problem
    - A MESHFREE PARTICLE METHOD FOR A VISION-BASED MICROSCOPIC PEDESTRIAN MODEL
- Design and analysis of control strategies for pedestrian flows
        - Control Strategies
- Modeling and Simulation of Macroscopic Pedestrian Flow Model
    - Modelling and Simulation of the Pedestrian Flow model

- This is the most relevant for our problem
    - Route optimization model for pedestrian evacuation in metro hubs
- Comparison of Algorithms for Solving Traveling Salesman Problem
    - https://www.ijeat.org/wp-content/uploads/papers/v4i6/F4173084615.pdf

Stage I

Deadline

4th February, Thursday - 11:30 AM

Requirements

- Verbal statement of the problem
    - Objective, constraints, and design variables
- Mathematical expressions for desired or preferred objective function and constraints
- Choice of search algorithms to implement
- You need to discuss what a realistic or practical problem is, and the thought process behind how you simplified it to a mathematical formulation within the scope of the current course.

Presentation

Link: Delta Optimization

1. Mention the end goal - verbal problem statement, design variables, constraints
2. Mention the sample of simple problems we would solve firstly to understand
    a. Each problem statement can have different objective functions, design variable, constraints, etc
        i. Mathematical formulation for each case
        ii. Mention the search algorithm in each case
    b. But they all need to be linear and integer
3. Our justification for way we have broken down the end goal using the simpler problem statements mentioned earlier
4. Talk about how we plan on using mixed optimization to the end

Problem Statement

End Goal

Optimize the route for a tourist visiting a crowd-less Louvre Museum, such that the satisfaction index is maximised by visiting all/select exhibits in a single working day.
The museum has 4 floors and both the entry and exit are at the same level.

Design variables

1. Time spent between consecutive exhibits
    a. (this would be a function of the crowd flow as well)
2. Time spent on an exhibit
3. Length of path travelled
4. Number of times stairs/escalator used

    a. Each would have its own pros and cons - velocity, waiting time, etc
5. Number of times elevator used
6. State of a route being one-way

- Don't prefer mixed integer problems, stick with real numbers.
- Keep it a linear problem for integer, restructure the problem if required to avoid mixed/nonlinearity in the beginning

Simple Travelling Salesman Problem Statements

For each problem find:

1. Objective Function
2. Constraints
3. Design Variables
4. Search Algorithms
5. How this is relevant to our end goal

Make sure there's no overlap between problem statements.

NOTE: SUBTOUR ELIMINATION CONSTRAINT UNDERSTANDING:
https://medium.com/swlh/techniques-for-subtour-elimination-in-traveling-salesman-problem-theory-and-implementation-in-71942e0baf0c

Problem 1

Traveling salesman problems

aTSP ILP Formulation

The integer linear programming formulation for an aTSP is given by

$$\min \quad \sum_i \sum_j c_{ij} y_{ij}$$

$$\text{s.t} \quad \sum_j y_{ij} = 1, \quad i = 0, 1, ..., n-1$$

$$\sum_i y_{ij} = 1, \quad j = 0, 1, ..., n-1$$

$$\sum_i \sum_j y_{ij} \leq |S| - 1 \quad S \subset V, 2 \leq |S| \leq n-2$$

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

Problem 2

Problem statement: School bus problem

Optimize the route travelled by visitor to minimize total time to visit all exhibits, by considering the time at each exhibit and travel time

https://www.researchgate.net/publication/245281062_The_school_bus_routing_problem_A_case_study

1. Objective Function

(3) Minimise the total bus travel time. This is another objective related to the service cost. It consists of two parts: minimising the total bus loaded travel time (from route origins to the school) and minimising the total bus vacant travel (deadheading) time (from parking places to the route origins).

The former can be formulated as

$$\min \sum_{k=1}^{K}\left[\sum_{i=1}^{n}\left(\sum_{j=1}^{n+1} t_{ij}x_{ijk} + Lz_{ik}\right)\right]$$

A solution resulting in the improvement of objective (2) usually leads to the improvement of this part of objective

2. Constraints

Constraints of the mathematical formulation

Assume that the school is the dummy starting point 0 for every route, and let $t_{0j} = 0$ $(j = 1, 2, \ldots, n)$, then the constraints of the problem can be formulated as

$$\sum_{i=1}^{n} y_{ik} \leqslant C_k \qquad (k = 1, 2, \ldots, K) \tag{1}$$

$$\sum_{k=1}^{K} y_{ik} = f_i \qquad (i = 1, 2, \ldots, n) \tag{2}$$

$$\sum_{i=0}^{n} x_{ilk} = \sum_{j=1}^{n+1} x_{ljk} \qquad (l = 1, 2, \ldots, n;\ k = 1, 2, \ldots, K) \tag{3}$$

$$\sum_{i=1}^{n} x_{i(n+1)k} = \sum_{j=1}^{n} x_{0jk} = 1 \qquad (k = 1, 2, \ldots, K) \tag{4}$$

$$\sum_{j=1}^{n+1} x_{ijk} \geqslant z_{ik} \qquad (i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, K) \tag{5}$$

$$y_{ik} \leqslant f_i z_{ik} \qquad (i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, K) \tag{6}$$

$$x_{ijk}, z_{ik} = 0 \text{ or } 1 \tag{7}$$

$$y_{ik} \geqslant 0 \text{ and an integer} \tag{8}$$

Constraint (1) is the bus capacity constraint. Constraint (2) ensures that all pupils are picked up. Constraint (3) requires that a bus visiting a point also leaves that point. Constraint (4) ensures that all buses visit the school, and Constraint (5) ensures that if a bus picks up pupils at a point, it must also visit that point. Constraint (6) ensures that if any pupil is picked up at point i by bus k, z_{ik} will be set to 1.

3. Design Variables

n is the total number of pick-up points.
M is the total number of pupils to be served.
$p_1, p_2, \ldots, p_n$, are the n pick-up points. These are ordered by decreasing distance from the school. p_{n+1} denotes the school.
t_{ij} is the travel time from p_i to p_j.
f_i is the number of pupils to be picked up at p_i.
L is the average pick-up time at pick-up points.

$$x_{ijk} = \begin{cases} 1 & \text{if bus } k \text{ travels directly from } p_i \text{ to } p_j \\ 0 & \text{otherwise.} \end{cases}$$

