IT257 - Tackling the Travelling Salesperson Problem using Dynamic Programming, Simulated Annealing and Firefly Algorithm

Satyam Agrawal - 211AI044
Anushka Agrawal - 211AI006
Gulshan Goyal - 211AI044
Information Technology, National Institute of Technology Karnataka, Surathkal, India 575025
Email: *satyamagrawal.211ee251@nitk.edu.in, †anushkaagrawal2580@gmail.com, ‡gulshangoyal01@gmail.com

***Abstract*—Traveling Salesman problem (TSP) is a well-known NP-hard problem. It is well known where a travel salesman starts from a specific city, visits every city, and returns back to the starting city. In this paper, we are using Dynamic programming (DP), firefly algorithm, and simulating annealing algorithm to find the optimal route for a salesman so that he can visit every given set of cities and the total distance travelled is minimized. Dynamic programming has been widely used for breaking down the problem into smaller sub-problems and stores the solution of the problem in a table. Since it is well known that dynamic programming always finds the optimal route by efficient computing. Firefly algorithm is inspired by the nature of fireflies; it imitates the flashing patterns of fireflies for searching an optimal path. On the other hand, Simulated Annealing considers both improving and worsening solutions so that it can escape local optima and find the optimal solution. The aforementioned two methods are heuristic in nature. The paper discusses the drawbacks and advantages of using each method while shedding light on the intricacies of each of them.**

Keywords:
Travelling Salesperson Problem, NP-Hard, Dynamic Programming, DP table, Firefly Algorithm, Simulating Annealing, Temperature, Combinatorial Explosion

I. Introduction

The problem we are trying to solve is essentially an optimization problem known as the Traveling Salesperson Problem (TSP). The problem can be defined as follows:

There is a salesman who wishes to sell his product in a particular region. He needs to travel from one city to another in that region and, at the end, return to where he started from. The problem is to provide a path that accomplishes this with the minimum distance possible, such that the salesman has to travel the least while covering each city.

The aim of the solution will be to find an optimal path for the aforementioned problem. The TSP is an NP-complete optimization problem, which means that finding an efficient solution has the potential to be mapped to many similar problems of practical application, such as scheduling and facility location. Moreover, many direct applications of the TSP are essential in daily world applications, such as vehicle routing and tour planning.

The TSP is challenging due to several reasons, with the primary one being combinatorial explosion. The number of possible paths for n nodes is equal to:

$$\frac{(n-1)!}{2}$$

Fig. 1: Number of all possible paths

and this number grows exponentially with the number of cities. Additionally, the TSP is NP-hard/complete, which means that no known polynomial-time solution exists.

There are multiple heuristic and brute-force approaches available to solve the problem. Some heuristic methods used in this project are the nearest neighbor algorithm, 2-opt method, simulated annealing (a meta-heuristic), and firefly optimization (a swarm intelligence technique). The core idea behind this project was to understand the workings of various heuristic-based algorithms and then compare their efficiency with the brute-force counterparts. Hence, we chose three contrasting methods: dynamic programming as a brute-force method, simulated annealing, and the firefly algorithm.

The motivation behind choosing this problem and the selected solutions was to solve related problems optimally. The main problem we focus on is tour planning. As the Karnataka state government plans to make the Dakshina Kannada region more open and accessible to the world as a tourist location, this project could be applicable in that scenario. Our idea is a fair representation of the types of methods available to solve the problem and serves the purpose of comparing the aforementioned methods and models. However, a more in-depth study is required to develop more complex and efficient models using a combination of swarm optimization and meta-heuristic methods.

The project aims to be a representative comparison among the best optimization techniques currently in place and serves as a good starting point to develop more advanced models by combining these components.

A. Workflow in Bullets

- Use of Dynamic Programming to solve the TSP
- Use of Greedy, Nearest Neighbor, and 2-opt approaches
- Use of Simulated Annealing
- Analysis of Simulated Annealing using different initial states
- Use of Firefly Algorithm

The rest of the paper is structured as follows. Section II contains the literature survey, which consists of the works considered by the authors. Section III provides a summarized version of the problem statement and the objectives. This is followed by Section IV, which presents the methodology applied in the project. Section V discusses the dataset used to solve the TSP and provides insight into how it was extracted with optimized values. Section VI analyzes the results obtained and presents the experimental observations. Time complexity analysis and other discussions are also included in this section. The paper ends with Section VII, which presents the final conclusion.

II. Literature Survey

[1]: Ben Amuer et al. carried out mathematical operations in order to determine aspects of initial temperature used for Simulated Annealing. In the cited paper, author first proposes a simple algorithm to compute a temperature that is compatible with a given acceptance ratio. The author then proceeds to study the properties of the acceptance probability, demonstrating that this function exhibits convex behavior for low temperatures and concave behavior for high temperatures. Additionally, the paper provides a lower bound for the number of plateaux in a simulated annealing algorithm based on a geometric cooling schedule.

[2]: Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach by Chetan et al. This paper used dynamic programming method for finding an optimal solution to the travel salesman problem.this method gives correct result in reasonable time. It also give two groups for TSP solver one of these include method like branch-and-cut,branch-and-bound,cutting plane and interior point and another one is using dynamic programming for smaller group.the common thing in both of the solver is that it give optimal solution at the expense of space requirements and execution time (running time).

[3]: In this literature survey, we provide a brief overview of the fundamental concepts of the firefly algorithm, highlighting its key characteristics and mechanisms (Yang et al.). We also present a curated selection of recent publications that showcase the extensive range of applications where the firefly algorithm has been successfully employed.

III. Problem Statement

The problem involves solving Traveling Salesman Problems using a Simulated Annealing algorithm, Dynamic Programming, and the Firefly algorithm. The objective is to find the optimal solution for a Traveling Salesman by considering factors such as computation time, solution quality, and its ability to handle larger problem instances.

