Solving the Travelling Salesman Problem using Optimization techniques

Arhita Kundu, 210102021

I. Introduction

The Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem that describes a scenario where a salesman needs to visit a set of cities, each connected by a distance or time. The objective is to find the shortest possible route that allows the salesman to visit each city exactly once and return to the starting point. While the problem is easy to explain, its solution becomes increasingly challenging as the number of cities grows.

TSP is a permutation problem represented as an undirected weighted graph, where cities are vertices, paths are edges, and the distance between cities determines the edge length. The goal is to minimize the total distance traveled, starting and finishing at a specified vertex after visiting each city exactly once.

As the number of cities increases, the exhaustive approach of considering all possible routes becomes inefficient due to its escalating complexity. TSP is recognized as an NP-hard problem, implying that finding an exact solution in polynomial time is impractical. Various heuristic algorithms have been developed in operations research to tackle TSP efficiently.

The problem finds applications in diverse fields such as military and traffic management. Despite its NP-hard nature, TSP can be effectively solved using heuristic algorithms like the Genetic Algorithm. The Genetic Algorithm's flexibility and robustness make it well-suited for addressing TSP and similar optimization challenges.

Distinctive applications of TSP include vehicle routing for minimum time or fuel consumption, in real world scenarios like businesses shipments and deliveries, transactions, computer wiring, cutting wallpaper, job sequencing, and more. In summary, TSP poses a fundamental challenge in combinatorial optimization, and its resolution is crucial in optimizing routes and sequences in various practical scenarios. Here, we intend to discuss and understand some optimisation algorithms to solve the infamous TSP.

II. 1. Genetic Algorithm

Genetic algorithms, introduced by John Holland in the 1970s and gaining popularity in the late 1980s, are optimization techniques inspired by Darwin's principle of natural selection. The fundamental idea is to "select the best and discard the rest," mirroring the survival of the fittest in a population of species.

The emphasis on these steps highlights the algorithm's ability to simulate the evolutionary process, leading to the gradual refinement of solutions and ultimately yielding an optimal solution. This iterative and adaptive nature makes genetic algorithms effective in solving complex optimization problems.

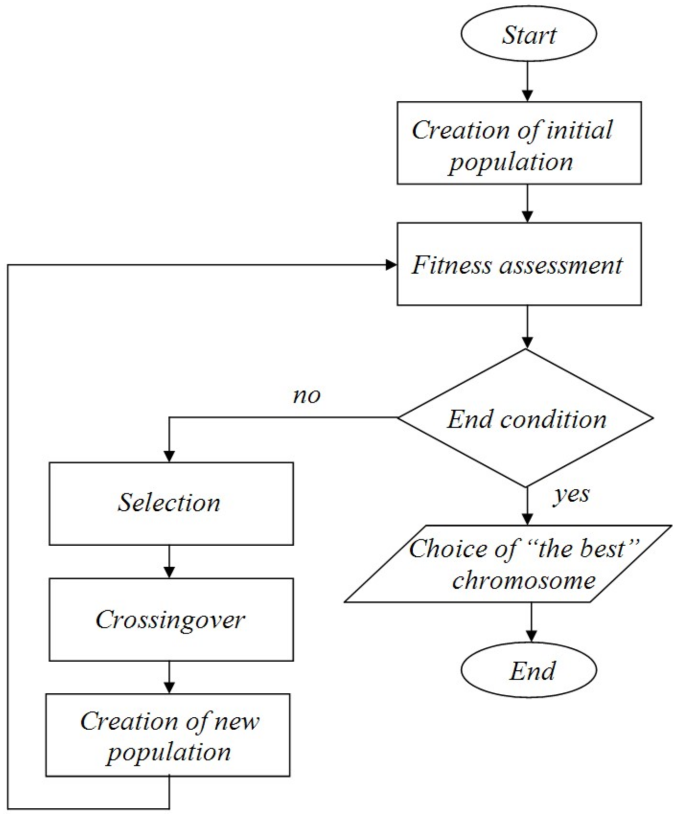

Start
Creation of initial population
Fitness assessment
End condition
no
yes
Selection
Crossingover
Creation of new population
Choice of "the best" chromosome
End