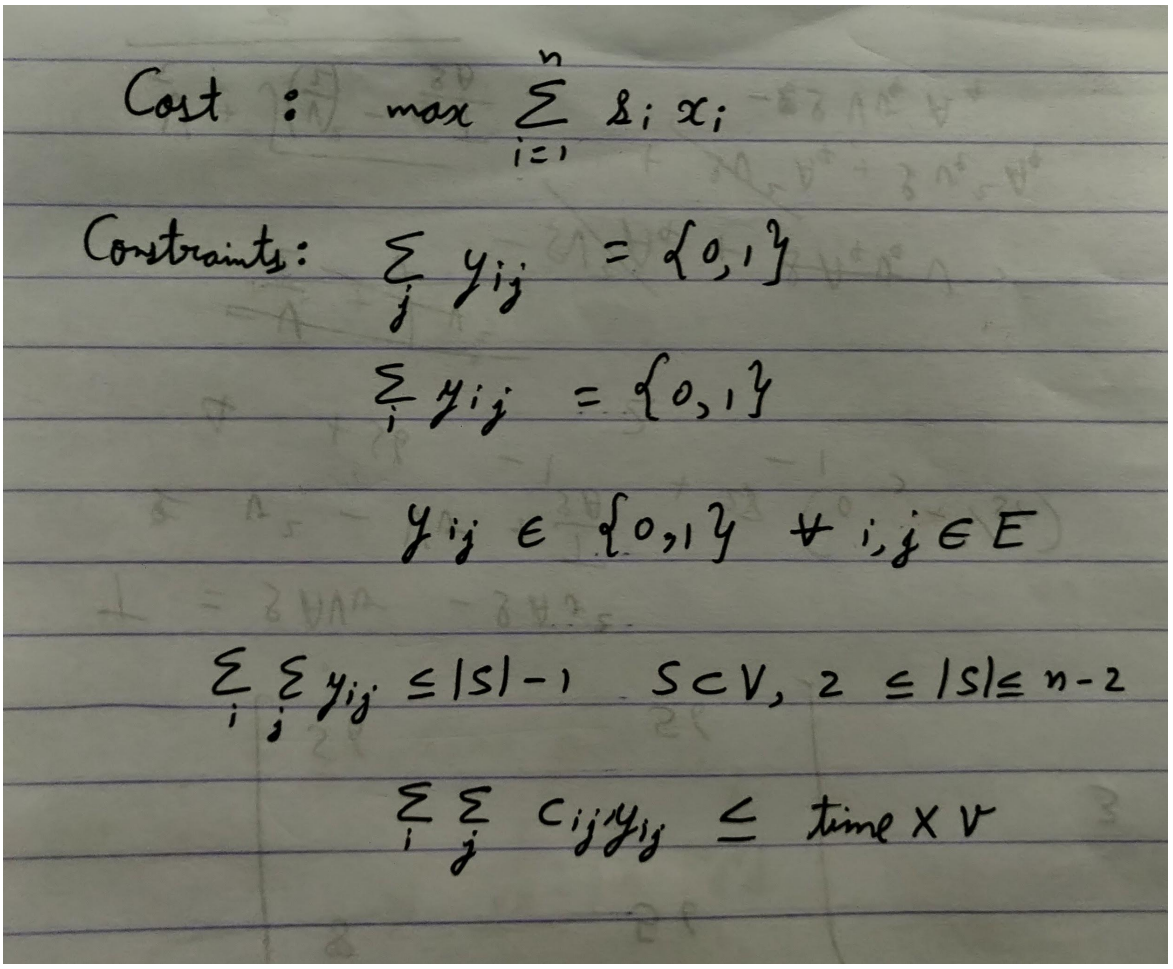
$$z_{ik} = \begin{cases} 1 & \text{if bus } k \text{ picks up pupils at } p_i \\ 0 & \text{otherwise.} \end{cases}$$

y_{ik} is the number of pupils picked up by bus k at p_i.

4. Search Algorithms
5. How this is relevant to our end goal
   consider only 1 bus, and the notion of bus is replaced by visitor, bus stops replaced by exhibits, time taken at each exhibit can be assumed average time.
   Here we get to understand about the main problem in simplest version by considering visit points, average time spent at the points and travel time only

Problem 3

1. All exhibits are not interconnected (or one way)
   a. y_{ij} = {0,1} for some {i,j}s (ones which are connected)and y_{ij} = y_{ji} = 0 for other {i,j}s (ones which are not connected)
2. Maximise satisfaction index in limited time
   a. Have a set of parameters ‘s_i’ which are satisfaction index for each exhibit and ‘v’ is the velocity of tourist
   b. Cost function :



3. ~~Multiple policemen/security guard~~

   ~~Ref.: Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches~~

3.3.1 Assignment-based integer programming formulations

The mTSP is usually formulated using an assignment based double-index integer linear programming formulation. We first define the following binary variable:

$$x_{ij} = \begin{cases} 1 \\ 0 \end{cases} \text{ If arc (i, j) is used in the tour,}$$

Otherwise.

Then, a general scheme of the assignment-based directed integer linear programming formulation of the mTSP can be given as follows:

Minimize

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Subject to

$$\sum_{j=2}^{n} x_{1j} = m \qquad (11)$$

$$\sum_{j=2}^{n} x_{j1} = m \qquad (12)$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 2,\ldots\ldots,n \qquad (13)$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 2,\ldots\ldots,n \qquad (14)$$

$$+ \text{ subtour elimination constraints,} \qquad (15)$$

$$x_{ij} \in \{0,1\}, \ \forall (i,j) \in A, \qquad (16)$$

where (13), (14) and (16) are the usual assignment constraints, (11) and (12) ensure that exactly m salesmen depart from and return back to node 1 (the depot). Although constraints (12) are already implied by (11), (13) and (14), we present them here for the sake of completeness.

Problem 4

Just try to read it for a similar kind of problems

[Parametric Mixed Integer Programming: An Application to Solid Waste Management]

Algorithms

- **Exact:**
  - Branch-and-bound algorithm
    - A branch- and bound based algorithm for the asymmetric TSP is proposed by (Ali & Kennington, 1986). The algorithm uses a Lagrangean relaxation of the degree constraints and a subgradient algorithm to solve the Lagrangean dual.
    - Ref.: [Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches]
  - Simplex Method (LP)
- **Heuristic algorithms**

- The algorithms do not guarantee an optimal solution, but gives near-optimal solutions in reasonable computational time. The Held-Karp lower bound can be calculated and used to judge the performance of a heuristic algorithm.
- There are two general heuristic classifications:
    - Tour construction procedures where a solution is gradually built by adding a new vertex at each step
    - Tour improvement procedures where a feasible solution is improved upon by performing various exchanges

  The best methods tend to be composite algorithms that combine these features.
- **Tour construction procedures:**
    - Greedy
- **Tour improvement procedures:**
    - 2-opt and 3-opt
    - Simulated Annealing
    - Genetic
        -
    - Ant Colony/ Particle Swarm
- (PDF) Optimization Techniques for Solving Travelling Salesman Problem