Name	Methodology	Merit	Limitations
Yang(2008)	Developed Firefly Algorithm	Fast convergence and global exploration	No local search
Ben-Ameur	Temperature Schedule for Simulated Annealing	Optimal Temperature schedule	Mathematically complicated
Chetan et.al	TSP using DP	Correct solution	Computationally expensive
Chen and Yang(2010)	Incorporated 2-opt local search	Improved solution quality	Limited exploration

Fig. 2: Literature Survey

A. Objectives

- To develop a representative comparison reference for various methods that are used to solve TSP
- Rate the methods based on their computational cost and accuracy
- Provide a path to develop ensemble models
- Develop an understanding of parameter tuning and the sensitivity

IV. Methodology

In totality, three methods have been employed to solve the Traveling Salesperson Problem (TSP), and they are explained in the following order:

1) **Dynamic Programming**
2) **Simulated Annealing**
3) **Firefly Optimization**

Dynamic Programming

In the Dynamic Programming approach, a TSP function is introduced, which takes several parameters:

- Bit-mask (*mask*)
- Current Position (*pos*)
- Graph (to represent distances between different cities)
- Look-up table (all cities and visited bit-mask)

The function checks if all the cities are visited or not. If all cities are visited, it returns the length from the current position back to the starting city/point. The function utilizes memoization to check if the result for the current bit-mask and current position is already computed and stored in the dynamic programming table (*dp*). If it is already stored, the function retrieves the stored value instead of recomputing it.

A variable is initialized to store the answer, initially set to infinity. The function then iterates through each city using a loop. For each city, it performs a bit-wise AND operation between (1<<city) and *mask* to check if the city is unvisited.

The function makes a recursive call to itself, passing the updated bit-mask, the new city as the current position, the total number of cities, and the visited bit-mask. It also updates and adds the length from the current position to the new city.

At each recursive call, the function compares the new value with the current minimum and updates the minimum value accordingly. If it is the minimum value, it stores the minimum value in the table for the bit-mask and current position.

The result is the minimum distance for the provided bit-mask and position among all cities. The graph is created with distances between cities, and all calculations, including the total number of cities and visited bit-masks, are based on the graph. A 2D array is used to store the results, initially initialized as -1.

To measure the execution time of the program, the code records the start time before running the TSP function and the end time after the function completes execution. The execution time is then calculated by subtracting the start time from the end time. The code outputs the shortest route distance and the execution time.

This approach utilizes dynamic programming to solve the TSP by finding the shortest route that visits all cities exactly once. It also measures the execution time and provides the corresponding result.

```
TSP(C,A)
  Visited[A]=1;
  if|C|=2 and k≠ A then
     Cost (C,k) = dist(A,k);
     return Cost;
  else
     for j ∈C, do
          for i∈C and visited[i]=0 do
               if j≠i and k≠A then
                    Cost(C,j) =min(TSP(C-{i},j)+dist(j,i))
                         visited[j]=1;
               end
          end
     end
  end
  Return Cost;
end
```

Fig. 3: Pseudo-code

A: starting point, C: a subset of input cities,dist(): distance between cities
result: Cost: Traveling sales man result,where Visited[N] =0 and Cost=0

Fig. 4: Legend

SIMULATED ANNEALING

Simulated annealing is a meta-heuristic algorithm inspired by the metallurgical process of annealing. It is employed to solve optimization problems and is a development over the hill climbing algorithm to solve the problem of local minima convergence. Annealing refers to a process where a piece of metal is heated and then allowed to cool slowly in order to reduce internal stress. This same analogy is applied to the state in the case of simulated annealing to reduce computational cost as well as to converge towards global minima.

Since the traveling salesperson problem is an optimization problem, simulated annealing is a reasonable method to solve it. Simulated annealing heavily relies on the initial state, similar to its precursor: the hill climbing or 2-opt method. In order to take this fact into account, it is essential to play around with the initial states and compare the results. The results obtained from the Greedy Algorithm and Nearest Neighbor are used as starting states for 2-Opt and Simulated Annealing.

Before proceeding with simulated annealing, it is necessary to clarify the reasoning behind the implementation of 2-opt, which are:

- to serve as a precursor to Simulated Annealing
- to help in comparing the results obtained

Nearest Neighbour:
In the nearest neighbour heuristic, one starts with a random city and mark it as the current city. The next step is to choose a city that is closest to the current city and accept it if it has not yet been visited.Following this, set this new city as the current city and continue the process until all nodes of the relevant graph are covered.

Greedy Algorithm:
The greedy approach is applied to select edges with optimum weights. The edges with minimum lengths are selected iteratively, with two conditions in mind. The edges should not form a cycle. No node should have a degree greater than two. Following these two constraints, the final path is obtained when all the edges are traversed.

2 Opt:
2-Opt stands for two optimization, where 2 represents the number of nodes exchanged at each step. We start by taking either a random or a certain special initial state. At every step, a random exchange of any two nodes of the state is carried out following which, the cost is calculated. If the calculated cost is optimal or better when compared to the current one, the new state is selected as the current state. This process is carried out until a situation is reached where no exchange can guarantee a better solution, also known as minima. Several iterations of this algorithm with varying initial states help in converging towards the global minimum solution. The algorithm chart is as follows:

Simulated Annealing:
A modification over 2 Opt, simulated annealing helps reach the global minima faster and more efficiently by infusing some uncertainty of selection in the algorithm. It consists of a few parameters to moderate and model the uncertainty:

- Temperature
- Change Rate
- Neighbours
- Stopping Temperature

Temperature is a parameter representative of the acceptance

probability of a non optimal solution state. The higher the temperature, the higher the probability that a non optimal solution state will be accepted. When the implementation of the algorithm begins, the temperature is at its maximum or initial value. It is progressively reduced as the number of iterations increases. This reduction of temperature is known as cooling and is essential to ensure that the algorithm converges to a minimum.

The **Change Rate** is the rate at which the temperature is reduced and is often referred to as the cooling constant. There can be three types of change rates:

- Linear
- Geometric
- Slow Reduction

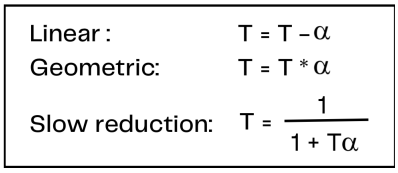

Fig. 5: Types of Change Rates

Neighbours refer to the set of states close to the current state with slight changes or modification. At each set, a set of neighbours is generated and are accepted or rejected as per their acceptance probability that is dependent on their cost. After each neighbour set is analysed, the temperature change or reduction is done to ensure convergence.

Stopping temperature is the value after which no non-optimal solutions is accepted. After the stopping temperature is reached, 2 Opt is applied to the state for convergence to minima.