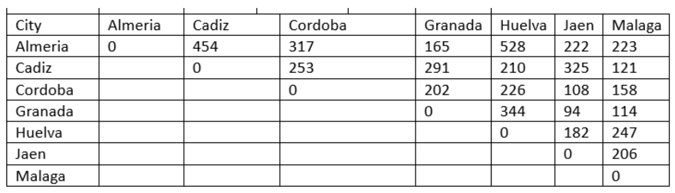
Fig. 1. Basic flowchart for Genetic Algorithm

A. Implementation

1) Encoding: We select 7 cities Almeria, Cadiz, Cordoba, Granada, Huelva, Jaen, Malaga, Sevilla and we encode them from indexes 0 to 6, in the respective order above.The distance-matrix is symmetric, therefore, the part above the main diagonal contains all necessary information.

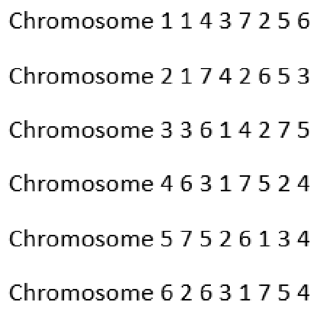
2) Fitness Function: The fitness function in the Travelling Salesman Problem evaluates the quality of a chromosome based on its total tour cost. This cost is determined by summing the distances of each travel segment in the chromosome. A lower total distance indicates a fitter solution, making the chromosome more favorable.

3) Selection: The selection process, aimed at choosing chromosomes with lower fitness values, employs tournament



City	Almeria	Cadiz	Cordoba	Granada	Huelva	Jaen	Malaga
Almeria	0	454	317	165	528	222	223
Cadiz		0	253	291	210	325	121
Cordoba			0	202	226	108	158
Granada				0	344	94	114
Huelva					0	182	247
Jaen						0	206
Malaga							0

Fig. 2. Distance table



Chromosome 1 1 4 3 7 2 5 6

Chromosome 2 1 7 4 2 6 5 3

Chromosome 3 3 6 1 4 2 7 5

Chromosome 4 6 3 1 7 5 2 4

Chromosome 5 7 5 2 6 1 3 4

Chromosome 6 2 6 3 1 7 5 4

Fig. 3.

selection. In this method, pairwise tournaments are conducted between two solutions, and the superior solution is selected for the mating pool. Subsequently, two additional solutions engage in another tournament, with the better-performing solution filling another slot in the mating pool.

4) Crossover: While single point crossover method randomly selects a crossover point in the string and swaps the substrings, in TSP single crossover method is not used, as it may produce some invalid offspring. We use a form of crossover known as partially mapped crossover (PMX). PMX is a technique used in genetic algorithms for recombination of two parent solutions to create two offspring. Repeated genes are addressed in the offspring by replacing them with genes from the corresponding parent, ensuring that each gene is unique in the child. A random position (pos) is selected to determine the crossover point. The code chooses a random index between 1 and (chromosome length-1). The offspring (child1 and child2) are generated by combining parts of the parents based on the crossover point.

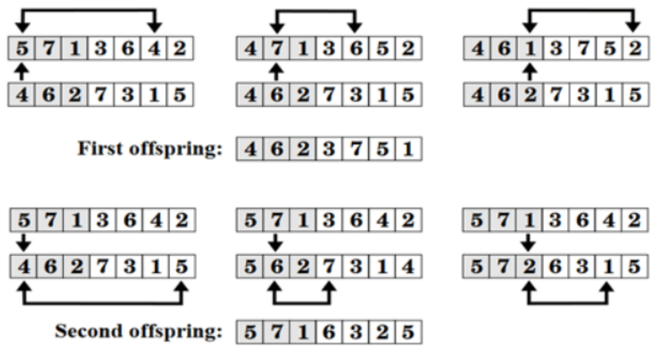


Fig. 4. PMX Crossover

5) Mutation: We use Inversion mutation here, which is suitable for problems where the order of genes in the chromosome significantly affects the solution quality. For the Traveling Salesman Problem (TSP), where the order of cities in the tour is crucial, it has the effect of reversing a segment of the chromosome, which can be useful for exploring different neighborhoods in the solution space, thus improving the Genetic algorithm.

