A PROJECT REPORT

On

TRAVELLING SALESMAN PROBLEM

Submitted in the requirements for the lab project for the course of B.Tech in CSE

Subject: Design and Analysis of Algorithms Lab (BCS- 204)

Submitted By:

HIMANSHI - 06901012022
ISHIKA CHHOKER- 07301012022
ADITI KASHYAP - 01201012022
BHUVI SINGH – 05201012022
ISHRAT KHAN-07601012022

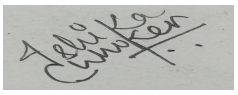
Under the guidance of
Dr. Vivekanand Jha
Associate Professor, CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN
KASHMERE GATE, DELHI-11000

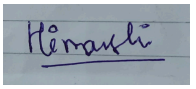
April 2024

UNDERTAKING REGARDING ANTI-PLAGIARISM

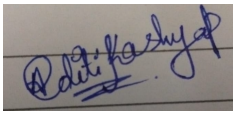
We,Himanshi,Ishika Chhoker, Aditi Kashyap,Bhuvi Singh and Ishrat Khan hereby, declare that the material/ content presented in the project report is free from plagiarism and is properly cited and written in our own words. We are fully aware about the UGC regulations for promotion of academic integrity and prevention of plagiarism in higher educational institutions. In case plagiarism is detected at any stage, we shall be solely responsible for it. A copy of the Plagiarism Report is also enclosed.

Sign Here 

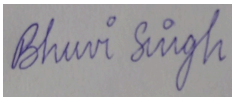
Name: Ishika Chokker
Enrollment No: 07301012022

Sign Here 

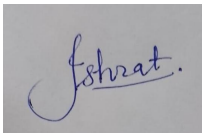
Name: Himanshi
Enrollment No: 06901012022

Sign Here 

Name: Aditi Kashyap
Enrollment No: 01201012022

Sign Here 

Name: Bhuvi Singh
Enrollment No: 05201012022

Sign Here 

Name: Ishrat Khan
Enrollment No: 07601012022

B. Tech, CSE, 4th Sem

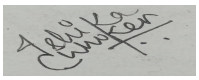
CERTIFICATE

This is to certify that the project entitled **Travelling Salesman Problem is an authentic** record of our own work, under the supervision of **Dr. Vivekanand Jha**, Associate Professor, CSE, Indira Gandhi Delhi Technical University for Women, Delhi. It is submitted in the requirements for the lab project of Design and Analysis of Algorithms (BCS- 204) for the course of the Bachelor of Technology degree in Computer Science and Engineering at Indira Gandhi Delhi Technical University for Women, Delhi, India. This certificate attests to the originality and authenticity of the work presented in this project. We further certify that:

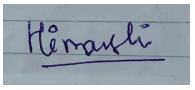
●This work has not been submitted to any other Institution for any sort of degree/diploma/certificate in India or abroad.

●We totally abided by the plagiarism guidelines given to us by the university for writing and preparing this report.

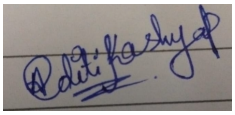
●Whenever we have used materials (text, data, theoretical analysis/equations, codes/program, figures, tables, pictures, text etc.) from other sources, we have done proper citation, linking of the owner's rightful source and giving due credit wherever needed.

Sign Here 

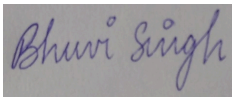
Name: Ishika Chokker
Enrollment No: 07301012022

Sign Here 

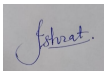
Name: Himanshi
Enrollment No: 06901012022

Sign Here 

Name: Aditi Kashyap
Enrollment No: 01201012022

Sign Here 

Name: Bhuvi Singh
Enrollment No: 05201012022

Sign Here 

Name: Ishrat Khan
Enrollment No: 07601012022

This is to certify that the project has been done under my supervision and guidance. It has not been submitted elsewhere either partial or full, for award of any other degree or diploma to the simplest of my knowledge and belief.

Vivekanand Jha
Associate Professor, CSE

ABSTRACT

Although the accelerated advances in current technology in all regions, still there are few real-world NP complex problems that still elude physicists. Some of such complex problems are Travelling Salesman Problem (TSP), Knapsack Problem, Graph Colouring, Vehicle Routing etc.
The Travelling Salesman Problem (TSP) is a classic combinatorial optimization problem accompanying applications in different fields. This group project proposed to solve the TSP using five various algorithms: Firefly Algorithm, Genetic Algorithm with Crossover Search, Genetic Algorithm with Reinforcement Learning, Ant Colonisation Algorithm, and Cuckoo Algorithm. To attain this, we extensively inspected five research documents that provided insights and methodologies for each algorithm.

For the Firefly Algorithm, we executed the light intensity-based optimization to find the shortest tour. The Genetic Algorithm with Crossover Search utilised genetic operations like crossover and mutation to evolve a population of tours towards an optimal solution. Incorporating Reinforcement Learning into the Genetic Algorithm granted the algorithm to learn from past experiences and adaptively refine the solutions.

The Ant Colonisation Algorithm mocked the foraging act of ants to find promising paths and assemble a possible tour. Lastly, the Cuckoo Algorithm was employed to imitate the brood parasitism behaviour of cuckoos to lay eggs (solutions) in the nests (solutions scope) of other birds (search scope), ensuring variety and exploration.

Through comparative study and test, we evaluated the performance of these algorithms in terms of resolution quality, convergence speed, and computational efficiency. Our results explained that each algorithm exhibited unique strengths and weaknesses in solving the TSP. The Genetic Algorithm with Reinforcement Learning and the Ant Colonisation Algorithm mainly outperformed others in finding high-quality solutions with justifiable computational resources.

In conclusion, this project gave a comprehensive study and realistic implementation of various algorithms for solving the TSP. The findings offer valuable insights into the influence of various optimization techniques and contribute to the ongoing research in combinatorial optimization and algorithm design.

This abstract provides an overview of your project, emphasising the algorithms used, the methods used, and the main verdicts and conclusions. Feel free to regulate or increase any particular details to better reflect your project's scope and results

TABLE OF CONTENTS

10.	Implementation and Results 10.1 Implementation environment 10.2 Individual paper results 10.3 Comparative results	
11.	Conclusion and Future Work References APPENDIX	

CHAPTER 1

1.INTRODUCTION

Overview: This chapter discusses IoT and its potential to create a smarter future after reading research papers [1] and [2]. Following that, chapter introduces IoT enabler technologies, such as Wireless Sensor Networks.

1.1 NP-Completeness

NP-completeness is a concept in computational complexity theory that concerns decision problems—problems with yes-or-no answers. A problem is NP-complete if it belongs to the set of NP (nondeterministic polynomial time) problems and has the property that any problem in NP can be transformed (or reduced) to it in polynomial time. This means that if you could solve one NP-complete problem in polynomial time, you could solve all NP problems in polynomial time, which is currently not known to be possible. NP-complete problems are considered among the most challenging problems in computer science.

1.2 P, NP, NPC, and NPH

- P (Polynomial Time): This class refers to decision problems that can be solved by a deterministic Turing machine (a theoretical computing model) in polynomial time, meaning the time required to solve the problem grows polynomially with the size of the input.
- NP (Nondeterministic Polynomial Time): This class includes decision problems that can be verified in polynomial time given a solution. In other words, if someone claims to have a solution to an NP problem, you can verify its correctness in polynomial time. NP problems are generally harder to solve than problems in P, but their solutions can be quickly verified.
- NPC (NP-complete): A problem is NP-complete if it is in NP and every problem in NP can be reduced to it in polynomial time. NP-complete problems are the hardest problems in NP in terms of computational complexity. If one NP-complete problem can be solved in polynomial time, then all problems in NP can be solved in polynomial time.
- NPH (NP-hard): A problem is NP-hard if every problem in NP can be reduced to it in polynomial time. Unlike NP-complete problems, NP-hard problems do not need to be in NP themselves. NP-hard problems are at least as hard as the hardest problems in NP but may not necessarily be solvable in polynomial time.

These classifications are fundamental in understanding the complexity landscape of computational problems and are essential in fields like computer science and mathematical theory of computation.

CHAPTER -2

PROBLEM STATEMENT

Travelling Salesman Problem (TSP): In computer science and mathematics, the Travelling Salesman Problem is a classic optimization problem. The goal is to find the most efficient route that visits a set of given cities (or points) exactly once and returns to the starting city, minimising the total distance or cost travelled.
The TSP belongs to the class of NP-complete problems, a category of decision problems for which no known polynomial-time algorithm exists. Its classification as NP-complete implies that as the number of cities increases, the time required to find an optimal solution grows exponentially. This characteristic places the TSP at the forefront of combinatorial optimization problems that are both intellectually intriguing and practically significant.

LOGICAL PROOF:

To logically prove that the Travelling Salesman Problem (TSP) is NP-complete, we follow two steps:

1. Show that TSP is in NP.
2. Demonstrate that TSP is NP-hard by reducing a known NP-complete problem toTSP.

Part 1: TSP is in NP
Verification:

- Given a proposed solution (a specific route), calculate its total length.
- Confirm if the total length is less than or equal to a given bound.

Polynomial Time Verification:
· The verification process involves iterating through the edges of the route, making it polynomial in the size of the input.

Part 2: TSP is NP-hard
Reduction of Hamiltonian Circuit (HAM-CYCLE) to TSP:

1.Hamiltonian Circuit (HAM-CYCLE):Given a graph, determine if there exists a simple circuit that visits every vertex exactly once.This is our Hamiltonian Circuit (HAM-CYCLE) .

2.Graph Transformation:

· For each vertex in the Hamiltonian Circuit graph, create a corresponding city in the TSP graph

·Connect each pair of cities with edges and assign weights:

If there's an edge in the Hamiltonian Circuit graph, set the weight to 1. If there's no edge, set the weight to infinity.

Claim:

·If there is a Hamiltonian Circuit in the original graph, the corresponding TSP instance has a tour with total distance less than or equal to the given bound.

·If there is no Hamiltonian Circuit in the original graph, every TSP tour in the corresponding instance has a total distance greater than the given bound.

Conclusion:

·Solving TSP in polynomial time would imply a polynomial-time solution for the Hamiltonian Circuit.