Mathematical Modelling: Fig 2 shows the mathematical modelling of temperature and cost and exactly how it affects the selection (or rejection) of a particular state. A certain acceptance probability is calculated using the mentioned formula which is then compared to a random number generated between 0 and 1, according to which the decision is made.

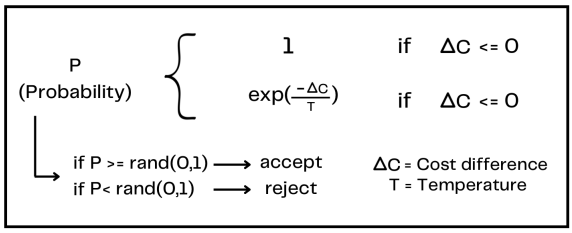

Fig. 6: Selection/Rejection of a state

- Initialize an **initial solution** <-- **Path**
- Initialize the **initial temperature** <-- **T**
- Repeat until the temperature drops below the **stopping temperature** (**s** = **0.025**):
  - Repeat for a fixed number of iterations:
    - Generate a random **neighboring** solution from the current solution.
    - Calculate the **energy** (cost) of the current solution.
    - Calculate the **energy** (cost) of the new solution.
    - If the new solution has a lower energy (**Δc**), accept it as the current solution and update the best solution if it's the best encountered so far.
    - If the new solution has a higher energy (**Δc**), calculate the acceptance probability based on the temperature and the energy difference between the current and new solutions.
    - Generate a random number between 0 and 1.
    - If the acceptance probability is **greater** than the random number, **accept** the new solution as the current solution.
  - Reduce the temperature by multiplying it by the **cooling rate**.
- Return the best solution found.

Fig. 7: Pseudo-code for Simulated Annealing

FIREFLY ALGORITHM

Fireflies possess a unique characteristic known as bioluminescence, where they emit flashes of light. This flashing behavior serves two purposes: attracting mates and warning off predators. In the context of an optimization algorithm, the flashing pattern forms the basis for determining the light intensity function (I), which is inversely proportional to the square of the distance.

In this algorithm, all fireflies are assumed to be unisexual, meaning each firefly can be attracted to any other firefly. The light intensity between two fireflies influences their attractiveness (beta) towards each other, and both light intensity and attractiveness are calculated using similar formulas. The distance (r) between fireflies is measured using the Euclidean distance.

During each iteration of the algorithm, an i-th firefly is moved towards a brighter j-th firefly, and the light intensities of the fireflies are updated accordingly. If two fireflies have equal brightness, one of them is randomly selected for movement. The algorithm begins with a random population, and with each iteration, it moves towards finding a better solution.

The Firefly Algorithm involves three parameters: the randomization parameter (alpha), the attractiveness parameter (beta), and the absorption coefficient (gamma), which can be adjusted using various techniques to optimize the algorithm.

Mathematical Modelling:

In the Firefly Algorithm adapted for the Traveling Salesman Problem (TSP), where each firefly represents a random permutation of cities, the following formulas are used:

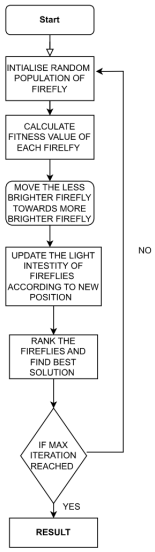

Fig. 8: Pseudocode for Firefly Algorithm

Light Intensity (I)

In the general Firefly Algorithm, the formula for light intensity (I) is:

$$I = I_0 \cdot \exp(-\gamma \cdot r^2)$$

In the TSP, the light intensity (I) represents the fitness or objective function value and is inversely proportional to the total distance of the tour:

$$I = \frac{1}{\text{total distance of the tour}}$$

Distance Calculation (r)

In a typical Firefly Algorithm, the distance (r) between two fireflies is calculated using the Euclidean distance formula:

$$r = \sqrt{\sum((x_i - x_j)^2 + (y_i - y_j)^2)}$$

However, in the context of the TSP, when calculating the distance between two fireflies representing different TSP solutions, their respective permutations of cities should be used to compute the total tour distance.

Attractiveness (β)

The attractiveness (β) between two fireflies representing different TSP solutions can be calculated based on their distances:

$$\beta = \beta_0 \cdot \exp(-\gamma \cdot r^2)$$

Here, β_0 is the initial attractiveness, and γ is the absorption coefficient.

Movement of Fireflies

In the typical Firefly Algorithm, the movement of a firefly (represented as s_i) towards a brighter firefly (represented as s_j) is given by the formula:

$$s_i = s_i + \beta_0 \cdot \exp(-\gamma \cdot r^2)_{ij} \cdot (s_j - s_i) + \alpha \cdot \epsilon_i$$

To move a firefly representing a TSP solution towards a brighter firefly, swap the cities that are not in common between their permutations. This swapping operation aims to improve the TSP solution by exchanging city visitation orders.

Randomization Parameter (α)

The randomization parameter (α) controls the level of randomness in the movement of fireflies and is typically a value between 0 and 1. It can be used to introduce exploration and diversification during the movement process.

To summarize:

1) Calculate the light intensity based on the inverse of the total distance of the TSP tour.
2) Calculate the distance between fireflies using their respective permutations of cities.
3) Calculate the attractiveness between fireflies based on the distance.
4) Move a firefly towards a brighter firefly by swapping the cities that are not in common.
5) Incorporate the randomization parameter to introduce randomness in the movement.

It is crucial to implement these formulas accurately and appropriately for the Firefly Algorithm in the context of the TSP to achieve optimal results.

V. Datasets

Standard TSP graphs were adopted from TSP.LIB library. The graphs were in the form of upper diagonal row and low diagonal row and had to be converted into an adjacency matrix format for use. Graphs chosen had the following configuration:

Dataset	Nodes	Optimal Cost
gr17	17	2085
gr24	24	1272
dantzig42	42	699
brazil58	58	25395
gr120	120	6942
eil22	22	283
berlin52	52	7542
rat99	99	1211

Fig. 9: Configuration of model graphs

VI. Experimental Results and Analysis

Dynamic Programming

To conduct an experimental analysis, the code was executed on various problem instances with different sizes of input (number of cities). The execution time was recorded for each node, and the results were analyzed by plotting a graph of problem size against execution time. This analysis provided insights into the performance characteristics of the

algorithm and assessed its accuracy and efficiency in solving the Traveling Salesman Problem (TSP).