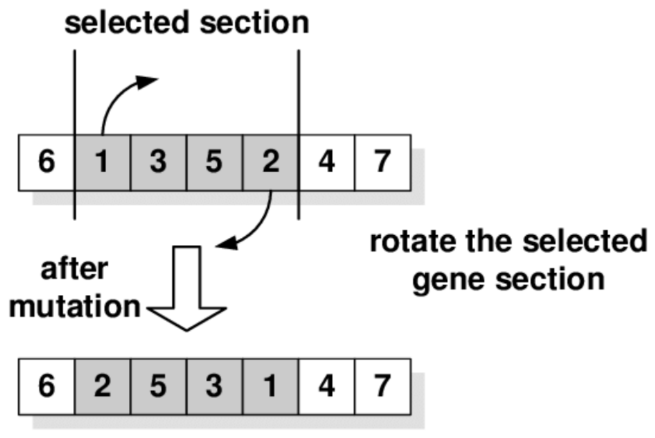


Fig. 5. Inversion mutation

B. Pseudocode

```
1. Initialize Population:
- Generate a set of random routes connecting cities.
2. Evaluate Fitness:
- Calculate the total distance of each route.
- Shorter distances indicate better fitness.
3. Select Parents:
- Choose pairs of routes based on their fitness.
- Better routes have a higher chance of being selected.
4. Crossover:
- Combine pairs of routes to create new routes (offspring).
- Mimic the process of genetic crossover.
5. Mutation:
- Introduce small random changes in some routes.
- Mimic genetic mutations for diversity.
6. Evaluate Offspring:
- Calculate the fitness of the new routes.
7. Select Survivors:
- Keep the best routes from both the current and new populations.
8. Repeat:
- Repeat steps 3-7 for a number of iterations (generations).
9. Output:
- The route with the best fitness after several iterations is the solution to the problem.
```

C. Result and Conclusion

The modified GA prints the best chromosome, its decoded genotype, its fitness value, and the number of generations it has been part of the population.The age mechanism adds an interesting dimension to the GA. By considering the age of chromosomes, it might help in preserving diversity and preventing premature convergence. The gene repeat method in crossover addresses repeated genes, potentially enhancing the diversity of the population. Hence, the modified GA with the age mechanism and the custom crossover has the potential to provide better exploration-exploitation balance.The custom crossover method may contribute to better genetic diversity.

The modified GA code achieved a an optimal solution of 1143 in 0.456 sec compared to the original GA code which achieves an optimal solution about path 1-4-2-3-5-0-6 in 0.927 sec in 12 iterations. This is a significant improvement in the run time of the algorithm as well as the optimal path cost function.

III. 2. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a nature-inspired optimization algorithm that simulates the social behavior of particles in a population seeking an optimal solution in a solution space. Each particle represents a potential solution and has a position and velocity. The algorithm iteratively refines these solutions by updating velocities based on the particles' own experience and the best solution found by any particle in the population. Positions are then updated accordingly. PSO maintains a personal best position for each particle and a global best position for the entire population. The fitness of each particle is evaluated based on an objective function, and the algorithm seeks to minimize or maximize this function. Through this iterative process of updating velocities and positions, PSO aims to converge towards the optimal solution for the given optimization problem.

PSO begins by initializing a population of particles (X_i), where each particle (X_i) represents a complete candidate solution for the Traveling Salesman Problem (TSP) and is typically generated randomly. For instance, in the context of a 5-city TSP (with $N = 5$) and an initial population size of 5, a set of possible initial population particles could be randomly generated as follows: X_1 representing (3-5-4-2-1-3), X_2 as (4-2-1-3-5-4), X_3 as (1-3-5-4-2-1), X_4 as (1-2-3-4-5-1), and X_5 as (2-1-4-5-3-2). Each particle in the population serves as a potential solution to the TSP, with its sequence of cities defined by the permutation of numbers.