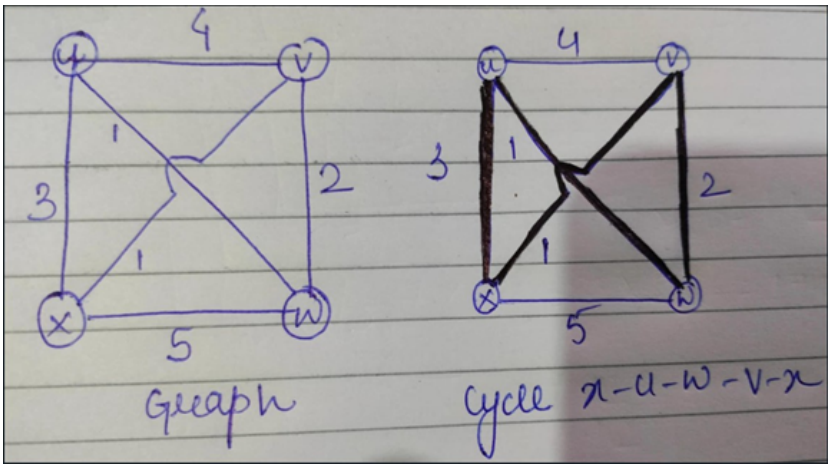
· Since the Hamiltonian Circuit is NP-complete, and TSP is NP-hard, TSP is NP-complete.

MATHEMATICAL PROOF

Prove - TSP is a NP Complete Problem

A graph of a set of cities is given . A salesman starts from a city. He must visit the city exactly once and should finish at the city he starts from (returns to the starting point). The path he follows should be the shortest one or should have a cost value which is less than K . Thus TSP is a special case of hamiltonian cycle problem where the path cost should be minimum.

Consider the following graph containing cities u, v, w ,x



Here the travelling salesman can follow the path x-u-v-x which forms the shortest path with cost 7km.

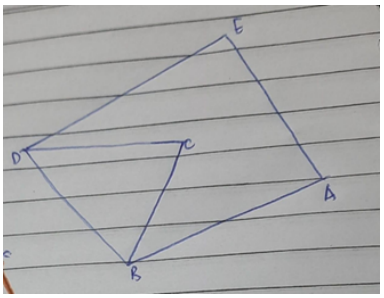
STEP 1 -: Prove that TSP is in NP

A set with n elements has 2^n possible subsets. Then a graph with n vertices has 2^n possible permutations of vertices. So an algorithm needs to check whether any one of these permutation form a hamiltonian path with cost value which is less than K.If the graph is represent as an adjacency matrix then it will n!.And it has worst time complexity $\Theta(2n$. Let us take a graph and say that sequence of vertices from a hamiltonian cycle with cost value less than K. An algorithm can verify . so TSP is in the class NP .

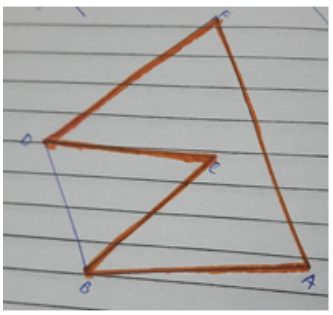
STEP - 2 -: Prove TSP is a NP Hard problem

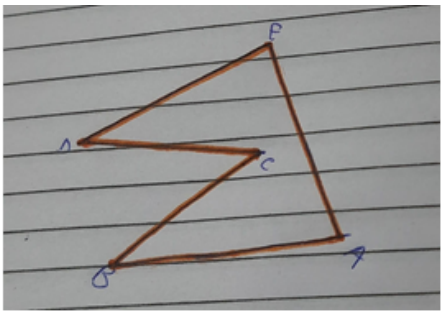
To prove TSP a NP Hard problem we have to take a known NP Hard problem and reduce that NP Hard problem to TSP.

Taking Hamiltonian Cycle Problem as known NP Hard and reducing it to the TSP. Consider a graph - G(V , E)

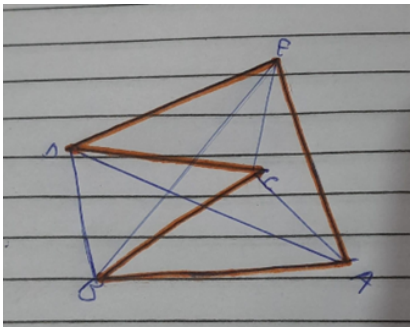


Taking an instance of the Hamiltonian cycle.

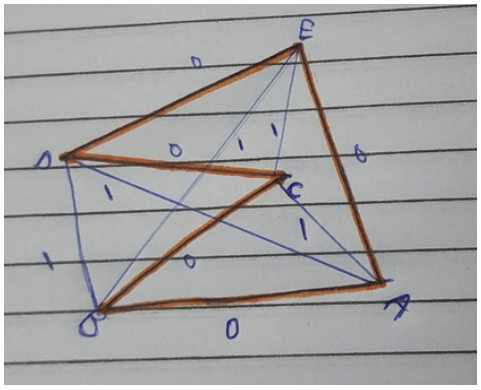




Now form a complete graph from the above instance of Hamiltonian cycle. Graph G' = (V , E')



Next we have assigned a cost value to every edge of this graph. If the edge is present in the hamiltonian cycle , then assign cost value 0 to it. If it is not, then assign cost value 1 to it.



We have shown that Graph G has a hamiltonian cycle if and only if Graph G' has a tour of cost almost '0' .
From this , an instance of TSP can be produced from the above graph by traversing through the edges which have a cost value of 0.

In step 1, we proved that TSP is in NP.

In step 2, we proved that TSP is NP-hard.

So, TSP is an NP-Complete problem.

CHAPTER - 3

LITERATURE REVIEW

PAPER 1
Genetic algo using crossover

This paper, "Solving TSP Problem By Using Genetic Algorithm" by Fozia Hanif Khan et al., explores a new approach to applying genetic algorithms (GAs) for the Travelling Salesman Problem (TSP).

Motivation and Background:

- TSP is a classic combinatorial optimization problem seeking the shortest route visiting all cities exactly once and returning to the origin. It is NP-hard, meaning finding the optimal solution becomes computationally expensive for large datasets.
- GAs mimic natural selection, where a population of individuals evolves towards better solutions. They are well-suited for problems like TSP where an exact solution might be intractable but finding a "good enough" solution is valuable.

Traditional GA Techniques for TSP:

- **Chromosome Representation:** Typically, a permutation encoding is used, where each element in a sequence represents a city to be visited in the order of the sequence. Other representations like edge-based encoding also exist.
- **Fitness Function:** This function evaluates the quality of a solution (chromosome). In TSP, it often measures the total distance travelled in the proposed route. Selection pressure is applied, favouring chromosomes with lower total distance (fitter solutions) for reproduction.
- **Genetic Operators:**
    - **Crossover:** Combines genetic material from two parent chromosomes to create offspring, potentially inheriting good qualities from both. Common

crossover techniques include single-point crossover, two-point crossover, and ordered crossover.

- **Mutation:** Introduces random changes to chromosomes, maintaining diversity in the population and preventing premature convergence. Common mutation techniques include bit-flip mutation (for binary representations) and swap mutation (for permutation encoding).

Proposed Method and Improvements:

- The paper suggests a new chromosome representation using a binary matrix. This is an alternative to the traditional permutation encoding, potentially offering advantages depending on the specific problem.
- It introduces a novel "fittest criteria" for selecting parents during crossover. This criteria can significantly impact the quality and convergence speed of the GA. The exact details of this criteria are not provided in the summary but would be interesting to explore in the full paper.
- The method is applicable to both symmetric TSP (distances between cities are the same in both directions) and asymmetric TSP (distances can differ). The paper also proposes a unique way to represent asymmetric problems within the GA framework, which is an important contribution as many real-world scenarios involve asymmetric distances.

Comparison with Existing Work:

The paper acknowledges other approximation techniques for TSP, including simulated annealing, ant colony optimization, and GAs from previous research. It positions its contribution as building upon existing GA applications for TSP but introducing improvements for both symmetric and asymmetric problem formulations.

Open Questions and Future Research Directions:

- The paper mentions a new "fittest criteria" but doesn't elaborate on its details. Understanding this criteria is crucial to evaluate the effectiveness of the proposed method.
- An empirical comparison with other GA approaches and existing approximation techniques for TSP would strengthen the paper's contribution.
- How do the choices of population size, crossover rate, and mutation rate affect the performance of the proposed method?

Overall, this paper presents a promising approach to solving TSP with GAs. The proposed binary matrix representation, novel fittest criteria, and applicability to asymmetric problems offer interesting directions for further research. A more detailed explanation of the fittest criteria and an empirical evaluation would be valuable additions to solidify the paper's contribution.

PAPER 2
Firefly algorithm

Firefly Algorithm (FA), a nature-inspired metaheuristic algorithm, for solving the Travelling Salesman Problem (TSP). The authors provide an extensive exploration of TSP, the Firefly Algorithm's foundational concepts, key parameters, and its transformation to address TSP. The study evaluates the performance of the adapted FA against well-known metaheuristic algorithms, namely Ant Colony Optimization (ACO), Genetic Algorithm (GA), and Simulated Annealing (SA), using standard TSP instances.

Strengths:

1. **Innovative Adaptation:** The paper introduces a unique adaptation of FA, originally designed for continuous optimization problems, to tackle the combinatorial optimization challenge posed by TSP. This adaptation showcases the flexibility and potential of FA in handling diverse optimization problems.
2. **Comprehensive Explanation:** The authors provide a thorough explanation of the Firefly Algorithm's principles, including its basic concepts and key parameters. This ensures readers with varying familiarity levels can understand the algorithm and its application to TSP.
3. **Comparative Analysis:** The study presents a detailed comparative analysis of the adapted FA against other metaheuristic algorithms like ACO, GA, and SA. This comparison provides valuable insights into the relative strengths and weaknesses of each algorithm when applied to TSP.
4. **Clear Implementation Details:** The paper offers clear pseudo-code for the basic FA algorithm and outlines the necessary modifications for TSP adaptation. This clarity aids in understanding and potential replication of the study.
5. **Results Presentation:** Both numerical and graphical representations of results are included, enhancing the clarity of the findings and facilitating easier comparison between algorithms.

Areas for Improvement:

1. **Literature Review Depth:** While the paper mentions other nature-inspired algorithms like GA, MA, ACO, and SA, a deeper literature review that compares FA's performance with these algorithms specifically for TSP would add more context and depth to the study.
2. **Termination Criteria Elaboration:** The paper mentions using result stagnation as a termination criterion but could benefit from elaborating on how this criterion is defined and its potential impact on algorithm performance.
3. **Experimental Setup Details:** Additional details on the experimental setup, such as parameter tuning and initialization methods, would offer better insights into the algorithm's robustness and behaviour.