When attempting to solve the TSP using dynamic programming, we noticed that it works perfectly for 17 nodes. However, as we increased the number of nodes in our program, the execution time increased drastically due to its time complexity. The dynamic programming approach involves calculating C(i, S) for all subsets of size 2, then calculating C(i, S) for all subsets of size 3, and so on, following the formula g(i, s) = minC(i, k) + g(k, S - k) to calculate the optimal path.

For the experimental analysis, we executed the TSP problem for each node and recorded the execution time. The results were then analyzed by plotting a graph of problem size against execution time. This analysis provided valuable insights into the algorithm's performance characteristics, as well as assessing its accuracy and efficiency in solving the TSP.

To calculate the cost or optimal route, we stored the results for subproblems in a dynamic programming table, which was used to build up the solution for larger subproblems until the complete optimal route was obtained. The TSP function took into account the bitmask, distance between cities, current position, the dynamic programming table for storing values, visited mask, and the total number of cities.

Inside the TSP function, we checked if the mask was equal to the visited bitmask. If so, it meant that the city was already visited, and the function returned the distance from the current position back to the starting city. If the result for the bitmask and current position was already computed and stored in the dynamic programming table, it was retrieved directly without recomputing.

If the base case was not met, the function proceeded to calculate the optimal route. We initialized the answer variable with a large value to track the minimum cost. The loop iterated through each city using the range from 0 to the total number of cities minus one. It also checked if the city was not visited by performing a bitwise AND operation between the mask and (1 << city). If the result was zero, it indicated that the city was unvisited.

For the cities that were not visited, the function made a recursive call to itself, updating the mask by setting the corresponding city's bit to one using the bitwise OR operation. The minimum value was updated according to preference.

When the loop finished, the minimum cost of the position and mask was stored in the dynamic programming table.

However, it is important to note that the execution time poses a challenge for nodes with larger sizes, such as 24, 42, 58, 120, etc., due to the exponential growth of the problem complexity. The time complexity of dynamic programming for solving the TSP is:

$$\mathrm{O}(N^2 \cdot 2^N)$$

where N is the number of nodes. Therefore, as the number of nodes increases, the execution time increases exponentially. This occurs because the number of possible permutations of paths to explore grows factorially with the total number of nodes provided in the graph as input.

SIMULATED ANNEALING

The setup starts with importing the model graphs from the TSP.LIB library. The graphs are then preprocessed into their adjacency matrix format. The next step is to obtain the paths for the Nearest Neighbor approach and Greedy approach for them to serve as the starting point for both 2-opt and Simulated Annealing. Once the initial points in the form of the nearest neighbor and greedy algorithm are obtained, the 2-opt model is now employed. The results of the 2-opt model are noted in a format so that an idea about the relevance of the starting state can be obtained. The following figures show the results with variable start states:

	Greedy Algorithm	Nearest Neighbour	2 opt with Greedy	2 opt with Nearest Neighbour	2 opt
Graph1	2351.000000	2187.000000	2253.000000	2085.000000	2103.000000
Graph2	1639.000000	1553.000000	1376.000000	1475.000000	1425.000000
Graph3	1005.000000	956.000000	935.000000	856.000000	912.000000
Graph4	34491.000000	30774.000000	33367.000000	28593.000000	32951.000000
Graph5	8830.000000	9351.000000	8549.000000	8698.000000	11783.000000
Graph6	315.726851	312.088870	315.726851	300.854576	324.257709
Graph7	10108.395041	8980.918279	9293.399679	8667.835555	9875.300328
Graph8	1580.181675	1564.724855	1543.606889	1394.103139	1847.312182

Fig. 10: 2 opt results

Key Analysis (2-Opt):

- For Graph 1, we are able to obtain the optimal result by the application of 2-opt on the nearest neighbor path.
- 2-opt allows for exploration of the search space due to a random initial state and hence tends to have higher variability. This proves useful in the case of Graph 2 and Graph 3, where the lower bound for plain 2-opt is better than the nearest neighbor or greedy 2-opt.
- In the case of Graph 4 and Graph 5, we observe that the nearest neighbor and greedy 2-opt outperform the plain 2-opt. This is due to the fact that with an increasing number of nodes, the number of local minimum rises, which makes less variation an advantage.

Initial Temperature in Simulated Annealing:

One of the most important factors for convergence or not convergence was seen to be Initial Temperature used for the problem. In the paper titled- Computing the Initial Temperature of Simulated Annealing, the authors talk about the initial temperature calculation. Two of the methods discussed in the same were adopted apart from the iterative version of the solution.

Method 1 suggested the maximum cost difference between any two states of the problem to be considered as the initial temperature.

Method 2 suggests that Initial temperature is equal to $T_0 = -\sum E \ln(\chi_0)$, where $\sum E$ an estimate of the cost difference values. This estimation is obtained by averaging the cost difference values for some randomly generated transitions.

$-\delta_t \ln(\chi_0)$, where δ_t is the cost increased by t, χ_0 is the acceptance probability. One can also comment that, $T_0 = -\sum E \ln(\chi_0)$ is the average of these temperatures over a range of transformations with a certain acceptance rate.
There is also a third value that was taken into account and gave some extremely optimal results, which was the average of the values of temperature obtained by the aforementioned two methods. Apart from these, in case of a non convergence, iterative form of temperature search was carried out. However, it must be reported that the extent of the same was limited due to limitations on the computational ability available at hand.