At the beginning, each particle is assigned a random velocity V_i, where velocity is expressed as a Swap Sequence (SS) consisting of several Swap Operators (SO). A Swap Operator, for instance $SO_{2,5}$, denotes swapping the positions 2 and 5 within a particle. For instance, if $SO_{2,5}$ operates on the first particle X_1, the resulting particle would be $X_1 \oplus SO_{2,5} = (3, 1, 4, 2, 5, 3)$, where the symbol $\oplus$ signifies the application of the swap operator on the particle, causing it to move to its new position. A Swap Sequence is formed by combining two or more swap operators, representing the velocity for a particle to move accordingly.

A crucial aspect of Particle Swarm Optimization (PSO) involves updating the velocities of particles in each iteration to guide them toward the optimal solution. This update is calculated using the formula:

$$V_i^{(t)} = V_i^{(t-1)} \oplus (\beta \cdot P_{i \to X_i^{(t-1)}}) \oplus (\gamma \cdot G \to X_i^{(t-1)})$$

Here: - $V_i^{(t)}$ is the velocity of particle i at iteration t, - $V_i^{(t-1)}$ is the previous velocity of particle i, - $P_{i \to X_i^{(t-1)}}$ represents the personal best of particle i at the previous iteration guiding it towards its best-known solution, - $G \to X_i^{(t-1)}$ is the global best known solution across all particles at the previous iteration, - β and γ are scaling factors that control the impact of the personal best and global best components.

This velocity update equation combines the particle's previous velocity with adjustments based on its own historical best (P_i) and the best solution across all particles (G), aiming to steer the particle towards promising regions in the solution space. When generating the initial population of particles or updating a particle's position, the conventional approach is to consider any feasible solution involving all nodes or cities, regardless of the sequence of cities or nodes. However, this approach has a notable limitation. It becomes evident and easily noticeable that solutions like (3-5-4-2-1-3), (4-2-1-3-5-4), and (1-3-5-4-2-1) are technically identical, differing only in the starting node. Despite this similarity, their fitness or objective function values remain exactly the same. We address this shortcoming of the PSO by generating only unique permutations for the population, thus reducing computational power and time by some extent. Furthermore, the inertia weight (w) significantly influences the behavior of global search, particularly in terms of convergence. A lower w value accelerates the convergence towards the global optimum, while a higher w value facilitates exploration across the entire search space. To achieve an improved global search capability in the initial iterations and enhance local exploitation later on, the inertia weight (w) undergoes dynamic adjustment with a linear descent over iterations.

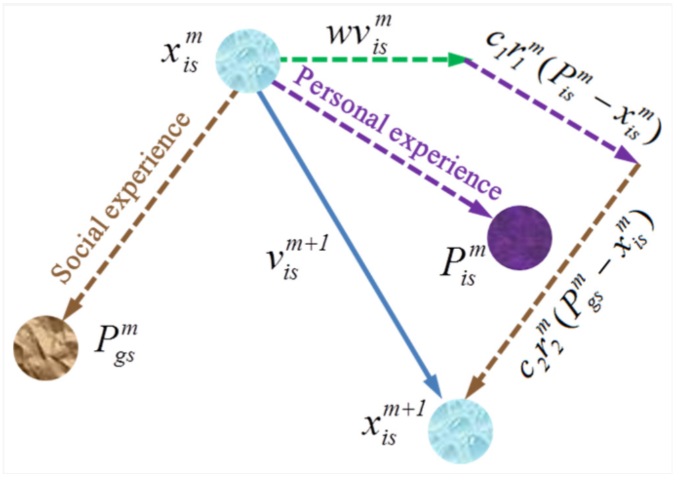


Fig. 6. Basic PSO representation

$$w = w_{\max} - m \times \frac{(w_{\max} - w_{\min})}{M}$$

Here, $w_{\max}$ and $w_{\min}$ denote the maximum and minimum values of the inertia weight (w), respectively.

We also update the algorithm carefully to handle dynamic TSP cases, cases when weights or distances continuously change over iterations

We also implement a Variable Neighborhood Search, which involves changing the neighborhood structure during the optimization process. This can help escape local optima and explore the solution space more effectively. In the context of the Traveling Salesman Problem (TSP), neighborhoods typically refer to different ways of rearranging the order of cities in a solution. A random neighbor is generated by swapping two random cities in the current solution. The VNS process is applied after updating the particle position using the velocity. The number of neighborhood changes is limited by a parameter that is controlled by us.