4. **Discussion on Limitations:** A section discussing the limitations encountered while adapting FA for TSP would provide a more balanced view and suggest areas for potential future research.

Conclusion:

The paper by Sharad N. Kumbharana and Prof. Gopal M. Pandey presents a novel adaptation of the Firefly Algorithm for solving the Travelling Salesman Problem. The study's thorough exploration of FA's principles, its application to TSP, and the comparative analysis against other metaheuristic algorithms make it a valuable contribution to the optimization field. However, addressing the suggested areas for improvement could further enhance the paper's quality, depth, and impact. Overall, the paper showcases the potential of nature-inspired algorithms like FA in addressing complex combinatorial optimization problems like TSP, paving the way for future research and applications in this domain.

PAPER 3

Genetic Algorithm and Reinforced Learning

The Travelling Salesman Problem (TSP) is a well-known and challenging problem in the field of computer and mathematics education. The problem is described as simple, yet mathematically complex, and falls under the category of NP-hard problems, making it difficult to solve (Biswas, Mitra, & Sengupta, 2020). In recent years, researchers have explored various methods to tackle this problem, including Genetic Algorithm (GA) and Reinforcement Learning (RL).

One study by Biswas, Mitra, and Sengupta (2020) focused on utilising GA and RL to solve the TSP. The authors demonstrated the application of GA in solving the TSP, highlighting the use of Crossover and Mutation operators, as well as the sorting of solutions to calculate the best options. This study provides valuable insights into the practical implementation of GA and RL for addressing the TSP.

In addition to the study by Biswas, Mitra, and Sengupta (2020), previous research has also delved into the use of GA for solving optimization problems. Islam et al. (2012) proposed an Adaptive Differential Evolution Algorithm with novel mutation and crossover strategies for global numerical optimization. While this study does not directly focus on TSP, it provides valuable insights into the potential crossover and mutation strategies that can be applied to optimization problems, including TSP.

Furthermore, Kora and Yadlapalli (2017) conducted a review of different crossover operators in genetic algorithms. Although their study did not specifically address TSP, the insights provided are relevant for understanding the different approaches that can be employed to improve the efficiency and effectiveness of genetic algorithms in solving complex optimization problems.

Despite the existing research on GA and RL for addressing the TSP, there are still knowledge gaps that warrant further investigation. One potential future research direction is to explore the combination of RL with other metaheuristic algorithms, such as simulated annealing or particle swarm optimization, to enhance the performance of TSP solutions. Additionally, there is a need for comparative studies that evaluate the effectiveness of different crossover and mutation operators in GA for solving TSP instances of varying complexities.

In conclusion, the literature reviewed showcases the significance of GA and RL in addressing the TSP, as well as the potential for further advancements in this area. By integrating and synthesising the provided research findings, this literature review offers a comprehensive understanding of the current state of research on TSP using GA and RL, while also highlighting potential future research directions.

PAPER 4

Ant Colony Algorithm

The Ant Colony Optimization (ACO) algorithm is a metaheuristic inspired by the foraging behaviour of ants to solve optimization problems. One of its popular applications is in solving the Travelling Salesman Problem (TSP), where the goal is to find the shortest possible route that visits each city exactly once and returns to the original city It is designed to simulate the ability of ant colonies to determine shortest paths to food. Although individual ants posses few capabilities,their operation as a colony is capable of complex behavior.Real ants can indirectly communicate by pheromone

information without using visual cues and are capable of finding the shortest path between food sources and their nests.The ant deposits pheromone on the trail while walking, and the other ants follow the pheromone trails with some probability which are proportioned to the density of the pheromone. The more ants walk on a trail, the morepheromone is deposited on it and more and more ants follow the trail. Through this mechanism, ants will eventually find the shortest path. Artificial ants imitate the behavior of real ants how they forage the food, but can solve much more complicated problems than real ants can. A search algorithm with such concept is called Ant Colony Optimization.

Strengths for TSP:

1)Finding Optimal Routes: ACO excels at finding good, close-to-optimal routes in TSP. It efficiently explores the search space due to the probabilistic selection process and pheromone-based communication between "ants."

2)Flexibility: ACO can handle variations of TSP, such as asymmetric TSP (distances differ in two directions between cities) or TSP with additional constraints (time windows for visiting cities).

Result and Advantages:

1)Good Solutions: Over time, the positive feedback loop from pheromone updates steers the search towards shorter paths.

2)Adaptability: ACO can adapt to changing distances or new cities added to the TSP instance.

Effective Exploration:

1)Probabilistic Selection: Unlike greedy algorithms that always pick the closest unvisited city, ACO's probabilistic approach allows exploring a wider range of paths. This is because even less optimal connections have a chance of being chosen, leading to the discovery of potentially shorter routes.

2)Pheromone Evaporation: The gradual decay of pheromone over time prevents the algorithm from getting stuck in local optima (good but not necessarily the best solutions). This ensures ACO keeps exploring new possibilities and avoids converging on suboptimal solutions prematurely.

Conclusion

1)Effective Exploration: ACO utilises probabilistic selection and pheromone evaporation to explore a diverse range of paths. This helps avoid getting stuck in local optima and allows for the discovery of potentially shorter routes.

2)Convergence to Good Solutions: The positive feedback loop involving pheromone deposition on shorter paths steers the search towards the optimal solution. As more ants find shorter routes, those paths become more attractive, leading to convergence.
pen_spark

PAPER 5

Cuckoo Search Algorithm

Introduction

The travelling salesman problem (TSP) is a well-known NP-hard combinatorial optimization problem that has been studied extensively in the field of operations research and computer science. Researchers have applied various metaheuristic algorithms to solve the TSP, one of which is the cuckoo search algorithm. This algorithm is inspired by the brood parasitism of some cuckoo species.

Thematic Sections

Discrete Cuckoo Search Algorithm

In their study, Ouaarab, Ahiod, and Yang (2014) presented an improved and discrete version of the cuckoo search (CS) algorithm to solve the TSP (Neural Computing and Applications, Springer). They applied a Levy flight strategy and a combination of local search methods to enhance the performance of the algorithm. This innovation represents a key theme in the literature, showing the potential of hybrid approaches that combine global and local search strategies.

Novel Approaches to Cuckoo Search

Ouyang, Zhou, Luo, and others (2013) proposed a discrete cuckoo search algorithm for solving the Euclidean TSP (Applied mathematics & information sciences, naturalspublishing.com). Their novel approach included adaptations to the algorithm specific to the spherical nature of the problem, showing the flexibility and adaptability of cuckoo search for various representations of TSP.

Random-Key Cuckoo Search

Another variation was explored by Ouaarab, Ahiod, and Yang (2015) with the random-key cuckoo search for the TSP (Soft Computing, Springer). Their method used random keys and a simple local search procedure, highlighting the effectiveness of randomization techniques in solution encoding and the importance of simplicity in design for practical applications.

Discrete Cuckoo Search with Cluster Analysis

Zhang and Yang (2022) developed a discrete cuckoo search algorithm based on random walk and cluster analysis for the TSP (Computers & Industrial Engineering, Elsevier). This approach focused on improving the algorithm's ability to explore the search space effectively using clustering methods, demonstrating a trend towards integrating data analysis techniques with metaheuristics.

Conclusion and Further Research

The literature highlights that the cuckoo search algorithm is a promising metaheuristic for solving TSP problems, with various studies demonstrating its adaptability and potential for hybridization with other optimization techniques. However, there are gaps in research

regarding the scalability of these algorithms for larger instances of TSP and the comparison of their performance against other state-of-the-art algorithms.

Future research could focus on evaluating the performance of cuckoo search algorithms on large-scale TSP instances and developing more sophisticated hybrid models that combine cuckoo search with other powerful optimization methods. There is also a need for more systematic and comparative studies to establish a clearer understanding of the strengths and limitations of cuckoo search in the context of solving TSP problems

CHAPTER 4

IMPLEMENTATION AND RESULTS

4.1 Section

GENETIC ALGO USING CROSSOVER OPERATION

Here's the algorithmic representation of the provided Genetic Algorithm code for solving the optimization problem:

GENETIC ALGORITHMS: The Genetic Algorithm involves the following basic steps.

- Evaluation
- Cross over
- Mutation

Here we are trying to describe the main function used by an algorithm to find the shortest closed path through a group of two dimensional positions. The Genetic Algorithm transforms a population of individual objects, each with an associated fitness value, into a new generation of the population using the Darwin principle of individual reproduction and survival of the fittest and naturally occurring genetic operation such as a crossover (recombination) and mutation. Each individual in the population represents a possible solution to a given problem. The genetic algorithm attempts to find a very good or best solution to the problem by genetically breeding the population of individuals.

EVOLUTION: The evolution function plays a very important role in genetic algorithms. Here we use a fittest function to decide how good a chromosome is. It also helps us in the selection of better chromosomes for the next cross over step, if we have a relatively large number of initial population.

CROSS OVER: After the completion of the selection process, the chromosomes chosen to be parents for the next generation are recombined to form children that are new chromosomes. This combination can take many forms and the crossover operator can greatly affect the result of the search.

MUTATION: The operation of mutation allows new individuals to be created. It begins by selecting an individual from the population based on its fitness. A point along the string is selected at random and the character at that point is randomly changed, the alternate individual is then copied into the next generation of the population.

Mutation is performed after crossover by randomly choosing a chromosome in the new generation to mutate. We randomly choose a point to mutate and switch that

point. Many types of mutation operators exist. Here we are using the bit flip method which is only used with a binary chromosome representation, that changes a particular point on the chromosome to its opposite.