The following tables help demostrate the effect of starting temperature on the goal state:

Temp: 1797.9300893632344 Nodes: 17

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	2154	2095	2090	2123	2085	2085	2088	2088
1	15	2088	2085	2090	2088	2088	2085	2123	2085
2	10	2085	2098	2090	2088	2095	2088	2085	2095
3	5	2085	2085	2085	2088	2090	2103	2085	2090

Temp: 1602 Nodes: 17

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	2098	2085	2088	2088	2120	2095	2085	2085
1	15	2090	2090	2085	2088	2123	2120	2120	2095
2	10	2090	2085	2085	2129	2090	2085	2123	2095
3	5	2088	2095	2088	2090	2085	2090	2090	2085

Temp: 1993.860178726469 Nodes: 17

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	2090	2090	2119	2153	2088	2090	2095	2088
1	15	2098	2095	2095	2103	2098	2085	2088	2151
2	10	2098	2088	2120	2088	2088	2085	2095	2090
3	5	2095	2088	2098	2095	2095	2098	2088	2090

Fig. 11: Graph 1 - gr17 results

Temp: 1184.7908000289458 Nodes: 24

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	1352	1355	1322	1362	1326	1358	1388	1341
1	15	1368	1359	1371	1428	1393	1368	1298	1372
2	10	1373	1341	1341	1393	1272	1326	1304	1327
3	5	1332	1366	1328	1339	1364	1367	1368	1319

Temp: 1026 Nodes: 24

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	1342	1336	1300	1367	1342	1316	1337	1335
1	15	1373	1351	1317	1343	1344	1363	1368	1357
2	10	1337	1388	1304	1397	1332	1359	1352	1286
3	5	1346	1405	1365	1326	1382	1367	1389	1349

Temp: 1343.5816000578916 Nodes: 24

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	1373	1386	1383	1350	1346	1345	1367	1358
1	15	1381	1418	1420	1272	1373	1398	1390	1365
2	10	1382	1347	1369	1365	1346	1326	1381	1341
3	5	1326	1361	1371	1316	1381	1318	1382	1335

Fig. 12: Graph 2 - gr24 results

Key Analysis (Simulated Annealing):

- Simulated Annealing performs extremely well for graph with nodes<50.
- Due to its heuristic nature, Simulated Annealing is time efficient when it comes to other methods.

Temp: 757.8644593152621 Nodes: 42

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	870	859	884	774	919	823	699	901
1	15	852	699	826	699	701	728	699	699
2	10	924	853	699	869	764	699	699	896
3	5	699	908	699	837	699	699	699	699

Temp: 687 Nodes: 42

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	823	874	699	844	918	850	699	875
1	15	834	786	814	875	699	924	861	881
2	10	842	860	699	844	784	699	699	699
3	5	833	888	734	699	699	699	699	701

Temp: 828.7289186305243 Nodes: 42

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20	883	880	850	899	855	850	874	867
1	15	843	835	882	868	699	893	930	699
2	10	843	855	859	913	699	816	825	699
3	5	838	867	699	699	699	706	699	725

Fig. 13: Graph 3 - dantzig42 results

Temp: 23177.877991073685 Nodes: 58

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	40	33003	31925	32525	33823	32637	33553	32765	33004
1	30	32626	33185	32804	31907	32187	32363	34273	32728
2	20	33666	32115	33777	32915	33054	34362	33788	32593
3	10	32973	32041	33451	32209	33426	33841	30952	32754

Temp: 21420 Nodes: 58

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	40	32033	34192	32643	34605	33727	32800	30662	30686
1	30	31480	31149	31779	32754	34003	33386	30328	34222
2	20	33382	33609	33133	31933	34211	29755	33938	32298
3	10	31062	33908	31225	33409	33169	33481	32321	33467

Temp: 24935.75598214737 Nodes: 58

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	40	31619	33370	33867	32466	35737	32316	31921	30784
1	30	31935	30376	34225	32922	35351	31741	34851	33843
2	20	31659	34989	30964	32062	32997	34801	31791	32242
3	10	31825	32207	31533	31229	33026	29708	33309	34473

Fig. 14: Graph 4 - brazil58 results

- When it comes to graphs with more number of nodes, the performance worsens.
- Simulated Annealing is very sensitive to temperature changes and a slight modification may result in major improvements.

Key Analysis (Firefly algorithm):

- Due to its heuristic nature, firefly is also one of the most efficient method to solve tsp .
- Firefly is very sensitive to beta changes and a slight modification may impact final result significantly .
- The Firefly Algorithm was applied to solve the Traveling Salesman Problem (TSP) on the ”EIL22” instance. The algorithm utilized a population size of 100 and a maximum of 3 iterations. With a beta value of 0.5and an alpha value of 1, the algorithm successfully found a solution with a total tour length of 283.277 units. This solution represents an optimal visit to all 22 cities in the TSP instance. The algorithm’s parameters and computational analysis led to the achievement of this solution.

Temp: 5122.033004874996 Nodes: 120

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	60	10826	11534	10895	10903	12098	10688	11616	10439
1	50	11472	11772	11077	11179	10644	11503	11729	10861
2	40	10953	10656	11352	11527	10522	12073	10872	11732
3	30	10996	10494	11239	11583	11543	10935	11563	11718
4	20	11689	10485	10889	11427	11607	10798	12218	11354
5	10	10915	11210	11434	10407	11348	10469	11784	11421

Temp: 4507 Nodes: 120

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	60	11924	11914	11712	11910	11989	10628	11676	10617
1	50	12160	9979	11429	10554	11550	11002	12178	11795
2	40	11381	11490	11432	11707	11859	11190	11652	12042
3	30	11209	11198	10587	10565	12015	11263	11372	10349
4	20	11461	11814	11541	11724	10986	11766	11142	12051
5	10	11166	11058	11893	11982	11879	11774	12114	11294

Temp: 5737.066009749991 Nodes: 120

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	60	11992	10457	11676	11442	11643	11294	10695	11813
1	50	11698	11881	10880	11103	10241	11391	11647	11542
2	40	12090	12501	11671	11493	10871	10841	11823	10529
3	30	11631	10738	12350	10884	11300	10437	10852	11585
4	20	10473	10407	11267	11684	11967	10997	10627	10767
5	10	10613	11304	11961	10746	11240	11663	11846	11320

Fig. 15: Graph 5 - gr120 results

Temp: 6009.820996382635 Nodes: 52

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	30.0	9129.48	9354.31	9374.59	9711.72	9566.28	9192.25	9210.86	9762.18
1	25.0	9215.85	8928.39	9226.0	9478.33	9508.81	9958.99	9654.13	9172.57
2	20.0	9738.48	9876.57	9789.93	9545.47	9384.65	9672.16	9768.47	9121.78
3	15.0	9148.84	9432.53	9911.47	9364.34	9406.96	9556.17	9594.83	9534.15
4	10.0	9748.78	9398.45	9291.7	9200.1	9215.62	9369.2	9479.92	9556.91
5	5.0	9075.92	9813.46	9012.73	9189.26	9347.29	8976.97	9620.51	9409.94