Stage 2

Deadline

PPT Submission: 18th February, Thursday - 11:30 AM

Presentation Link

Sic Mundus | Stage 2 - Presentation

Requirements

WAKE UP EVERYONE: **HARSHAL** 😴
TASKS TO DO BY WEDNESDAY NOON
TASKS THAT MAY BE DONE LATER

- **Final version of mathematical formulation**
  - Write for Problem 1, 2 - **Harshal**
  - Problem 3 modify based TTDP - **Miloni, Nakul**
    - Verify if our formulation is the same as the one in the paper, if not change it
    - Ref.:
      - (PDF) A survey on algorithmic approaches for solving tourist trip design problems
      - (PDF) A Genetic Algorithm for Solving Travelling Salesman Problem
- **Pseudo codes for the optimization algorithms**
  - Simulated Annealing - **Prajwal, Umang**
  - Genetic - **Souvik, Miloni**
  - Ant Colony - **Arsh, Harshal**
- **A step-by-step of what you plan to do as part of your analysis**
  - Branch-and-bound or/ Held-Karp - **Nakul, Joy**
  - Use existing graphs from TSP libraries to run all algorithms on Problem 1, and Problem 2 - **Apurva**
    - Look at how people contrast algorithms in general - metrics used to compare algorithms' quality of the solution; how many times are we supposed to run a algorithm - should we take the mean/media/min/ or some other quantity to compare the effectiveness of the algo to the global min/benchmark results; how do we quantify the efficiency of the algorithm - just the runtime/ or something more sophisticated?
    - Contact people to find out which papers they are referring to write the pseudo-code, look into those paper to see how those authors have compared their algorithms
  - Use results from existing literature to verify results of all algos for problem 3 - **Mridul**
- **This is the template that you will follow (to the extent possible) for your next two stages**

Task Allocation

1. Check for inconsistency in the mathematical formulations
    a. Problem 1 - Prajwal, Harshal
    b. Problem 2 - Arsh, Mridul
    c. Problem 3 - Nakul, Miloni *(We are not ensuring come from and go to constraint for each vertex)*
2. Write pseudo-codes for:
    a. Branch-and-bound Algorithm - Nakul
        i. Find a paper which has implemented this on TTDP
    ~~b. Greedy - Umang~~
    ~~c. 2-opt - Mridul~~
    d. Simulated Annealing - Prajwal, Umang
    e. Genetic - Souvik, Miloni
    f. Ant Colony - Arsh, Harshal
    ~~g. Particle Swarm - Harshal~~
        i. Can be worked on future **if** crowds are implemented
            1. (PDF) Particle Swarm Optimization Algorithm for the Traveling Salesman Problem
    h. Held Karp - Joy
        i. Identify the different variants for the three problems 1,2,3
3. Analysis:
    a. Quantify the closeness of algorithms - Joy
        i. Held-Karp lower bound - for heuristic
        ii. Look for more
    b. Computation cost of algorithms - Individual
        i. Number of iterations
        ii. Computation time per iteration
    c. Limitations of the algorithms (if any)
    d. Look for more - Apurva
4. Create nodes similar to a standard benchmark generic problem, so that when we optimise such a problem, we'll have literature to compare the results against - Apurva
5. Structure of pseudocode
6. Finalise/choose few algos
    a. Issues with each algo
7. Decide the set of generic problems for testing algos
    a. Orienteering Problem: A survey of recent variants, solution approaches and applications

(10/13/2021) Saturday => 11:00 AM - 12:00 PM and 4:30 - 5:00 PM

Mathematical Formulation

- A Fast Algorithm for Personalized Travel Planning Recommendation
    - This seems to capture our problem statement 2 and 3 pretty well, with scope for us to add new things as well

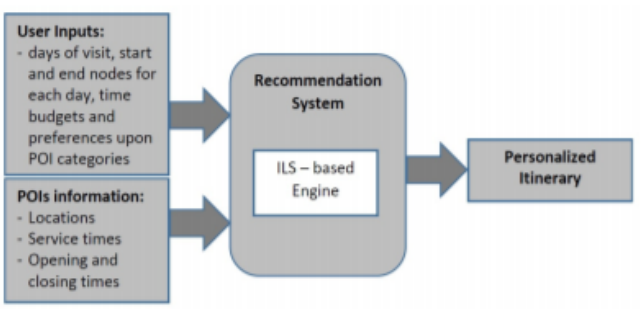


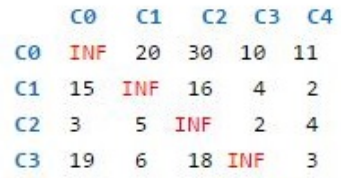
Fig. 1: TTDP Recommendation System

- [A survey on algorithmic approaches for solving tourist trip design problems]
    - This is an excellent survey paper regarding Tourist Trip Design Problems
    - Has info on loads of algorithms (20+) which are used to solve this
        - Time complexity, references, method complexity, etc
    - Good mathematical formulation

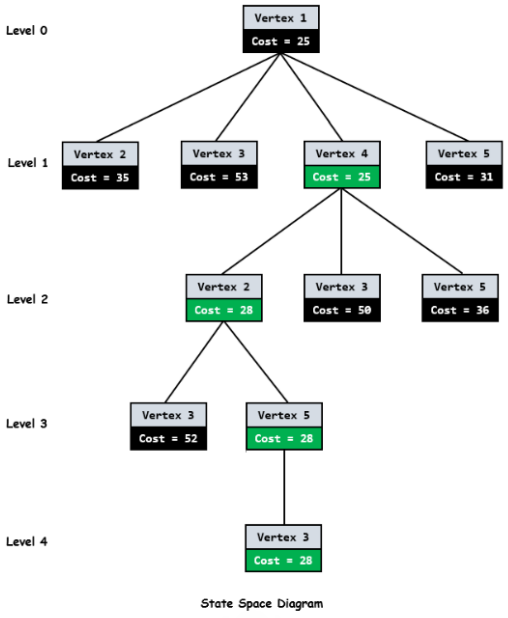
Algorithms

Branch-and-Bound Algorithms (Exact)

Final route:



	C0	C1	C2	C3	C4
C0	INF	20	30	10	11
C1	15	INF	16	4	2
C2	3	5	INF	2	4
C3	19	6	18	INF	3
C4	16	4	7	16	INF



State Space Diagram

1. References

a. Algorithm pseudocode: https://www.math.cmu.edu/~bkell/21257-2014f/tsp.pdf
b. Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches
c. Youtube video: 7.3 Traveling Salesman Problem - Branch and Bound

2. Algorithm coded in C++ and its explanation (**best one**) : Travelling Salesman Problem using Branch and Bound
3. Slides explaining the algo: Branch and Bound
4. Python: Traveling Salesman Problem | OR-Tools
5. Tasks
    a. Relate it to problem 3
        i. The selective travelling salesman problem
        ii. (PDF) A survey on algorithmic approaches for solving tourist trip design problems

6. Pseudo-code:

Global Variables: N %number of nodes%;

Struct: Node {tour %path followed%, reduced_matrix % N*N %, cost %lower bound%, id %node number%, level %total nodes visited so far%};

Functions: reduceMatrix(Node) -> reduced_matrix, incurred_cost;
createNode(parent_matrix[N][N], tour, level, i %come from%, j %go to%) -> Node;