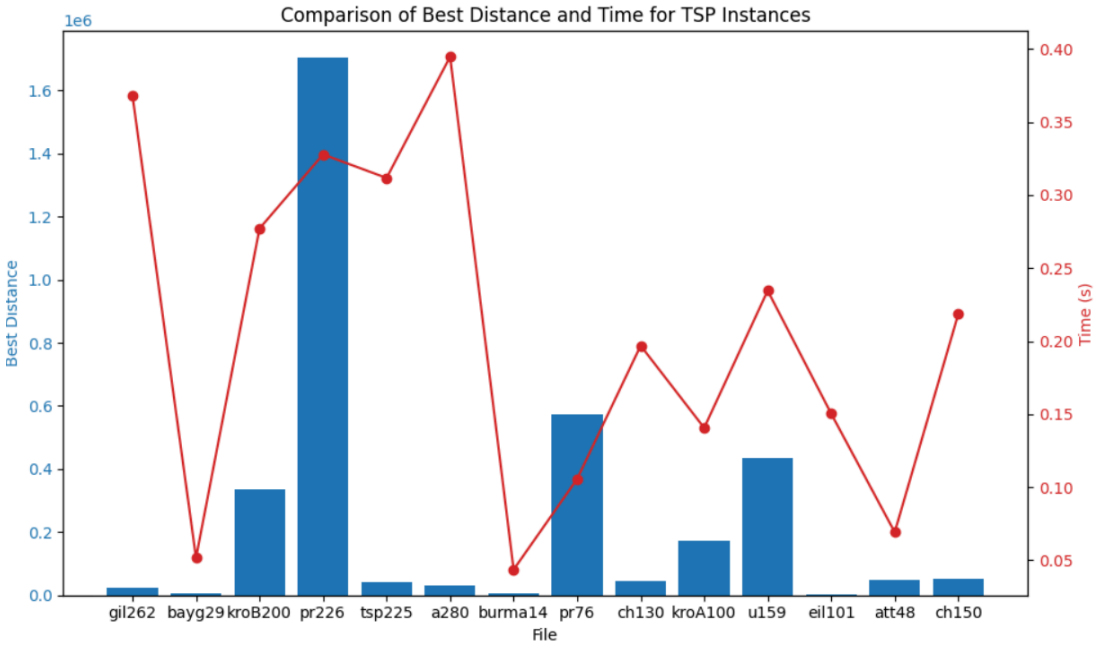
This algorithm outlines the steps involved in the genetic algorithm for solving the optimization problem, including initialization, selection, crossover, mutation, and termination. It provides a framework for solving optimization problems using genetic algorithms.

```
Best individual: [1 0 1 1 1 1 0 0 1 1]
Best fitness: 7
```

THIS IS THE OUTPUT OF THE CODE OF THE BASIC IMPLEMENTATION OF THE ALGO FUNCTIONING FOR GENETIC ALGORITHM (USED FOR SOLVING TSP)

```
/content/drive/MyDrive/tsp_problem/gil262.tsp
/content/drive/MyDrive/tsp_problem/bayg29.tsp
/content/drive/MyDrive/tsp_problem/kroB200.tsp
/content/drive/MyDrive/tsp_problem/pr226.tsp
/content/drive/MyDrive/tsp_problem/tsp225.tsp
/content/drive/MyDrive/tsp_problem/a280.tsp
/content/drive/MyDrive/tsp_problem/burma14.tsp
/content/drive/MyDrive/tsp_problem/pr76.tsp
/content/drive/MyDrive/tsp_problem/ch130.tsp
/content/drive/MyDrive/tsp_problem/kroA100.tsp
/content/drive/MyDrive/tsp_problem/u159.tsp
/content/drive/MyDrive/tsp_problem/eil101.tsp
/content/drive/MyDrive/tsp_problem/att48.tsp
/content/drive/MyDrive/tsp_problem/ch150.tsp
       File  Best Distance  Time (s)
0    gil262          25476  0.367915
1    bayg29           4549  0.051724
2   kroB200         335147  0.277117
3     pr226        1702839  0.327812
4    tsp225          41380  0.311697
5      a280          32631  0.394684
6   burma14           4780  0.043600
7      pr76         574337  0.105632
8     ch130          44492  0.196900
9   kroA100         174436  0.140638
10     u159         435038  0.234559
11   eil101           3614  0.150328
12    att48          49651  0.069211
13    ch150          52827  0.218427
```

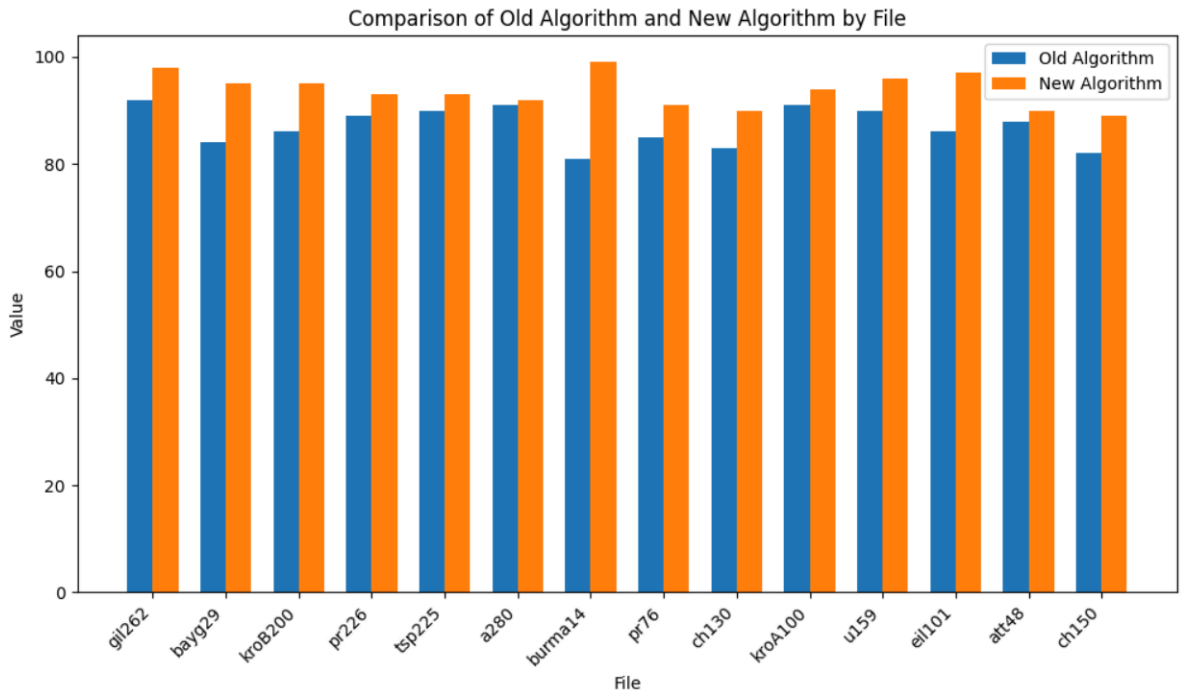
THIS IS THE OUTPUT OF THE CODE BY TAKING THE FINAL INSTANCES AND USING THE ALGO BY CALLING IT IN ONE MORE CODE, HENCE THE FINAL IMPLEMENTATION OF THE ALGORITHM



THIS IS THE FINAL GRAPH OF THE ABOVE IMPLEMENTED INSTANCES USING THE GENETIC ALGORITHM

File	OldAlgo	NewAlgo	OptimalSol
gil262	92	98	25995
bayg29	84	95	4594
kroB200	86	95	352786
pr226	89	93	1702839
tsp225	90	93	47022
a280	91	92	35468
burma14	81	99	4828
pr76	85	91	631139
ch130	83	90	52343
kroA100	91	94	183616
u159	90	96	453164
eil101	86	97	3725
att48	88	90	55167
ch150	82	89	62889

THIS IS THE TABLE IN WHICH WE TOOK DATA FROM OUR RESEARCH PAPER AND THEN WE CALCULATED OUR OWN DATA OF THE INSTANCES AND THEN DISPLAYED IT IN THE FORM OF A TABLE ABOVE



THIS IS THE FINAL WAY OF MAKING THE GRAPH AS REPRESENTED IN THE RESEARCH PAPER

4.2 Section

SOLVING TRAVELLING SALESMAN PROBLEM USING FIREFLY ALGORITHM

This research paper proposes an adaptation of the Firefly Algorithm specifically tailored to address the Travelling Salesman Problem. The implementation details and experimental results are presented as follows:

Basic Firefly algorithm

```
Objective function f(x),   x = (x1... xd)T
Generate initial population of fireflies xi (i = 1, 2... n)
Light intensity Ii at xi is determined by f(xi)
Define light absorption coefficient γ
while (t < MaxGeneration)
  for  i = 1 : n all n fireflies
      for  j = 1 : n all n fireflies      (inner loop)
          if  (Ii  <  Ij),
              Move firefly i towards j;
          end if
          Vary attractiveness with distance r via exp[−γ r]
          Evaluate new solutions and update light intensity
      end for j
  end for i
  Rank the fireflies and find the current global best g*
end while
Post process results and visualization
```

Adapting Firefly Algorithm For TSP

Input:

- **n**: Number of fireflies
- **MaxGeneration**: Maximum number of generations
- **beta0**, **gamma**: Attractiveness parameters

Initialization:

1. Read TSP instance from TSPLIB file and extract city coordinates.
2. Compute the distance matrix between cities.
3. Initialize **n** fireflies with random tours.
4. Compute initial distances (**Ii**) for all fireflies.

5. Main Loop:
1. For each generation **t** from 1 to **MaxGeneration**:
    1. For each firefly **i** from 1 to **n**:
        1. For each firefly **j** from 1 to **n**:
            - If **Ii(i) < Ii(j)**:
                1. Compute distance **r** between fireflies **i** and **j**.
                2. Compute attractiveness **beta** using **beta0** and **gamma**.
                3. Perform 2-opt local search to generate a new tour.
                4. Compute distance of the new tour (**new_distance**).
                5. If **new_distance < Ii(i)**:
                    - Update firefly **i**'s tour with the new tour.
                    - Update **Ii(i)** with **new_distance**.
    2. Rank fireflies based on their distances (**Ii**).
    3. Update global best tour **g_star**.
    4. If **t mod 50 == 0**:
        - Print best tour length of generation **t**.

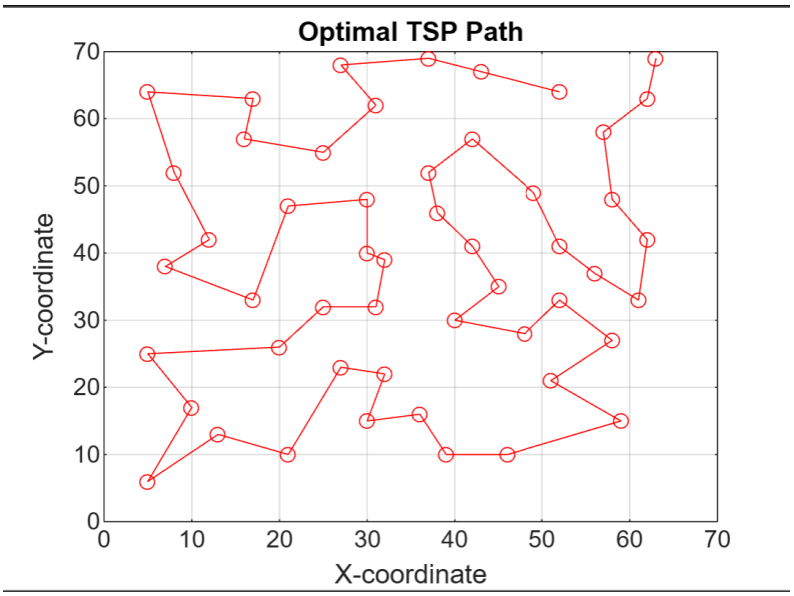
Output:

- Print the number of cities and the best tour length (**sorted_Ii(1)**).
- Plot the optimal path (best tour) on a 2D plot using city coordinates.