Temp: 5671.642883035263 Nodes: 52

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	30.0	9334.81	9414.27	9480.66	9921.51	9491.3	9515.9	9143.46	9320.62
1	25.0	9433.15	9347.22	9188.37	9304.03	9473.33	9557.24	9542.36	8991.47
2	20.0	9275.15	9066.01	9034.88	9128.01	9491.68	9529.66	10067.73	9108.88
3	15.0	9563.14	9574.37	9261.58	8991.14	9236.39	8920.03	9268.76	9834.62
4	10.0	9620.14	8909.91	9605.97	9484.93	9443.44	9081.98	9731.09	8884.45
5	5.0	9088.34	9467.55	9345.84	9033.25	9411.41	9739.45	9514.65	9698.9

Temp: 6347.999109730006 Nodes: 52

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	30.0	9513.77	8966.64	9595.46	9382.77	9704.91	8778.43	9050.16	9441.15
1	25.0	9423.5	9834.29	9371.57	9567.89	9725.37	9462.98	9552.57	9862.95
2	20.0	9224.08	9868.95	9519.07	9135.25	9284.49	9479.57	9477.72	9796.07
3	15.0	9595.65	8736.71	9311.92	9486.5	9575.18	8920.85	9697.84	9249.05
4	10.0	9524.37	9131.76	9603.38	9220.71	9889.94	9077.17	9981.23	8873.98
5	5.0	9566.29	9355.41	8883.78	8593.4	9822.55	9473.38	9581.3	9543.2

Fig. 17: Graph 7 - berlin52 results

Temp: 294.2284483935057 Nodes: 22

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	286.42	290.01	294.6	279.96	296.44	295.56	286.44	297.26
1	15.0	296.87	291.76	279.65	300.77	291.64	296.31	295.64	286.37
2	10.0	312.53	289.15	289.15	292.76	287.34	294.2	298.43	300.81
3	5.0	299.01	289.44	301.98	295.64	295.55	296.66	299.21	288.04

Temp: 254.67823685173272 Nodes: 22

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	285.84	287.76	297.35	300.74	296.87	288.58	294.41	288.58
1	15.0	300.32	289.79	299.21	302.07	294.6	296.38	299.12	294.96
2	10.0	286.55	300.34	310.62	297.88	290.22	286.37	299.13	292.94
3	5.0	297.79	296.58	302.08	298.3	286.02	303.39	299.48	297.28

Temp: 333.77865993527865 Nodes: 22

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	284.9	292.76	302.44	293.25	306.15	285.49	297.79	306.47
1	15.0	295.56	299.89	285.91	289.83	298.47	288.01	295.9	292.56
2	10.0	296.18	302.95	288.01	292.97	282.67	279.65	302.54	288.49
3	5.0	282.96	288.86	279.65	287.73	310.38	307.59	294.36	301.53

Fig. 16: Graph 6 - eil22 results

Temp: 957.3749332210649 Nodes: 99

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	2018.66	1839.82	1812.33	1907.04	1873.51	1888.43	1948.76	1891.51
1	15.0	1908.89	1784.37	2003.58	1938.5	1951.21	1919.42	1981.06	1718.7
2	10.0	1921.75	1860.56	1865.36	1911.08	1713.99	1797.55	1776.75	1885.24
3	5.0	1861.13	1720.4	1739.14	1854.42	1852.32	1823.04	1654.49	1838.2

Temp: 809.5349602101148 Nodes: 99

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	1748.36	2098.79	1877.93	1873.63	1875.88	1840.81	1867.1	1835.82
1	15.0	1913.42	1874.12	1916.97	1706.49	2004.96	1799.25	1775.2	1739.37
2	10.0	1817.54	1766.26	1705.29	1803.34	1796.15	1715.14	1772.05	1826.94
3	5.0	1833.61	1858.38	1660.38	1798.54	1830.86	1842.49	1761.5	1741.3

Temp: 1105.214906232015 Nodes: 99

	Neighbours	0.95	0.9	0.85	0.8	0.75	0.7	0.65	0.6
0	20.0	1901.88	1954.19	1831.72	1802.04	1764.29	1832.21	1803.15	1896.42
1	15.0	1858.48	1710.17	1986.75	1866.66	1959.8	1926.06	1844.56	1913.36
2	10.0	1858.13	1915.14	1691.4	1804.79	1856.47	1896.14	1776.64	1829.06
3	5.0	1925.02	1899.21	1748.58	1754.01	1711.5	1819.86	1772.96	1761.64

Fig. 18: Graph 8 - rat99 results

- The Firefly Algorithm was applied to the Berlin52 instance of the Traveling Salesman Problem (TSP) with 52 cities. With a population size of 200, the algorithm iterated for a maximum of 30 cycles. The beta value was set to 0.8 to control the attractiveness between fireflies during the search process.Through extensive computational analysis, the Firefly Algorithm successfully obtained a solution with an optimal cost of 8182.19155 for the Berlin52 TSP instance within the 30 iterations
- The Firefly Algorithm was employed with a beta value of 0.8 to solve the Traveling Salesman Problem (TSP) instance known as ”rat99,” comprising 99 cities. To enhance the algorithm’s performance, various improvements were implemented, such as a population size of 300 and a maximum of 50 iterations. Additionally, the heuristic function was refined to provide more effective guidance to the fireflies in their search for an optimal solution. By considering factors such as distance, pheromone trails, and local information, the modified heuristic function aimed to guide the fireflies towards better solutions. Through extensive computational analysis, the algorithm successfully obtained an optimal solution with a cost of 2628.56 for the rat99 TSP problem. This remarkable improvement underscores the efficacy of the adapted Firefly Algorithm and its potential to deliver superior solutions when tackling the complexities of the TSP.
- During the observation of parameter tuning for problems with a large number of nodes, it was noticed that finding optimal solutions became increasingly challenging. However, one notable observation was that by increasing the population size, it became possible to approximate near-optimal solutions even for problems with an even

greater number of nodes. Although this approach showed promise, it came at the cost of increased computational time. The trade-off between solution quality and computational time became more pronounced as the population size grew. Therefore, finding an appropriate balance between population size and computational resources was crucial in order to achieve satisfactory results for larger-scale problems.