Algorithm:

- **Begin**
    - Create a queue to store "live_nodes" {set of all accessible nodes from current node }
    - Create a root node "root"
        - input(cost_matrix %N*N%, constraint_matrix %number of constraints * 2% = 0 {it includes the set of inaccessible paths/edges; '[i,j]' is a row of constraint matrix which showcases that edge from node 'i' to node 'j' is not accessible})
        - Modify cost_matrix to set all constrained paths to infinity; set Cost_matrix[i][i] to infinity (you can't visit node 'i' from node 'i')
    - reduceMatrix(root)-> root.reduced_matrix, root.cost
    - Add root to "live_nodes"
    - while(elements are present in "live_nodes")
        - Find Node "min" which has minimum cost from the "live_nodes"
        - Remove "min" from "live_nodes"
        - int i = min.id
        - if (min is at level N-1)

- return to starting city
- Return optimal cost (“min.cost”)

➢ do for each branch node of “min” (all ‘j’ such that min.reduced_matrix[id][j] is not infinity) {`(i, j)` forms an edge in a space tree}

- if (min.reduced_matrix[i][j] is not infinity)
  - create a child node and calculate its cost;
    branch_node = createNode(min.reduced_matrix, min.tour, min.level+1, i, j)
  - Cost of the branch node = cost of the parent node + cost of the edge(i, j) + lower bound of the path starting at node j
  - Add the branch node to the list of live nodes
- Delete node as we have already stored edges ‘(i, j)` in tour

Function Definitions:

1. **reduceMatrix**(Node)
   A. rowReduction {reduce each row to get at least one zero in each row}, columnReduction {similar to rowReduction}

   B. calculate cost -> the lower bound on the path starting at current node {the total expected cost is the sum of all reductions}

2. **createNode**(parent_matrix, tour, level, i, j)
   A. {(i, j) corresponds to visiting node ‘j’ from node ‘i’ }

   B. tour -> stores current path

   C. Change all elements of row ‘i’ and column ‘j’ to infinity

   D. Set (j, 0) to infinity (you cannot go to root node)

   E. level -> number of nodes visited so far

   F. id = j

Greedy Algorithm

Choosing the option that seems the immediate best at every step. This involves starting from a point and always choosing the path with the minimum cost among those going to exhibits not visited yet.
No guarantee of solution.
https://www.geeksforgeeks.org/travelling-salesman-problem-greedy-approach/

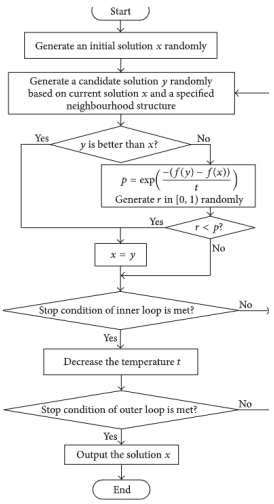
Very simple visual example included explanation:-
https://medium.com/ivymobility-developers/algorithm-a168afcd3611

Pseudocode as well:-
https://www.researchgate.net/publication/307856959_Implementation_of_Greedy_Algorithm_in_Travel_Salesman_Problem

Time complexity- $N^2 log(N)$
An approach for our Satisfaction problem: Initially apply the greedy algorithm till the time ends. Next start removing the end steps and using a heuristic of distance from end point to complete the trip.

Simulated Annealing

- Python Package for validation:
  - frigidum · PyPI - Contains an example for TSP (442 cities).
  - It is good practice (like in example) to use local_search_2opt after SA has found a potential solution.
- Caveats:
  - If it doesn't converge:
    - Check that indeed the acceptance function is always accepting shorter, and longer solutions with a probability
    - Check that in the first few proposals, most proposals are accepted, even when they are worse. If this is not the case, the initial temperature is not high enough.
    - Check that the proposals are working. You might want to print/debug the difference in the global minimum for each proposal in a round.
- Resources:
  - List-Based Simulated Annealing Algorithm for Traveling Salesman Problem



  - Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem

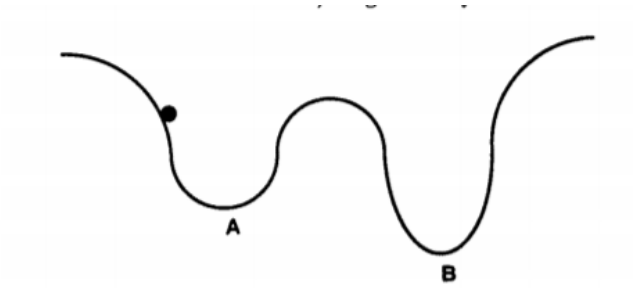


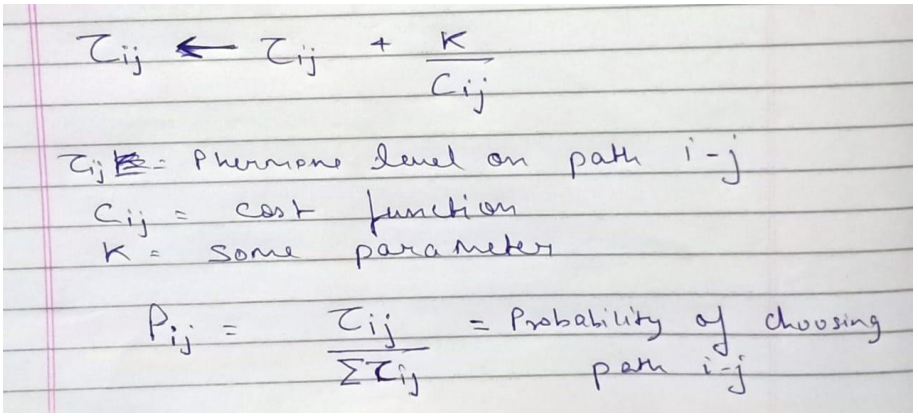
Fig. 2. Energy landscape.