OUTPUT The performance of the adapted Firefly Algorithm was evaluated by conducting experiments on various TSP instances.

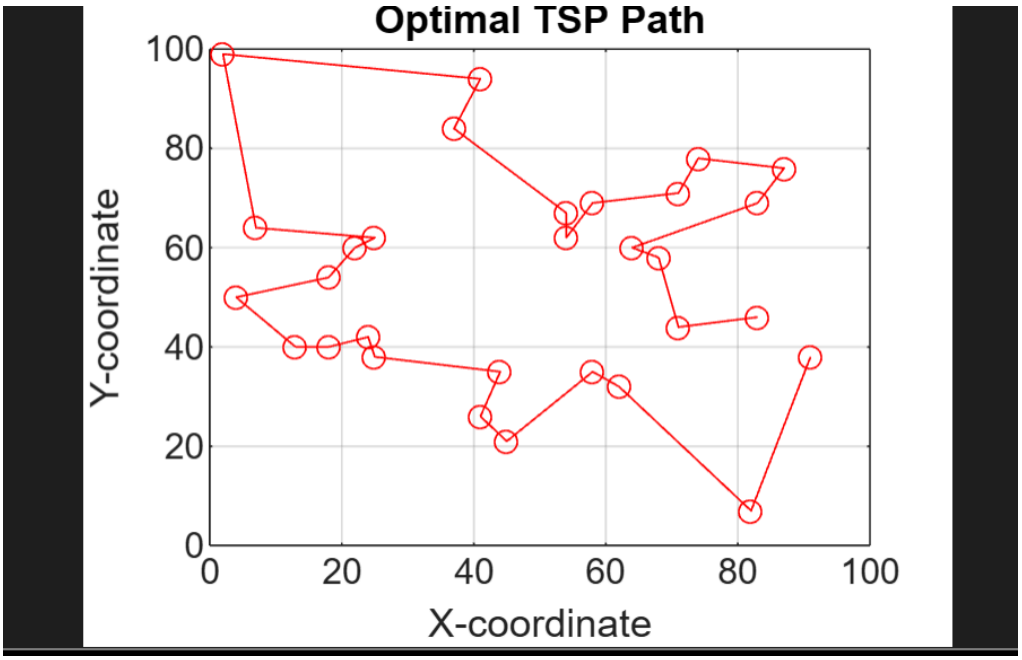
Using eil51 tsp standard instances

```
Generation 50, Best tour length: 478.05
Generation 100, Best tour length: 445.60
Generation 150, Best tour length: 445.60
Generation 200, Best tour length: 438.52
Generation 250, Best tour length: 435.14
Generation 300, Best tour length: 435.14
Generation 350, Best tour length: 435.14
Generation 400, Best tour length: 435.14
Generation 450, Best tour length: 435.14
Generation 500, Best tour length: 435.14
No. of Cities: 51
Best tour length: 435.14
```



Using oliver30 tsp instances

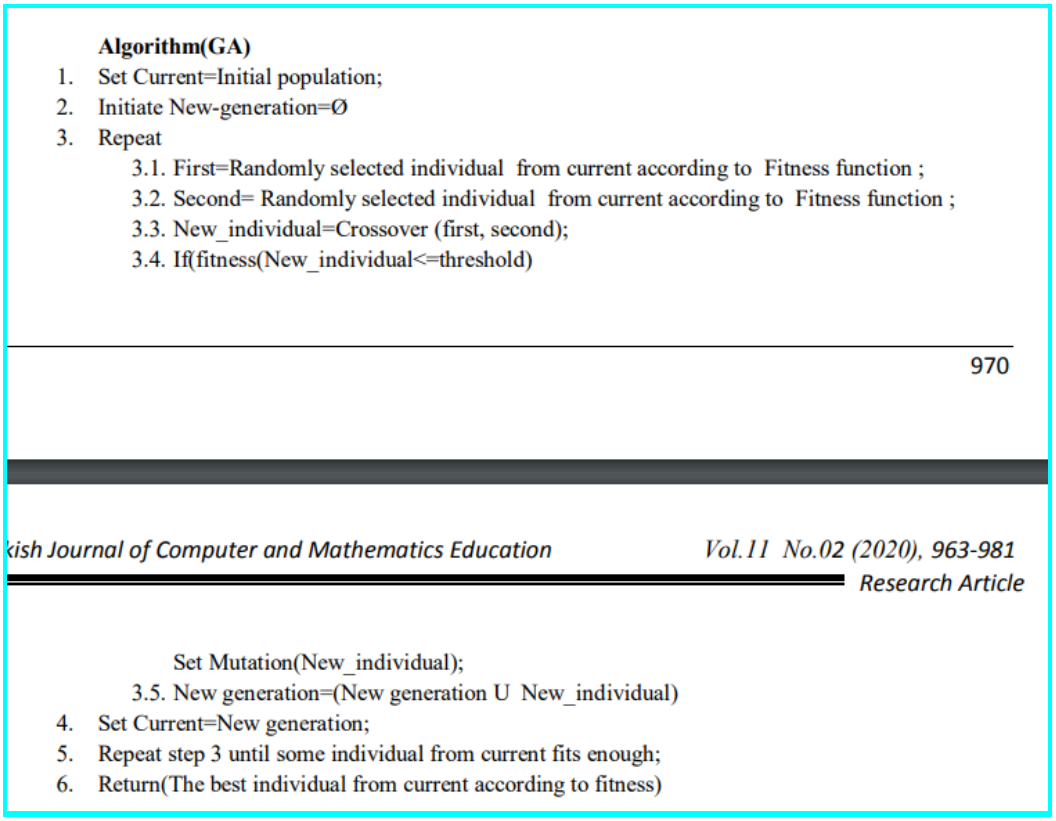
```
Generation 50, Best tour length: 427.14
Generation 100, Best tour length: 424.69
Generation 150, Best tour length: 423.74
Generation 200, Best tour length: 423.74
Generation 250, Best tour length: 423.74
Generation 300, Best tour length: 423.74
Generation 350, Best tour length: 423.74
Generation 400, Best tour length: 423.74
Generation 450, Best tour length: 423.74
Generation 500, Best tour length: 423.74
No. of Cities: 30
Best tour length: 423.74
```



This algorithm provides the best tour length for given TSPLIB instances.
For eil51
Best tour length is 435.14
For oliver30
Best tour length is 423.74.

4.3 Section

GENETIC ALGO AND REINFORCED LEARNING



Algorithm(GA)

1. Set Current=Initial population;
2. Initiate New-generation=Ø
3. Repeat
   - 3.1. First=Randomly selected individual from current according to Fitness function ;
   - 3.2. Second= Randomly selected individual from current according to Fitness function ;
   - 3.3. New_individual=Crossover (first, second);
   - 3.4. If(fitness(New_individual<=threshold)

970

kish Journal of Computer and Mathematics Education *Vol.11 No.02 (2020), 963-981*
Research Article

   - Set Mutation(New_individual);
   - 3.5. New generation=(New generation U New_individual)
4. Set Current=New generation;
5. Repeat step 3 until some individual from current fits enough;
6. Return(The best individual from current according to fitness)

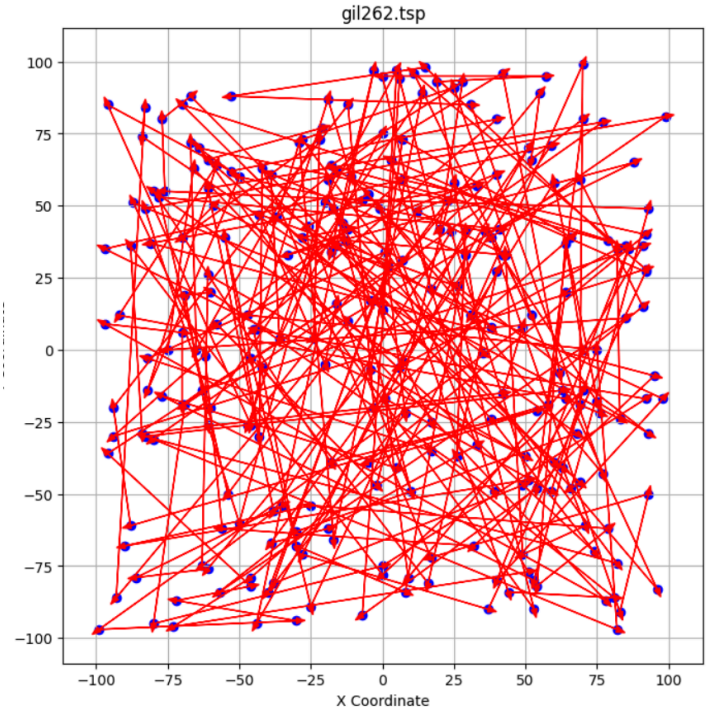
This is the algorithm used for solving TSP .