- Also during the observation of parameter tuning for problems with a larger number of nodes, an interesting finding emerged: the parameters that were successfully tuned for the larger problems also proved effective for smaller-scale problems. This suggested that the same set of parameters could be utilized across a range of problem sizes, eliminating the need for extensive parameter tuning for each specific instance. However, it should be noted that the effectiveness of the parameters was strongly influenced by the heuristic function employed. The heuristic function played a significant role in guiding the search process and influencing the algorithm's performance. Therefore, ensuring a well-designed and adaptable heuristic function was crucial in achieving favorable results across different problem sizes.

The following tables will show the result of firefly on different tsp problems :

Problem	Number of Cities	Population Size	Iterations	Beta Value	Optimal Cost
rat99	99	300	50	0.8	2628.56
berlin52	52	200	30	0.8	8182.19155
eil22	22	100	30	0.7	283.277

Fig. 19: Firefly Results

following graphs were obtained :

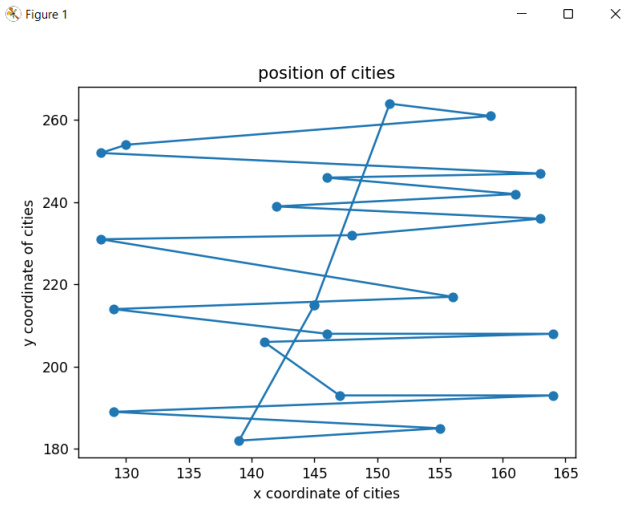

Fig. 20: eil22 initial solution

A. Discussion and Complexity Analysis

Dynamic Programming

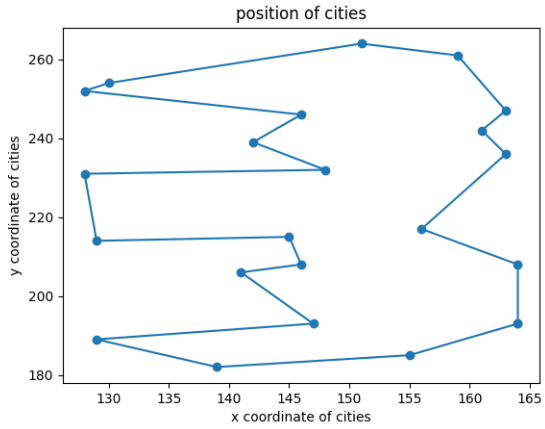

Fig. 21: eil22 results

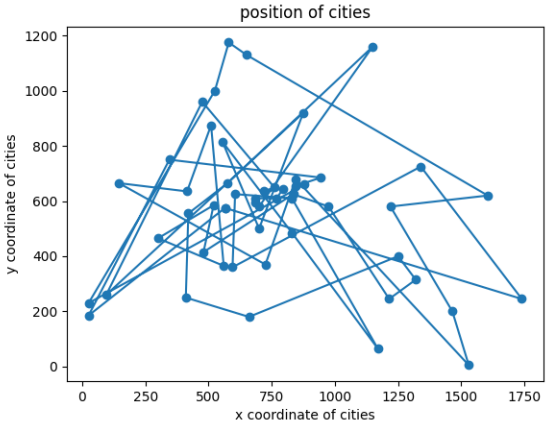

Fig. 22: Berlin52 initial solution

- Time Complexity: The time complexity of the solution is O($n^2 * 2^n$), where O($n * 2^n$) represents the maximum number of unique subproblems or states, and O(n) accounts for the transition that occurs in each state through the for loop in the code.
- Space Complexity: Auxiliary space complexity for this solution is O($n * 2^n$), where n represents the number of nodes or cities. To explain the approach in simpler terms, for a set of size n, we examine n-2 subsets, each consisting of n-1 elements, without including the nth element. By utilizing the recurrence relation mentioned earlier, we can formulate a solution based on dynamic programming.

Simulated Annealing

- Time Complexity: The time complexity of simulated annealing is typically measured in terms of the number of iterations or the number of function evaluations performed by the algorithm in order to reach an acceptable solution. In the analysis mentioned, it is assumed that the optimal temperature is known and it takes 100 iterations at that temperature for the algorithm to converge.