A. *Pseudocode*

```
1. Initialization:
- Generate a population of particles, each representing a random solution to the TSP.
- Assign random velocities to particles.
2. Objective Function Evaluation:
- Evaluate the fitness of each particle using the TSP objective function.
3. Global Best Initialization:
- Identify the particle with the best fitness as the global best.
4. PSO Iterations:
- For each iteration:
- Update the inertia weight (w) dynamically using a linear descent formula.
- For each particle:
- Evaluate the fitness of the current solution.
- Update the personal best if the current solution is better.
- Update the global best if the current solution is better than the global best.
- Update the particle's velocity using the PSO update equation.
- Generate a new particle position by applying the velocity as a swap sequence.
5. Dynamic TSP Part:
- Use a dynamic distance matrix that changes every few iterations to simulate a dynamic TSP scenario.
6. Unique Permutations:
- Ensure that each particle's position is a unique permutation of cities to avoid redundant solutions.
7. Inertial Weight (w) Update:
    $w = w_{\max} - m \times \frac{(w_{\max} - w_{\min})}{M}$
Here, $w_{\max}$ and $w_{\min}$ denote the maximum and minimum values of w
8. Variable Neighborhood Search (VNS):
- Optionally, integrate Variable Neighborhood Search for further exploration and exploitation.
9. Result Extraction:
- Obtain the best solution and its total distance from the global best particle.
```

B. *Result*

The optimal solution achieved by the modified PSO algorithm is 1143 in 0.0472 sec in 12 iterations compared to a run time 0.0871 sec taken by the original algorithm.The use of a dynamic TSP scenario, with a changing distance matrix, introduces realism to the optimization process, making the algorithm adaptable to dynamic environments. The incorporation of Unique Permutations safeguards against redundant solutions, promoting diversity within the population. The optional integration of Variable Neighborhood Search (VNS) further enhances exploration, allowing the algorithm to explore different solution spaces.

IV. 3. Ant Colony Optimization

Ant colony optimization (ACO) is a population-based metaheuristic for combinatorial optimization problems. It is inspired by the ability of ants to find the shortest path between their nest and a source of food. Marco Dorigo first introduced ACO in his Ph.D. thesis and applied it to the Traveling Salesman Problem (TSP). It has been applied to the quadratic assignment problem, the vehicle routing problem, bin packing, stock cutting, and RNA secondary structure prediction.

ACO mimics the way real ants find the shortest path between their nest and food sources. The algorithm involves the collaborative effort of artificial ants, which construct solutions by moving from city to city in a probabilistic manner.In the ACO process for TSP, a colony of artificial ants is initially distributed randomly across cities. Each ant iteratively chooses the next city to visit based on a combination of pheromone levels on the edges connecting cities and the distance between them. A probabilistic model, often represented by the formula:

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{u \in allowe\ d_k}[\tau_{iu}(t)]^{\alpha}[\eta_{iu}]^{\beta}} & j \in allowe\,d_k \\ 0 & \text{Otherwise} \end{cases}$$

Fig. 7. Formula

Where pijkt is the probability of the ant passing from node i to node j, ijt is the pheromone value between nodes i and j, ij(t) is the heuristic factor between nodes i and j at t moment, ij= 1/dij, dij is the distance between nodes i and j and , and are two adjustable positive parameters that control the relative weights of the edge pheromone trails and of the heuristic visibility.