THIS IS THE OUTPUT OF THE CODE BY TAKING THE FINAL INSTANCES AND USING THE ALGO BY CALLING IT IN ONE MORE CODE, HENCE THE FINAL IMPLEMENTATION OF THE ALGORITHM

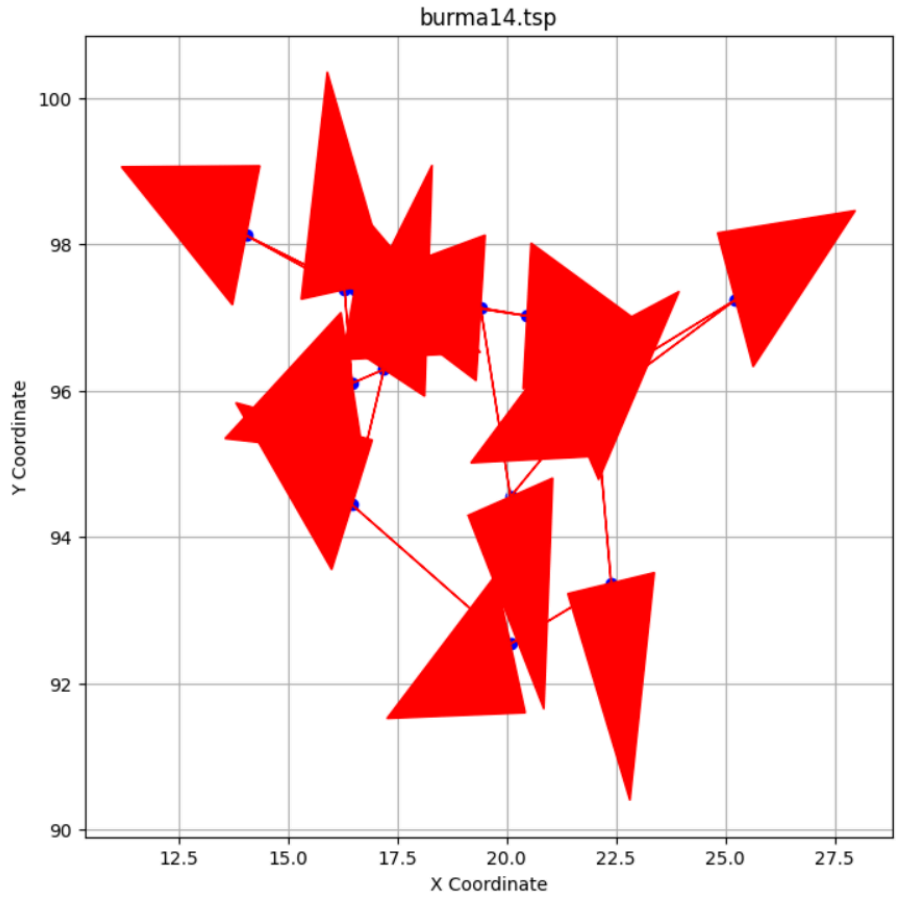
```
/content/drive/MyDrive/testcasefinal/gil262.tsp
/content/drive/MyDrive/testcasefinal/burma14.tsp
/content/drive/MyDrive/testcasefinal/ch130.tsp
/content/drive/MyDrive/testcasefinal/ch150.tsp
/content/drive/MyDrive/testcasefinal/eil101.tsp
/content/drive/MyDrive/testcasefinal/kroB200.tsp
/content/drive/MyDrive/testcasefinal/att48.tsp
/content/drive/MyDrive/testcasefinal/kroA100.tsp
          File                                          Best Path  \
0   gil262.tsp  [102, 9, 68, 160, 207, 63, 180, 213, 54, 78, 1...
1  burma14.tsp     [8, 0, 7, 1, 2, 13, 3, 4, 5, 11, 6, 12, 10, 9]
2    ch130.tsp  [21, 98, 11, 93, 46, 119, 19, 111, 105, 7, 20,...
3    ch150.tsp  [138, 132, 21, 88, 34, 10, 38, 51, 58, 120, 14...
4   eil101.tsp  [53, 29, 49, 45, 48, 2, 96, 57, 82, 95, 62, 46...
5  kroB200.tsp  [121, 84, 71, 66, 138, 73, 175, 160, 31, 164, ...
6    att48.tsp  [47, 26, 36, 6, 39, 32, 14, 10, 34, 41, 1, 13,...
7  kroA100.tsp  [0, 29, 54, 33, 38, 94, 32, 24, 62, 75, 81, 76...

   Best Fitness
0      0.000043
1      0.030948
2      0.000026
3      0.000022
4      0.000358
5      0.000004
6      0.000009
7      0.000008
```

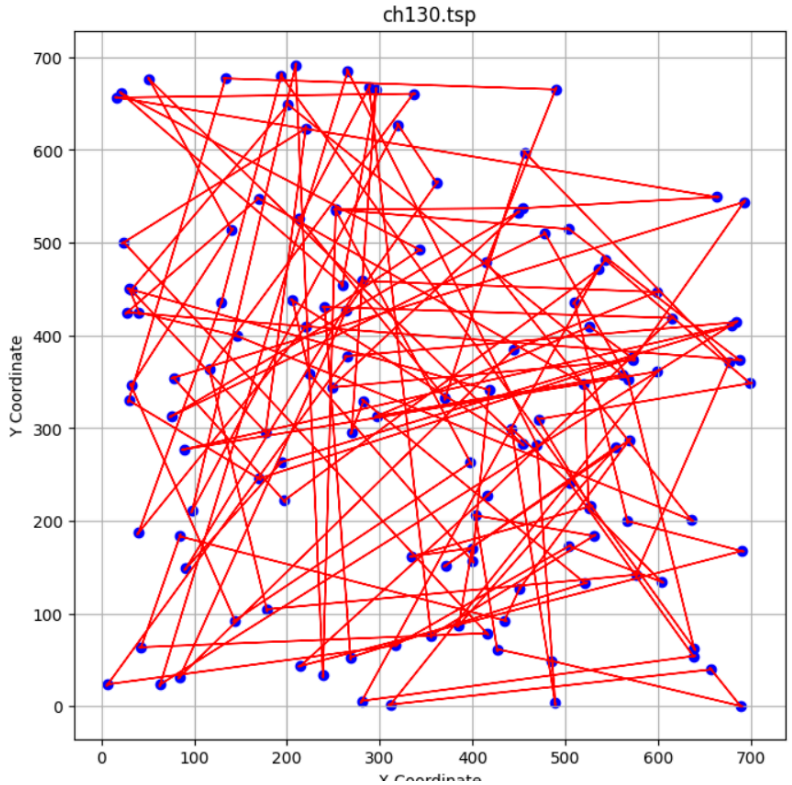
THe FOLLOWING ARE THE FINAL GRAPHS OF THE ABOVE IMPLEMENTED INSTANCES USING THE GENETIC ALGORITHM :



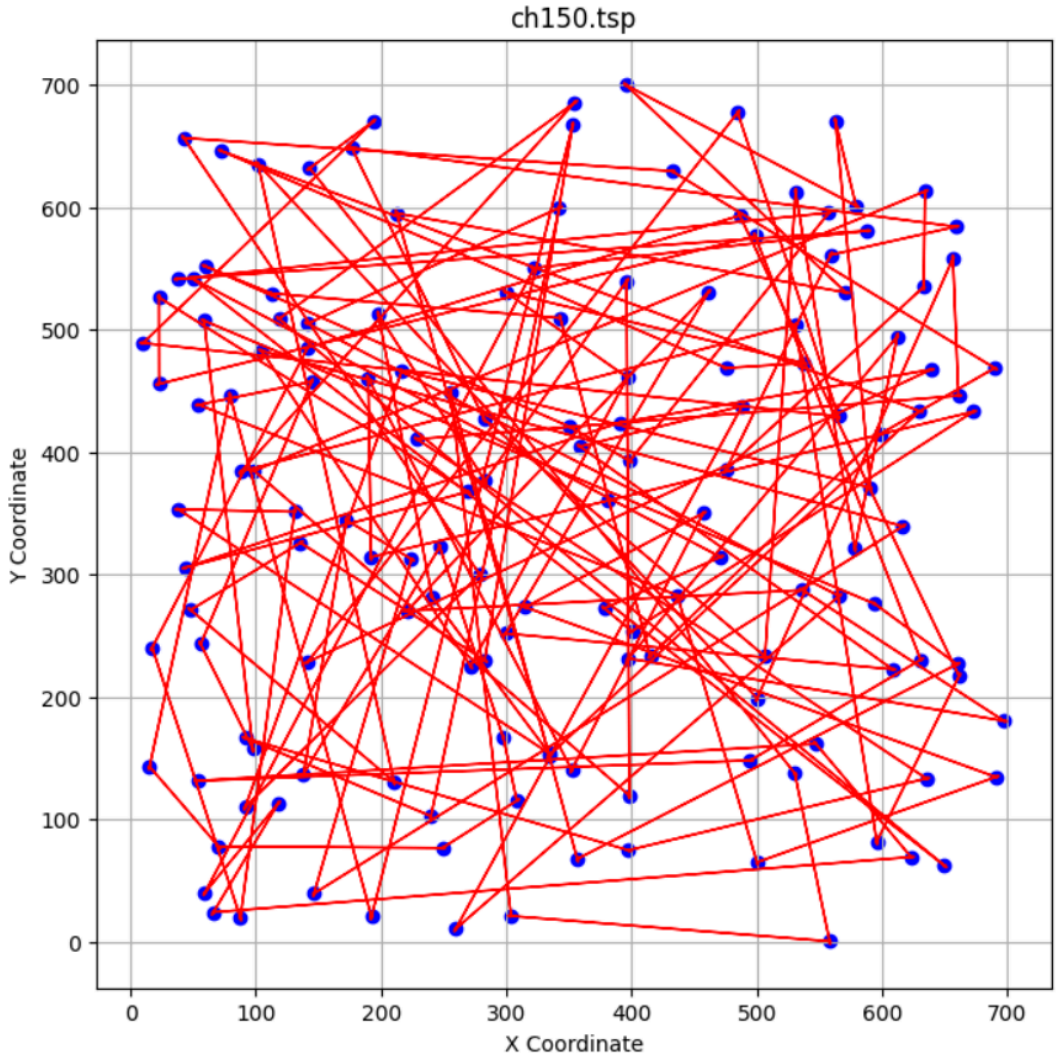
```
/content/drive/MyDrive/testcasefinal/burma14.tsp
Best Path: [ 6 12 13 11  4  5  3  2  1  7  0  8  9 10]
Best Fitness: 0.028414146754767448
```