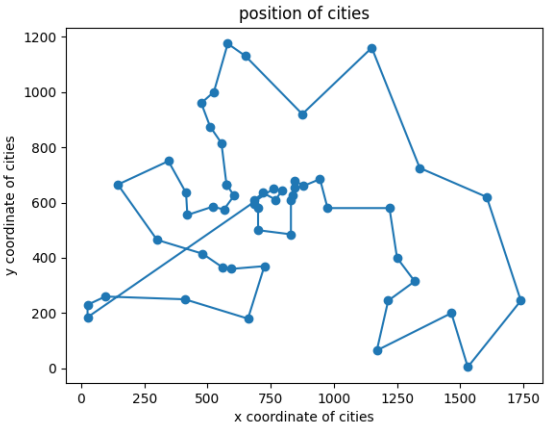

Fig. 23: Berlin52 results

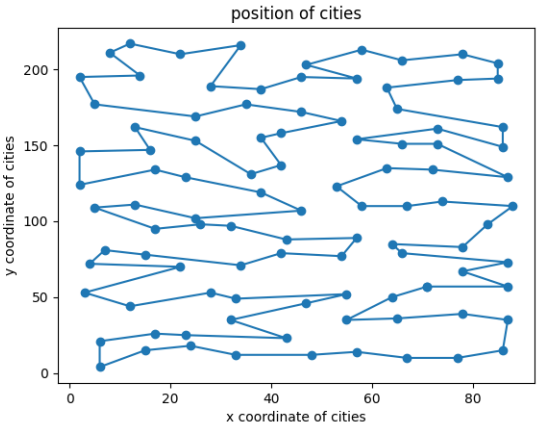

Fig. 25: rat99 results

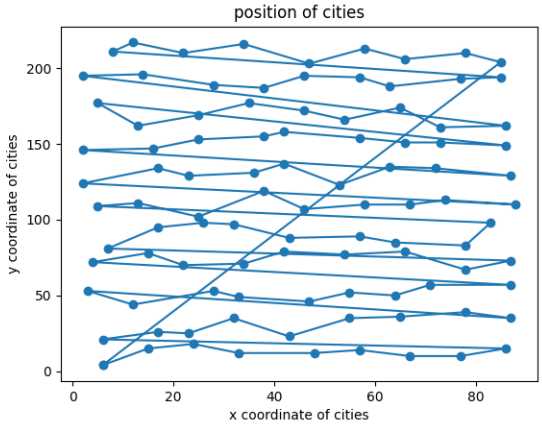

Fig. 24: Berlin52 initial solution

Name	Time in seconds
Graph1 - gr17	3.55
Graph2 - gr24	905.65
Graph3 - dantzig42	72,70,74,616
Graph4 - brazil58	9.08 * 10^12
Graph5 - gr120	1.79 * 10^33
Graph6 - eil22	190.25
Graph7 - berlin52	11,41,26,64,37,861
Graph8 - rat99	5.821 * 10^26

Fig. 26: Time required for each graph - Dynamic Programming

- Space Complexity: Space complexity for Simulated Annealing is majorly depend on graph size and increases with the same.

Firefly Algorithm

- Time Complexity: The time complexity of the Firefly Algorithm for TSP depends on various factors, including the number of fireflies, the number of iterations, the complexity of the local search heuristics, and the objective function evaluation. The main iteration loop of the algorithm updates the attractiveness and movement of each firefly. Since all fireflies need to be considered for each update, the time complexity of the main loop is proportional to the square of the number of fireflies. The local search heuristics, such as 2-Opt, nearest neighbor, or nearest insertion, also contribute to the overall time complexity. The complexity of these heuristics depends on the size of the TSP problem instance, usually resulting in quadratic or cubic time complexity. Evaluating the objective function involves calculating the total tour distance. While the individual calculations have constant time complexity, they are performed for all city pairs, resulting in a quadratic time complexity. Considering these factors, the overall time complexity of the Firefly Algorithm for TSP is approximately proportional to the number of iterations multiplied by the square of the number of fireflies.
- Space Complexity: The space complexity of the Firefly Algorithm for TSP is determined by the storage requirements for the firefly population, the adjacency matrix, and auxiliary variables. The firefly population needs storage for the position of each firefly in the solution space. The space required for storing the firefly population is proportional to the number of fireflies. The adjacency matrix represents the TSP graph and requires storage for the distances between each pair of cities. The space required for the adjacency matrix is proportional to the square of the number of cities. Other auxiliary variables, such as attractiveness values, movement vectors, and temporary variables, have negligible impact on the overall space complexity. Considering these factors, the overall space complexity of the Firefly Algorithm for TSP is proportional to the square of the number of cities.no

Name	Time in seconds
Graph1 - gr17	2.4375
Graph2 - gr24	4.6875
Graph3 - dantzig42	21.66
Graph4 - brazil58	70
Graph5 - gr120	583
Graph6 - eil22	2.5625
Graph7 - berlin52	46.875
Graph8 - rat99	250

Fig. 27: Time in second required for 100 iterations - Simulated Annealing

VII. Conclusion

In conclusion, it is clear that there is no one method suitable to solve TSP and each method has its drawback and advantage. DP is suitable for solving graphs with less number of nodes but the complexity and computational power makes it unfeasible for problems with more number of nodes. Simulated annealing presents near optimal or sometimes even optimal results for nodes less than 50. But with higher nodes, the temperature related sensitivity is very high and the process to calculate temperature is still and area of research. The approach employed, which uses iterative method is computationally expensive in itself. Further research in the area of temperature, with specific focus on stopping and initial temperature is suggested.In conclusion, the Firefly Algorithm shows promise for solving problems with larger numbers of nodes. Parameter tuning for larger problems can also yield effective results for smaller-scale problems, showcasing the algorithm's versatility. However, it is important to note that computational time may increase when dealing with larger problem sizes. Finding a balance between solution quality and computational resources becomes crucial. With careful parameter selection and consideration of the problem's characteristics, the Firefly Algorithm can be a valuable tool for addressing complex optimization problems with a significant number of nodes

Individual contribution

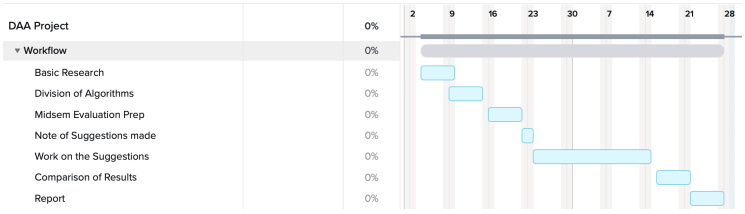

Fig. 28: Gantt Chart

- Anushka Agrawal: Dynamic Programming approach to solve TSP.
- Satyam Agrawal: Simulated Annealing approach to solve TSP.
- Gulshan Goyal: Firefly Algorithm approach to solve TSP.

List of Figures

REFERENCES

[1] Walid Ben-Ameur. "Computing the Initial Temperature of Simulated Annealing". In: *Computational Optimization and Applications* 29 (Dec. 2004), pp. 369–385. DOI: 10.1023/B:COAP.0000044187.23143.bd.

[2] Chetan Chauhan, Ravindra Gupta, and Kshitij Pathak. "Survey of Methods of Solving TSP along with its Implementation using Dynamic Programming Approach". In: *International Journal of Computer Applications* 52 (Aug. 2012), pp. 12–19. DOI: 10.5120/8189-1550.

[3] Xin She Yang and Xingshi He. "Firefly algorithm: recent advances and applications". In: *International Journal of Swarm Intelligence* 1.1 (2013), p. 36. DOI: 10.1504/ijsi.2013.055801. URL: https://doi.org/10.1504%2Fijsi.2013.055801.