- A quantitative analysis of the simulated annealing algorithm: A case study for the traveling salesman problem
  - Analysis of SA in detail
- A simulated annealing heuristic for the team orienteering problem with time windows (Link to accessible paper)
  - Contains modifications we will have to make for Problem Statement 3

- [1, 2, 3, 4, 9, 10 | 5, 6 , 7], location of bar = n
  - ApplySwap() -> Swap 2 elements in the array of nodes being visited
  - ApplyShake() -> Swap 2 elements in the array of nodes being visited and array of nodes not being visited
  - Insert()

Ant Colony Algorithm

(Ants use this technique to find the shortest path to food as well as quality of food. So we can use this algorithm to do both: Path/Time Minimisation and Maximising Satisfaction)
References:
→ How the ant colony optimization works and the mathematics behind it: YouTube Video, Article
→ An ant colony optimization method for generalized TSP problem (Contains Pseudo Code)
For problem 1:



Tasks:

1. Relate it to problem 3
   Ant colony approach to the orienteering problem
2. Read in detail
3. Time complexity
   Efficiency improvement of ant colony optimization in solving the moderate Large TSP

Time Complexity: $O(n^2m)$
Space Complexity: $O(n^2) + O(nm)$
(n = total no. of nodes, m = total no. of ants)

Pseudo code

Probability for choosing path i-j

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{s \in allowed_k} [\tau_{is}]^\alpha \cdot [\eta_{is}]^\beta} & j \in allowed_k \\ 0 & \text{otherwise} \end{cases}$$

Pheromone update (ρ = evaporation rate of pheromones, τ_{ij} = pheromones density on path i-j)

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{l} \Delta\tau_{ij}^k$$

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ travels on edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

For problem 1 and 2: $\eta_{ij} = Q/c_{ij}$ (Q = some constant)

For problem 3 = $\eta_{ij} = S_j/c_{ij}$

Code---->

1. Initialize:

Set *I*:=0 {*I is no. of iterations*}

For every edge (*i, j*) set an initial *τij* = *c* for trail density and $\Delta\tau_{ij} = 0$.

2. Set *s*:=0 {*s* is travel step counter}

For *k*:=1 to *l* do

Place ant *k* on entry. Place the entry exhibit in $visited_k$.

3. Repeat until $s \leqslant n$ (Total # of exhibits)

Set *s*:=*s* + 1

For *k*:=1 to *l* do

If T_k + T(from previous exhibit to entry) < T_{max}:

Choose the next exhibit to be visited according to the probability p_{ij}^k .

Move the ant *k* to the selected exhibit.

Insert the selected city in $visited_k$.

$T_k = T_k + (c_{ij}/v + t)$ *(Here t is fixed amount of time spent at each exhibit)*

Else:

s = n

4. For k:=1 to l do

Move the ant k from $visited_k(n)$ to $visited_k(1)$.

Compute the tour length $Lk = \Sigma c_{ij} y_{ij}$ traveled by ant k. (for problem 1 and 2)

OR

Compute total satisfaction $S_k = \Sigma S_i \Sigma y_{ij}$ traveled by ant k. (for problem 3)

For every edge (i, j) do

For k:=1 to l do

Update the pheromone trail density τij according to Eqs.

I:=I + 1

5. If (I<I_MAX) then

Empty all $visited_k$

Goto Step 2.

Else

Print the shortest tour OR highest satisfaction tour

Stop

Input : C_{ij} matrix, S_i (for problem 3)

Genetic Algorithm

- Resources
  - https://www.researchgate.net/publication/49613263_A_Genetic_Algorithm_for_Solving_Travelling_Salesman_Problem
  - https://www.geeksforgeeks.org/traveling-salesman-problem-using-genetic-algorithm/?ref=rp
  -
  - https://www.youtube.com/watch?v=3GAfjE_ChRI
  - Improved Genetic - https://aip.scitation.org/doi/pdf/10.1063/1.5039131

    - Simpler - Detailed explanation - https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
    - https://arxiv.org/pdf/1802.03155.pdf
- Limitations
    - Reaches local minima, may not find good solution
- Tasks
    - Paper to be read - Souvik
    - Mutations and crossovers to be looked into
        - Mutations - Swapping any two cities by generating random number
        - Crossovers - Complicated, will have to ensure a valid solution is obtained
    - Relate it to problem 3
        - https://www.google.com/url?sa=t&source=web&rct=j&url=http://www.msc-les.org/proceedings/mas/2015/MAS2015_196.pdf&ved=2ahUKEwimiMrstOXuAhVfgUsFHWRxBPQQFjAEegQIBxAB&usg=AOvVaw1NkBKyk5BVjBJGWcgo_5T_&cshid=1613170150485
        - https://ieeexplore.ieee.org/document/870739
        - https://www.researchgate.net/publication/226104665_Using_a_Genetic_Algorithm_to_Solve_the_Generalized_Orienteering_Problem
    - Further complexity added
        - https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9078112
    - Time complexity
        - https://www.sciencedirect.com/science/article/pii/S2211381911002232/pdf?md5=5711990197ffd967d127719195062aca&pid=1-s2.0-S2211381911002232-main.pdf

Pseudo Code:

1. Create the population
    a. Creating a random route
    b. Creating population of size n using the above function
2. Determine fitness
    a. Using a defined function to calculate fitness of our initial population
3. Select the mating pool - Selection
    a. Using Elitism to select the best k (<n) route
    b. Select (n-k) using random russian roulette method, or some new random (n-k) routes
4. Breed - Crossover
    a. Retain the elite population in children
    b. Create rest of children using breed function, Crossover by selecting 2 out of Mating pool
5. Mutate
    a. Compare a random generated number to the mutation rate, if number < mutation rate perform mutation
    b. Mutate each of the children (Swap two indices using random number)
6. Repeat
    a. Go to step 2

Input - PopulationSize, distance matrix, constraints?, Elite population number, Max number of generations, Mutation rate

createIndividual, Fitness function, Mutation Function, Crossover function,

Stopping criterion

1. Stop after max-generation is reached?
2. If we are getting the same best result for some 'm' times?

Output -

1. Best route in the generation

Example:

Cities = 25;
geneticAlgorithmPlot(population=cityList, popSize=100, eliteSize=20, mutationRate=0.01, generations=500)

2-Opt Algorithm:

- Resources:
  - https://on-demand.gputechconf.com/gtc/2014/presentations/S4534-high-speed-2-opt-tsp-solver.pdf
  - https://www.youtube.com/watch?v=dYEWqrp-mho&ab_channel=TimRoughgardenLectures
  - https://www.youtube.com/watch?v=8vbKIfpDPJI&ab_channel=TimRoughgardenLectures
  - https://www.researchgate.net/figure/Pseudo-code-for-2-opt-algorithm_tbl1_268981882

    - https://en.wikipedia.org/wiki/2-opt#:~:text=In%20optimization%2C%202%2Dopt%20is,solving%20the%20traveling%20salesman%20problem.&text=A%20complete%202%2Dopt%20local,well%20as%20many%20related%20problems.
- Limitations:
- Tasks:
    - Relate it to museum problem (Problem 2/3)
    - Time complexity

Particle Swarm

- (PDF) Particle Swarm Optimization Algorithm for the Traveling Salesman Problem
- Tasks:
    - Paper to be read
    - Relate it to museum problem
    - Time complexity

Held Karp Lower Bound

- Implementation
- Another paper explaining HK method
- Time Complexity: $O(n^2 \times 2^n)$
- Accelerating the Held-Karp Algorithm for the Symmetric Traveling Salesman Problem
- Other lower bounding and reduction procedures for STSP
- Branch and bound methods

Tasks

- Check if for our different problem statements (1,2,3) we'll be needing different lower-bound algorithms or a single will do

Some more algorithms for TSP

Benchmarks for algo performance:

Genetic -

https://www.researchgate.net/publication/27382766_On_benchmarking_functions_for_genetic_algorithm

https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume24/ortizboyer05a-html/node6.html

TSPLIB :

http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/XML-TSPLIB/instances/

Stage 3

Deadline

18th March, Thursday - 11:30 AM

Requirements

- Optimization results from at least one working algorithm.
- Bench-marking using existing built-in or standard optimization libraries for either your problem statement or a simplified version of it, or any other.