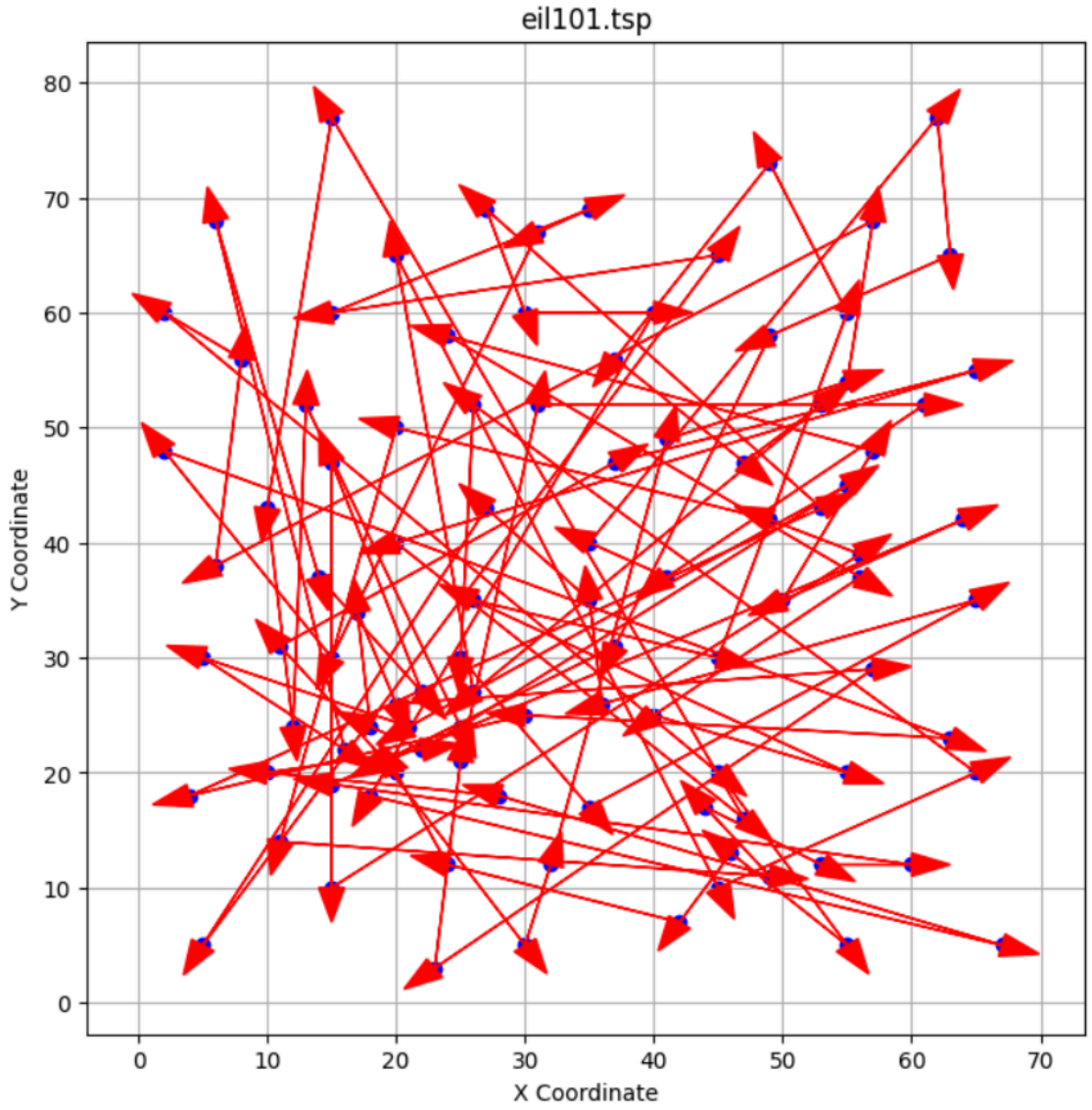
```
/content/drive/MyDrive/testcasefinal/ch130.tsp
Best Path: [ 75  82  30  33  73  94 122  37  14   6  93  56  89  21  52  67  57  26
  46 101  97 126  50  10 124  17 117  27  45  51  68   9  99  85  44   2
  58 107  92 102  31  66 119  11  48   5 110  98 109 128  29  95  71  53
  80  74  15 113 129  61 114  65  84   4  22  62  13 103  25  78  54   3
  63 115  91  72  69   8 108  86   1 123  24  76 100  79  28  18  38  34
  83 112  88 120 105  90  81  47 106  39  43  16  36  55  35  42  64 118
 104  87  96 111 116  32  40  20 127  70   0   7  19  12  59  41 125  77
  60  49 121  23]
Best Fitness: 2.557056810349524e-05
```



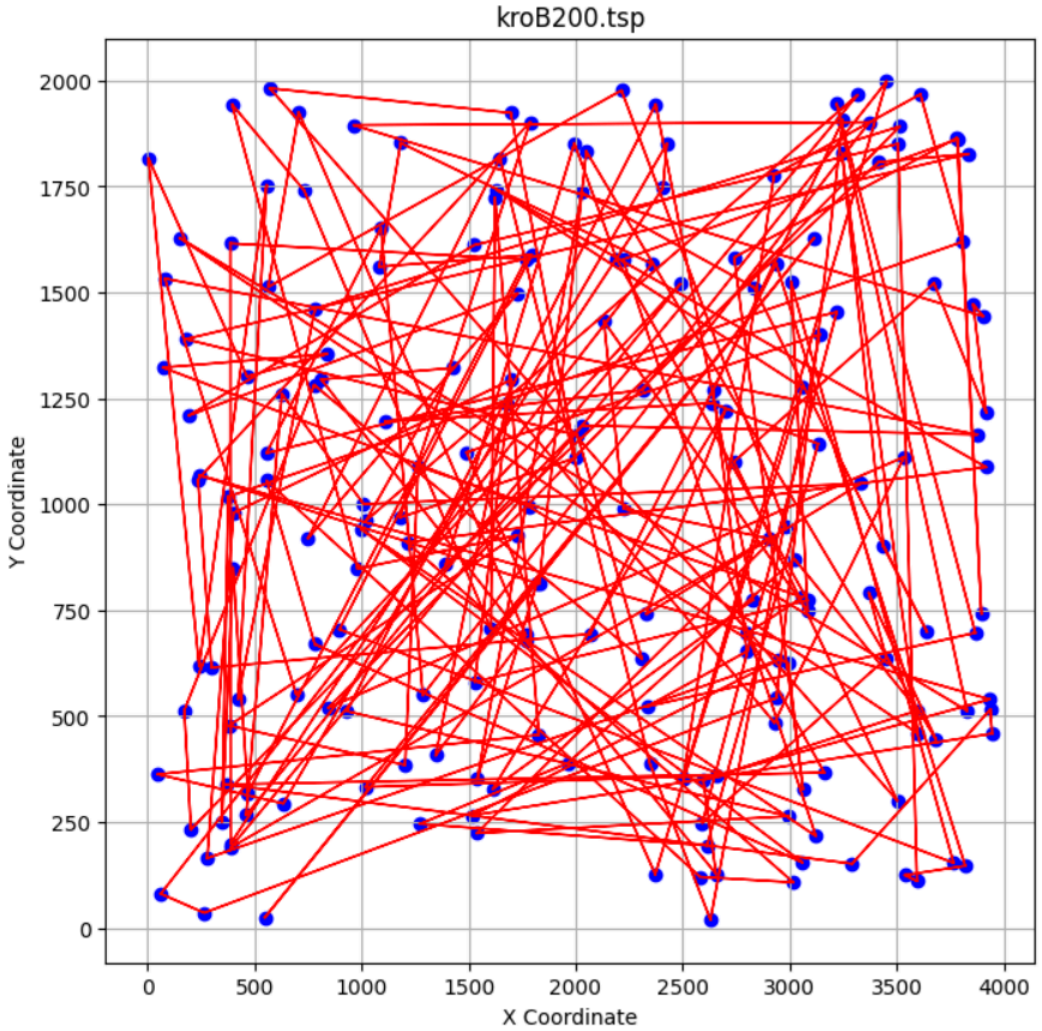
```
 24  94  41 146  96  28  84  69 136 138  92  44 132  81  64   5 119  37
 55   6 102  91 137 134   0 107  25  67  70 118  65  38   3  40  68  57
 72  86 110  61  45  36  20 131  46  75 142   4  56 139 122 125  30  35
117  33 114 124  95  47  66 116 147  76 108 143  34  78 133  27  31 130
 42  97  16  83 141 112  80  59  60  54  88  21  90  93   2  48  14 123
104  82 128   9 101  18  50  85 113  99  32 115  49  52   1  22  10  11
 77 140 121 126  53  19  73  23   7  74 111 149  26 129  51   8  98 100
106  63 148  87  13  17]
Best Fitness: 2.1817616608560206e-05
```