A. *Modifications*

1.A separate class introduces a structured representation for ants, each tracking its path and cost. This allows for easy

comparison and sorting based on cost, a crucial factor in selecting the best solution.
2.Instead of a single ant per city, the modification creates multiple ants for each city, enhancing exploration. This diversification increases the chances of discovering optimal or near-optimal paths.
3. Pheromone initialization now considers infinite distances, preventing undefined values. This ensures a proper start for the pheromone matrix, setting the stage for effective learning.
4.The ’turn’ function orchestrates the iteration process, updating both ant paths and pheromone levels. This organized control improves the overall efficiency of the algorithm.
5.Ants reaching the final step of the TSP are removed, preventing redundant exploration. This helps maintain a diverse and focused ant population, contributing to exploration efficiency. It also determines the next city for an ant to move based on pheromone levels and visibility. It also handles cases where an ant cannot move, avoiding dead-end exploration and ensuring a continuous search.
6.Pheromone levels on edges are updated based on ant paths, and a decay factor (rho) is introduced for realistic evaporation. This modification helps maintain pheromone diversity, preventing premature convergence to suboptimal solutions.
7.We select the ant with the lowest cost among those starting from a specific city. This ensures that the best solutions are retained and considered for further exploration.
8.Next possible paths are generated based on whether an ant is in the final step or not. This refinement improves path generation, facilitating efficient exploration of the solution space.

B. Pseudocode

```
Ant at City A:
1. Calculate probabilities for moving to neighboring cities (B, C, D).
2. Choose the next city based on these probabilities (e.g., B).
3. Update the ant’s path to include the chosen city (A $\rightarrow$ B).
4. Update the ant’s cost based on the distance from A to B.
```

C. Result and Conclusion

We extend the basic Ant Colony Optimization (ACO) algorithm for the Traveling Salesman Problem (TSP). Each city now hosts multiple ants, enhancing exploration. Pheromone initialization accounts for infinite distances, and decay is introduced for realistic evaporation. The algorithm intelligently guides ants to choose the next city based on pheromone levels and visibility, promoting diverse exploration. The update mechanism refines both ant paths and pheromone levels, ensuring efficient convergence. The identification of the best ant starting from a specific city aids in selecting the optimal solution. These modifications collectively improve the ACO algorithm’s performance in finding high-quality solutions for the TSP.

The original algorithm achieves an optimal cost of 1149 in 0.08 sec in 6 iterations whereas the modified algorithm does the same in under 0.03 sec in each iteration. Thus a significant improvement is seen in run time as well as number of iterations required to converge to optimal solution.

V. Branch and Bound

Branch and Bound for the Traveling Salesman Problem (TSP) involves systematically exploring the solution space to find an optimal tour. The algorithm begins by initializing the best cost and best path variables. It then employs a recursive approach, systematically considering potential tours by branching at each decision point. To enhance efficiency, the algorithm prunes branches early by evaluating a lower bound for each partial tour and comparing it with the current best cost. If a partial tour cannot lead to a better solution, the branch is pruned. The algorithm continues this exploration until all cities are visited, updating the best solution whenever a lower-cost tour is found. The final output is the best path and its corresponding cost, representing an optimized solution to the TSP. The basic idea behind Branch and Bound is to systematically search the solution space, eliminating branches that cannot lead to an optimal solution.

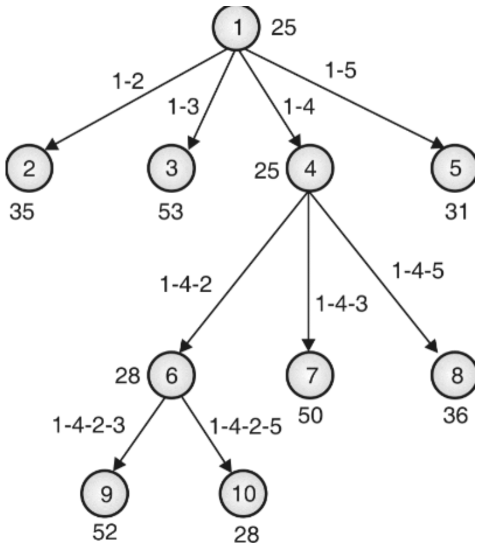


Fig. 8. Basic Branch and Bound for TSP

A. Modifications to basic Branch and Bound

1. **Initial Bound Calculation (Heuristic - Greedy Algorithm):**
- Modification: The initial bound is calculated using a greedy algorithm to obtain a lower bound on the cost of the solution.
- Impact: This heuristic provides a quick and reasonably good estimate of the optimal solution, potentially pruning branches early if they cannot lead to a better solution.