Our own requirements

- Compare our algo with a standard benchmark which is typically used by that algo in literature
- Analyse our algo w.r.t the benchmark, and the paper's result as well

Tasks

1. **TSP-Library - Souvik, Apurva (By Saturday - 13/03)**
    a. Come up with a **data-structure** for our problem
        Eg: a class in python which is initialized by the input from TSP-Lib, and has all the different types of inputs for different algorithms
    b. Will act as a common interface for all the three different algorithms
        i. Will do all the preprocessing if any, and pass only the inputs required by each algo
    c. Look for a TTDP library similar to the TSP-Lib so that we can benchmark it with some standard resource like a research paper or a standard library
2. **Simulated Annealing - Miloni, Prajwal**
    a. Input - Exhibit Number, Coordinates of exhibits
    b. Code
        ~~i. Simple (Problem 1) **(By Saturday - 13/03)**~~
        ~~ii. Read up on pseudo-code and the paper, start with the code for the complex SA algo~~
            ~~1. Slight modifications to the Simple SA python file should be enough to write the complex algo I'm guessing~~
        ~~iii. Define Performance metrics and find standard libraries to benchmark the algo **(By Sunday - 14/03)**~~
        ~~iv. Complex (Problem 1, 2, 3) **(By Monday - 15/03)**~~
    c. Check with paper's results
3. **Genetic - Mridul, (will be joined by Souvik, Apurva)**
    a. Problem 1, 2, 3 (?) **(By Sunday - 13/03)**
4. **Ant Colony - Harshal, Arsh**
    a. Problem 1, 2 **(By Sunday - 13/03)**
5. **Branch-and-Bound - Umang, Nakul, Joy**
    a. Problem 1, 2 **(By Saturday - 13/03)**

    b. Problem 3
6. **Presentation**
    a. Need to identify which algorithm to go ahead with the presentation
        i. That algorithm needs to be able work on all three problems
        ii. Performance metrics need to be defined for that algorithm
        iii. Benchmark against standard libraries
7. ~~**Make a GitHub Repo - Prajwal**~~

Presentation

Link

[Sic Mundus _ Stage 3 - Presentation]

To Do

1. BnB:
    a. Problem 1, 2
        i. Tell it's not working for test cases
    b. Problem 4
        i. Mathematical formulation
        ii. Implementation
        iii. Analysis with **temp (heuristic)** parameter
2. ACO:
    a. Problem 1, 2
        i. Compare with standard test-cases
    b. Compare with solution from the paper
    c. Problem 3
        i. Compare same tests manually generated test cases used by the team
    d. Analysis - evaporation rate, number of iterations?
3. SA:
    a. Problem 1, 2
        i. Compare with standard test-cases
    b. Compare with solution from the paper
    c. Problem 3
        i. Compare same tests manually generated test cases used by the team
    d. Analysis of all of the parameters

Comparison of Algorithms (for same input)

- Closeness of final solution
- Total runtime in each case
- Function calls
- Mean & std. dev of final solution out of **N** runs
- For a fixed runtime - quality of solution

Stage 4

Link

[Sic Mundus _ Stage 3 - Presentation]

To Do

- Presentation
    - Verification and benchmarking each algorithm.
        - Branch-and-bound
    - Optimization results from all 3 algorithms.
        - Problem Statement 3
        - Modified Problem Statement 3 with time constraint in the objective function
    - Learning and recommendations
- Source Codes
- Thoughts on post-processing

Data Input

- Use standard test-cases which use Euclidean 2D distance (5 test cases for symmetric)
    - get_cost_matrix(), get_coordinates_list()
- Add asymmetric test-cases with nodes < 20

Branch-and-Bound

- Debug the code for problem 3

Simulated Annealing

- Parameter Tuning for simple and complex SA
    - Identify the response as the parameters are modified
- Modify the objective function with time constraints
- Run the algo once by setting the same satisfaction level for all exhibits, and then with specific levels - compare the solutions
- Check with different initial solutions

GitHub Repository

Prelude

Link

https://github.com/Prajwal-Prathiksh/Museum-Path-Optimization

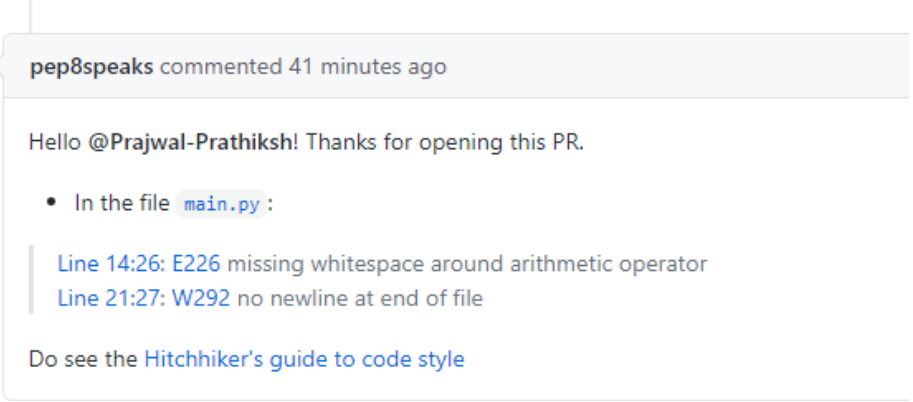
Autopep8 - Helps a lot in writing clean code (Automates the whole process!)