2. **Matrix Reduction for Minimum Spanning Tree (MST):**
- Modification: The matrix reduction step is applied during

the construction of the Minimum Spanning Tree (MST). This involves reducing rows and columns to eliminate redundant paths.
- Impact: Reducing the matrix helps to lower the complexity of the subsequent computations, contributing to more efficient exploration of the solution space.

3. **Update BSSF During Search:**
- Modification: The Best-So-Far Solution (BSSF) is updated whenever a new feasible solution is found during the search.
- Impact: This ensures that the algorithm returns the best solution found within the time limit, even if the optimal solution is not reached.

4. **Pruning Unpromising Paths:**
- Modification: Pruning is applied to paths with bounds exceeding the current BSSF, avoiding further exploration of unpromising routes.
-Impact: Reduces the number of states explored and improves the overall efficiency of the algorithm.

5. **Priority Queue Implementation:**
- Modification: A priority queue is used to manage the states during the search, ensuring that states with lower bounds are explored first.
- Impact: Improves the efficiency of state exploration, allowing the algorithm to focus on more promising paths and potentially reducing the search space.

6. **Efficient State Representation:**
- Modification: The state is represented efficiently, including the bound, route, reduced matrix, and depth.
- Impact: Enables faster comparisons and updates during the search process, contributing to overall algorithm efficiency.

7. **Optimizing Minimum Weight Matching:**
- Modification: The minimum weight matching algorithm is optimized to handle invalid city connections efficiently.
- Impact: Ensures that the matching step contributes positively to the construction of the solution, even in the presence of invalid connections.

8. **Eulerian Tour Construction:**
- Modification: An Eulerian tour is constructed efficiently from the minimum weight matching, ensuring a valid tour is obtained.
- Impact: Provides a complete circuit for the TSP, forming the basis for the final solution.

These modifications collectively enhance the performance of the branch-and-bound algorithm for TSP by incorporating heuristics, efficient data structures, and pruning strategies, ultimately leading to a more effective exploration of the solution space within the given time constraints.

B. *Pseudocode*

```
1. Initialize priority queue for states.
- Calculate initial lower bound using a heuristic.
- Set initial BSSF as infinity.
- Create a matrix for pairwise distances.

2. Apply row and column reduction to the distance matrix.
- Update the initial lower bound based on reductions.

3. Run a greedy algorithm to obtain an initial feasible
solution.
- Update the BSSF if a better solution is found.

4. Enqueue initial states for each city, considering the
reduced matrix and initial bound.
- Initialize counters for states created, pruned, and maximum
queue size.

5. While there are states in the priority queue and within
the time limit:
- Dequeue the state with the lowest bound.
- Check if the state is a potential solution:
- If yes, update BSSF and relevant statistics.
- Expand the state by considering valid next cities:
- Calculate bounds and enqueue new states.
- Update counters and continue the search loop.
```

C. *Result*

The modified Branch and Bound algorithm for TSP demonstrates a successful fusion of heuristic techniques and algorithmic optimizations, resulting in a robust and efficient solution approach. The incorporation of greedy algorithms, matrix reduction, and pruning mechanisms significantly accelerates the search process, enabling the algorithm to handle larger problem instances within practical time constraints. The constant update of the BSSF ensures that the algorithm returns the best-known solution, providing a balance between optimality and computational efficiency.

The updated solution achieves an optimal solution of minimum cost 1149 in 15 iterations with a processing time of 0.9179 sec among a total of 5102 solutions found, out of which 3040 states are pruned. The original algorithm has a processing time of 0.07 sec for 100 csolutions altogether, which assures a significant improvement in computational time, of almost 4 times.

References

[1] Alka Singh and Rajnesh Singh ”Exploring Travelling Salesman Problem using Genetic Algorithm”,IJERT,Vol. 3 Issue 2, February - 2014

[2] Junfeng Xin,Shixin Li,Jinlu Sheng,Yongbo Zhang,Ying Cui-Application of Improved Particle Swarm Optimization for Navigation of Unmanned Surface Vehicles,13 July 2019

[3] A. K. M. Foysal Ahmed and Ji Ung Sun-An Improved Particle Swarm Optimization Algorithm for the Travelling Salesman Problem