Branches

- **main** - Will house all of the finalised codes, output and data
- **collab** - This is the branch we will be pushing individual codes/output/data. Once the code has been vetted and reviewed it will be merged into the **main** repo

Operational Instructions

1. Fork the above repo to your own GitHub account
2. Clone it onto your laptop
    a. You can use Git (CLI), GitHub Desktop (GUI app for Mac & Windows), VS-Code, or any other IDE
3. After you have written code, and have made changes to the code/output/data, save it and push it to the **collab** branch of your **forked repo**!
    a. NOTE: You will need to give a readable commit message as to what change was made in the commit which you are pushing to the **collab** branch
4. Create a back-pull request from the **main** branch of **Prajwal-Prathiksh / Museum-Path-Optimization** to the **collab** branch of your **forked repo!**
    a. This is to ensure if someone else had pushed their code to the main repo by then, you will have those changes in your forked repo as well
5. Now generate a pull request from the **collab** branch of your **forked repo to** the **collab** branch of **Prajwal-Prathiksh / Museum-Path-Optimization**
6. A PEP8 bot has been installed, so if there is any issue in the code it will generate a comment like so:



7. Have fun coding :)

Interaction with Prof

Meet 1 (26/01)

- Can talk about optimization of anything - identify what we want to optimize and freeze the rest.
- Missiles:
    a. Optimized to hit the target
    b. Cheap - be robust
    c. Freeze the design, optimise the trajectory
    d. Identify real-life missile which would benefit from the optimization we are doing
        - Eg: ICBM need not be accurate, but cruise is important, hence max CL/CD is relevant
        - Short range missiles do not benefit from max CL/CD
- Freeze design - Controls- guidance navigation/ path optimization; Structural Optimization; - decide what to keep constant and the **single** objective
- TSP:
    a. Will get very complicated if all realistic properties of the system is taken into consideration
    b. Start with a simple problem - leading a group of tourists
        - Want them to visit all the places, but don't want to spend too much time rewalking the same path
        - TSP has many variants
        - One might be time taken/maximising-profit/etc, but define the PS aptly for the museum problem
    c. Draw elements of the pedestrian problem
    d. Design limit of 10 variables to avoid problems with computational efficiency
    e. Do multiple problems- create a series

Meet 2 (17/02)

Pseudocode:

- For this stage we need to give problem specific pseudocode. This will be almost similar to actual code.
- We should be able to compare pseudocode with the actual code.
- All algorithms must have their own pseudocode and they have to be as detailed as possible.
- Sir wishes to see minimum 3 algorithms (do 5 if we get more time)

Code:

- In the main code: Input file, Main constraints, Optimizer
    - Optimizer should be independent of the problem statement

- Algorithm should be decoupled from the problem
    - Do not hardcode anything
- In the last stage sir might tell us to change input some params
- For GA we can look into MATLAB implementation to get an idea of input attributes
- Towards the end we can think of having an hybrid optimizer like Branch and bound + GA or something

Stage 2

- Decent job
- Complete Implementation of 1 algorithm along with benchmarking and analysis (in code using some formulas) to be done in stage 3
- Can complicate problem statement for the 1 algorithm

Pseudocode

Genetic

Pseudo Code

Inputs:

Parameters : PopulationSize, distance matrix, constraints, Elite population number, Max Generations, Mutation rate

Functions: *createChromosome(), Fitness(Chromosome, distance matrix, constraints), Mutate(Chromosome), Crossover(Chromosome1, Chromosome2)*

Class *Individual:*

Chromosome

Fitness value

Function *createIndividual (createChromosome())*

(*Individual→Chromosome) ← createChromosome()*

return *Individual*

Function *calculateFitness(Individual, Fitness())*

(Individual → Fitness value) ← Fitness(Individual → Chromosome)

Function *createPopulation (PopulationSize, createChromosome())*

Array Population*[PopulationSize]*

for Loop over *PopulationSize:*

createIndividual(createChromosome())

Add *individua*l to *Population* **Array**

return *Population*

Function *FitnessPop(Population, PopulationSize,Fitness())*

i = 0

for Loop i from 1 to (*PopulationSize-1):*

calculateFitness(Population[i], Fitness())

Function *matingPool (Population, PopulationSize, ElitePopulationNumber)*

Array Parents[PopulationSize];

Sort*(Population)* #Based on fitness value of each individual

Array *matingPool[PopulationSize];*

i = 0

for Loop i from 0 to (*ElitePopulationNumber-1):*

Parents[i] = Population[i]

for Loop i from *ElitePopulationNumber to (PopulationSize-1):*

Parents[i] = createIndividual(createChromosome())

return *Parents*

Function *crossoverPop(Parents, PopulationSize, ElitePopulationNumber,Breed())*

Array *Children[PopulationSize]*

i = 0

```
        for Loop i from 0 to (ElitePopulationNumber-1):
                Children[i] =Parents[i]
        Shuffle Parents and store in shuffledParents;
        for Loop i from ElitePopulationNumber to (PopulationSize-1):
                Children[i] = Breed(shuffledParents[i], shuffledParents[PopulationSize  - i - 1])
        Return Children

Function mutatePop(Children, PopulationSize, Mutation rate, Mutate())
        Array newChildren[PopulationSize]
        i = 0
        for Loop i from 0 to (PopulationSize - 1):
                Random_number < mutation rate          #generate random number
                        newChildren = Mutate(Children[i])
        Return newChildren

Array Population = createPopulation(PopulationSize, createChromosome())
Generation = 0
While Generation < MaxGeneration:
        Generation <- Generation + 1

        #Fitness of each individual
        FitnessPop(Population, PopulationSize)

        #Select Mating Pool
        Array Parents[PopulationSize] =  matingPool(Population, PopulationSize,
                                                ElitePopulationNumber)

        #Breed
        Array Children[PopulationSize] = crossoverPop(Parents, PopulationSize,
                                                ElitePopulationNumber)
        #Mutate
        Array newChildren[PopulationSize]  = mutatePop(Children, PopulationSize,
                                                Mutation rate, Mutate())

        Population = newChildren

Sort(Population)          #Based on fitness value
bestRoute = Population[0]

Function createChromosome(n)
        for (i:1->n)
                arr[i]=1
        for (j:1->n)
                k=randomise(n-1) %generates random indice from 0-n-1
                swap(arr[j],arr[k])
        return arr[n] %random path
```

Function *Fitness(Chromosome, distance matrix, constraints)*

```
for (i:1->n)
        check dist(Chromosome[i-1],Chromosome[i])= exists
        check constraints
if TRUE
        return 1
else
        return 0
```

Function *Mutate(Chromosome)*

```
l,k=0
While (l=k) Do {
        l=randomize(n-1)
        k=randomize(n-1)
}
Chromosome2= Swap(Chromosome[l],Chromosome[k]
return Chromosome2
```

Function *Crossover(Chromosome1, Chromosome2)*

```
Crossover[n], temp1[], temp2[]
i=randomize(n-1)
j=randomize(n-1)
p=min(i,j)
q=max(i,j)
for (k:p->q)
        temp1.append(Chromosome2[k])
for (r:0->n-1)
        If (Chromosome1[r] not in temp1)
                temp2.append(Chromosome1[r])
return Crossover=temp1+temp2
```

Branch-and-Bound

Global Variables: N %number of nodes%

Struct: Node {tour %path followed%, reduced_matrix %[N][N]%, cost %lower bound%, id %node number%, level %total nodes visited so far%}

Functions:

- reduceMatrix(Node) -> reduced_matrix, incurred_cost
- createNode(parent_matrix[N][N], tour, level, i %come from%, j %go to%) -> Node

Input:

- cost_matrix %[N][N]%
  {it consists of the cost/penalty involved in moving from one node(row number) to next(column number)}
- constraint_matrix %[number of constraints][2]%
  {it includes the set of inaccessible paths/edges; '[i,j]' is a row of constraint matrix which showcases that edge from node 'i' to node 'j' is not accessible})

Algorithm:

- Begin
  - Create a root node "root"
    - Take the input
    - Modify cost_matrix to set all constrained paths to infinity
    - Set Cost_matrix[i][i] to infinity (you can't visit node 'i' from node 'i')
  - reduceMatrix(root)-> root.reduced_matrix, root.cost
  - Create a queue to store "live_nodes" {set of all accessible nodes from current node}
  - Add root to "live_nodes"
  - while(elements are present in "live_nodes")
    - Find Node "min" which has minimum cost from the "live_nodes"
    - Remove "min" from "live_nodes"
    - int i = min.id
    - if (min is at level N-1)
      - Return to starting city
      - Return optimal cost ("min.cost")
    - do for each branch node of "min"
      (all 'j' such that min.reduced_matrix[id][j] is not infinity)
      {(i, j) is an edge}
      - if (min.reduced_matrix[i][j] is not infinity)
        - create a branch node and calculate its cost;
        - Cost of the branch node = cost of the parent node + cost of the edge(i, j) + lower bound of the path starting at node j
        - Add the branch node to the list of live nodes
    - Delete node as we have already stored edges '(i, j)` in tour

Function Definitions:

1. **reduceMatrix**(Node)
   A. rowReduction {reduce each row to get at least one zero in each row}, columnReduction {similar to rowReduction}
   B. calculate cost -> the lower bound on the path starting at current node {the total expected cost is the sum of all reductions}

2. **createNode**(parent_matrix, tour, level, i, j)
   A. {(i, j) corresponds to visiting node 'j' from node 'i' }
   B. tour -> stores current path
   C. Change all elements of row 'i' and column 'j' to infinity
   D. Set (j, 0) to infinity (you cannot go to root node)
   E. level -> number of nodes visited so far
   F. id = j

Simulated Annealing

procedure Simulated_Annealing(α, β, γ, δ, ϵ, ζ, μ, k)
    candidate_solution ← **GenerateInitialSolution**(ζ)
    Evaluate *candidate_solution*
    best_solution ← *candidate_solution*
    $T \leftarrow \beta, t \leftarrow 0, i \leftarrow 0, j \leftarrow 0$
    while $T > \gamma$ **do**
        for 1 to *N*
            new_solution ← **ApplySwap(***candidate_solution***)**
            if *new_solution* is a feasible solution **then**
                Evaluate *new_solution*
                if cost of *new_solution* > cost of *candidate_solution* **then**
                    candidate_solution ← *new_solution*
                    if cost of *new_solution* > cost of *best_solution* **then**
                        best_solution ← *candidate_solution,*
                        $i \leftarrow 0$
                    else
                        $i \leftarrow +1$
                    end if
                else
                    a ← GenerateRandomNumber(0,1)
                    $b \leftarrow \frac{T-\gamma}{\beta-\gamma}\exp(\frac{cost\ of\ new\_solution\ -\ cost\ of\ candidate\_solution}{T})$
                    if $a < b$ **then**
                        candidate_solution ← *new_solution*
                    end if
                end if
            else
                $j \leftarrow +1$
            end if
            if $j > \mu$ **then**
                candidate_solution ← **ModifyNodes(***candidate_solution* , k**)**
                continue
            else if $i > \delta$ **then**
                do
                    candidate_solution ← **ApplyShake**(*candidate_solution* , ϵ , p)
                while *candidate_solution* is not a feasible solution
            end if
        end for
        $t \leftarrow +1$
        $T \leftarrow$ **ApplyCoolingFunction**(t, α)
    end while
    return *best_solution*
end procedure

Input

- **Nodes being visited:** [1, 2, 3, 4, 5]
- **Nodes not visited:** [9, 8, 7]
- This can be represented by a single array like so: [1, 2, 3, 4, 5 **||** 9, 8, 7]
- **Actual input** is only a single array, and the location of the bar represented as:
    - **nodes=[1, 2, 3, 4, 5, 9, 8, 7]**
    - **loc_bar=5** *(=Number of elements in array of nodes being visited, thus, 1<=loc_bar<=n; n = total number of node)*

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]
- **ApplyConsecutiveSwap():** Swap 2 consecutive elements in the array randomly
- **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

Observations

- If we use `**exponential_cooling_func**`, then T>gamma condition has to be used, and a fixed number of epochs condition will not work since it leads to double overflow, i.e, T→ 0, and thus the check condition (L25) fails

Ant Colony

Probability for choosing path i-j

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{s \in allowed_k} [\tau_{is}]^\alpha \cdot [\eta_{is}]^\beta} & j \in allowed_k \\ 0 & \text{otherwise} \end{cases}$$

Pheromone update (ρ = evaporation rate of pheromones, τ_{ij} = pheromones density on path i-j)

$$\tau_{ij}(t+1) = \rho \cdot \tau_{ij}(t) + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{l} \Delta\tau_{ij}^k$$

$$\Delta\tau_{ij}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ travels on edge } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

For problem 1 and 2: $\eta_{ij} = Q/c_{ij}$ (Q = some constant)

For problem 3 = $\eta_{ij} = S_j/c_{ij}$

Code---->

1. Initialize:

Set I:=0 {*I is the no. of iterations*}

For all edges (i, j) set an initial $\tau_{ij} = c$ for pheromone density and $\Delta\tau_{ij} = 0$.

2. Set s:=0 {s is the count of no. of steps taken}

For k:=1 to l do

Place ant k on entry. Place the entry exhibit in $visited_k$.

Set $T_k = 0$ (T_k is used to calculate total time taken by each ant)

3. Repeat until $s \leqslant n$ (Total # of exhibits)

Set s:=s + 1

For k:=1 to l do

If T_k + T(from previous exhibit to entry) < T_{max}:

Choose the next exhibit to be visited according to the probability p_{ij}^{k} .

Move the ant k to the selected exhibit.

Add the selected exhibit in $visited_k$.

$T_k = T_k + (c_{ij}/v + t)$ *(Here t is fixed amount of time spent at each exhibit)*

Else:

s = n+1

4. For k:=1 to l do

Move the ant k from $visited_k(n)$ to $visited_k(1)$.

$T_k = T_k + (c_{i1}/v + 1)$ (Time to go to entry again)

Calculate the length of tour $L_k = \Sigma c_{ij} y_{ij}$ traveled by ant k. (for problem 1 and 2)

OR

Compute total satisfaction $S_k = \Sigma S_i \Sigma y_{ij}$ traveled by ant k. (for problem 3)

For every edge (i, j) do

For k:=1 to l do

Update the pheromone levels τ_{ij} using the formulas

I:=I + 1

5. If (I<I_MAX) then

Empty all $visited_k$

Repeat from Step 2.

Else

Print the tour with shortest length OR highest satisfaction tour

Stop

Input : C_{ij} matrix, S_i (for problem 3)