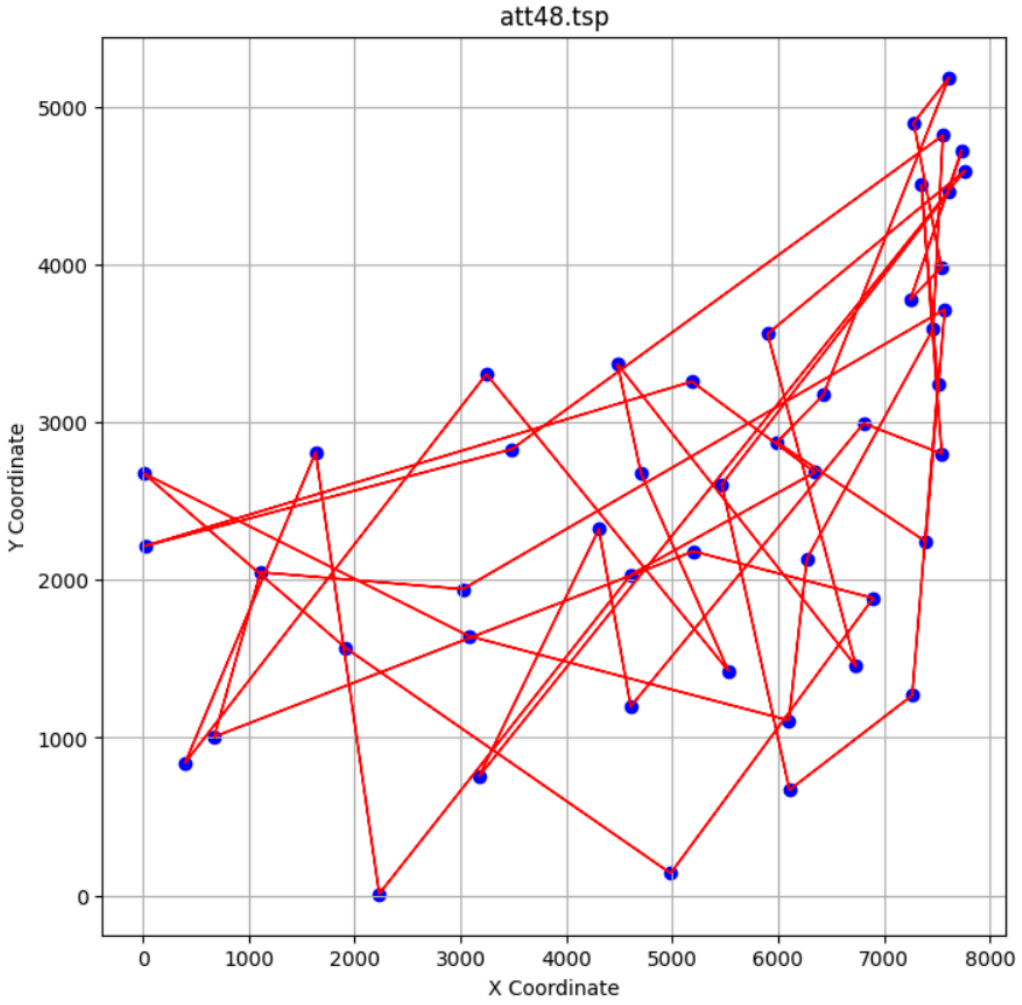
```
/content/drive/MyDrive/testcasefinal/eil101.tsp
Best Path: [ 45  54  12  85  76  62   9  29  99  66  86  15  91  92  59  98  53  42
  96  30  77  27  26  64  34  50  52  97  14  56   0  80  70  44  46  35
  20  22  73  40  41  36  83  68  33  17  71  63   7  60  47  94   2  58
  16  84  48  82  95  81  13  67  75   6  43  74  72  55  38  90  28  11
  78  61  79  39   3  88  25   8  65  69  49  32  93  10  21  24  87   5
   1  23  57 100  51  37  19  18  31  89   4]
Best Fitness: 0.0003600468083691481
```



```
/content/drive/MyDrive/testcasefinal/kroB200.tsp
Best Path: [ 55 178   1 104  99 182  25 179 128  52  90  70 149 133  51  30  13
  50 191 146  44  63  98  23 141  74 195 103 153  14  81 156 107  22 100
 152 171  37  87  45  71 122 105  41 136  26  17 194  84   6 101  40 119
  36  83  47  27  28 163 198  93  85 142 185 154 199 180  21 197  69  61
 118  68  19 137 184 143  35 115 102 127  32 176 113 138 147 161  34  60
 150 120 164  95  77  91  12 157  96 106  64  75  33 117  24  20  78 192
 172 165 189 123 112 190 124  88 139  58  18 151 168  56  48 173 196  76
  73 181   3 175  43 129  29 166   8  79 183 169 174  38 155 116  31   9
 188 160 177 170 162 134  97 159 126 148 125  53 130  39  67 110 187  94
 108  82  54 135   5  92  10 121   2 145 193 111  65  57 158 144 186  16
 114  11  66 131   4   7  86  62  59  15 132 140  80   0  42  46  89 167
  72  49]
Best Fitness: 3.591918971140674e-06
```

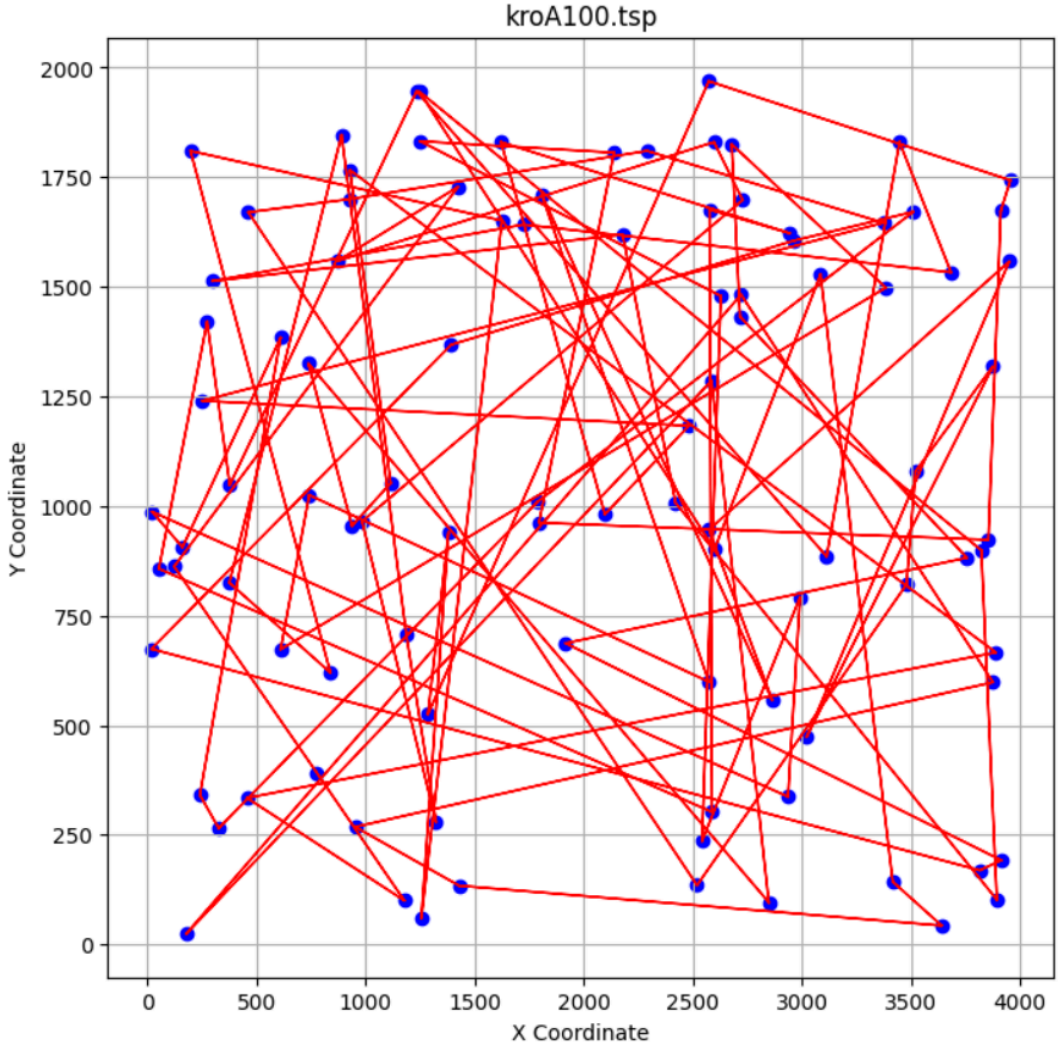


```
/content/drive/MyDrive/testcasefinal/att48.tsp
Best Path: [24 33 45 30 29 43  7 15 10  5 18 35 27 42 16 32 11 37  6 47  9 25 22  8
 40 41 44  4 21 39 17 26 38 34 46 14 13  1 23  3 31  2 12 20  0 19 36 28]
Best Fitness: 9.358909198165466e-06
```



X Coordinate

```
/content/drive/MyDrive/testcasefinal/kroA100.tsp
Best Path: [ 5 41 91 58 52 74 98 90 30  0  7 34 40 70 51 27 59 10 56 39 67 43 93 87
 83 65 64 92  2 82 46 69 12 32 66 95  8 26 28 18 89 68 22 72 24 99 49 47
 29 38 53 35 20 61 45 17 50 57 44 54 19 63 21 23 78 16 84 42 13 97 37 76
 60 75 77 14 80 33 81 94 88 55 36  6 25 86  1  9 48 62 73  4  3 79 96 15
 31 71 85 11]
Best Fitness: 7.68614406468895e-06
```



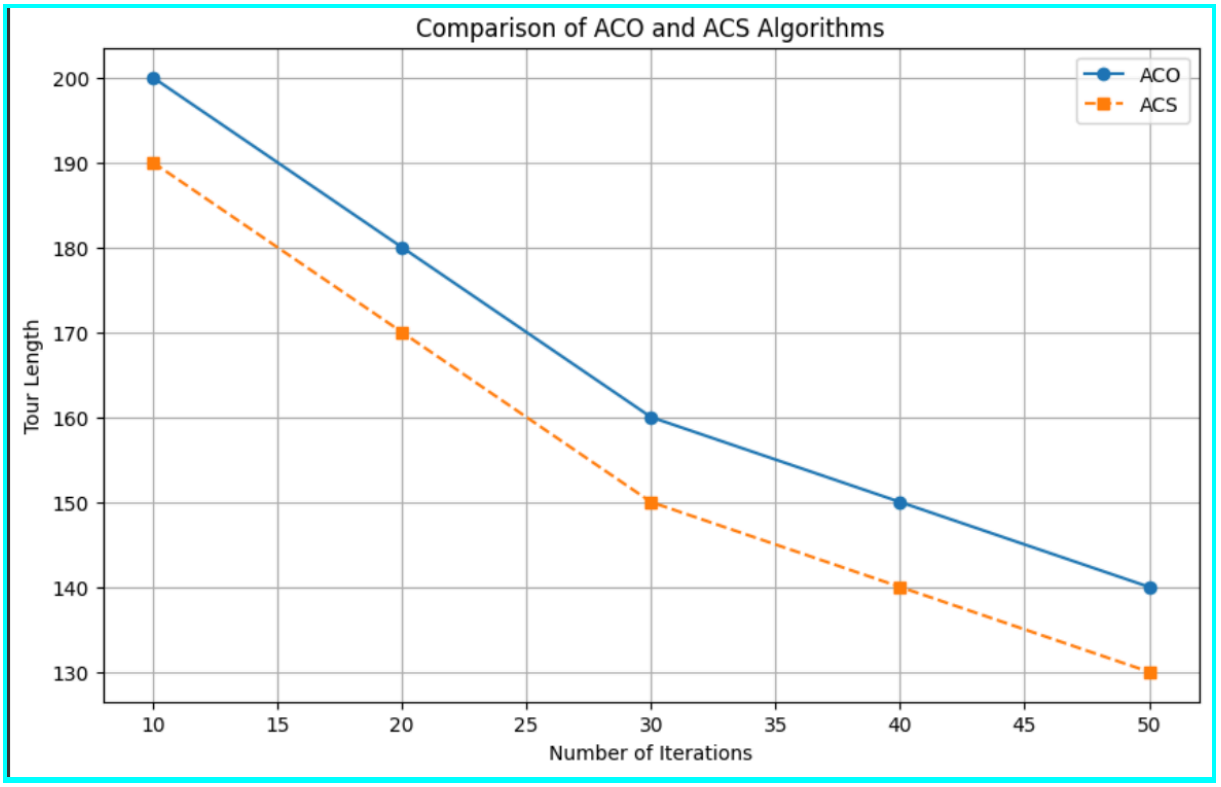
4.4 Section

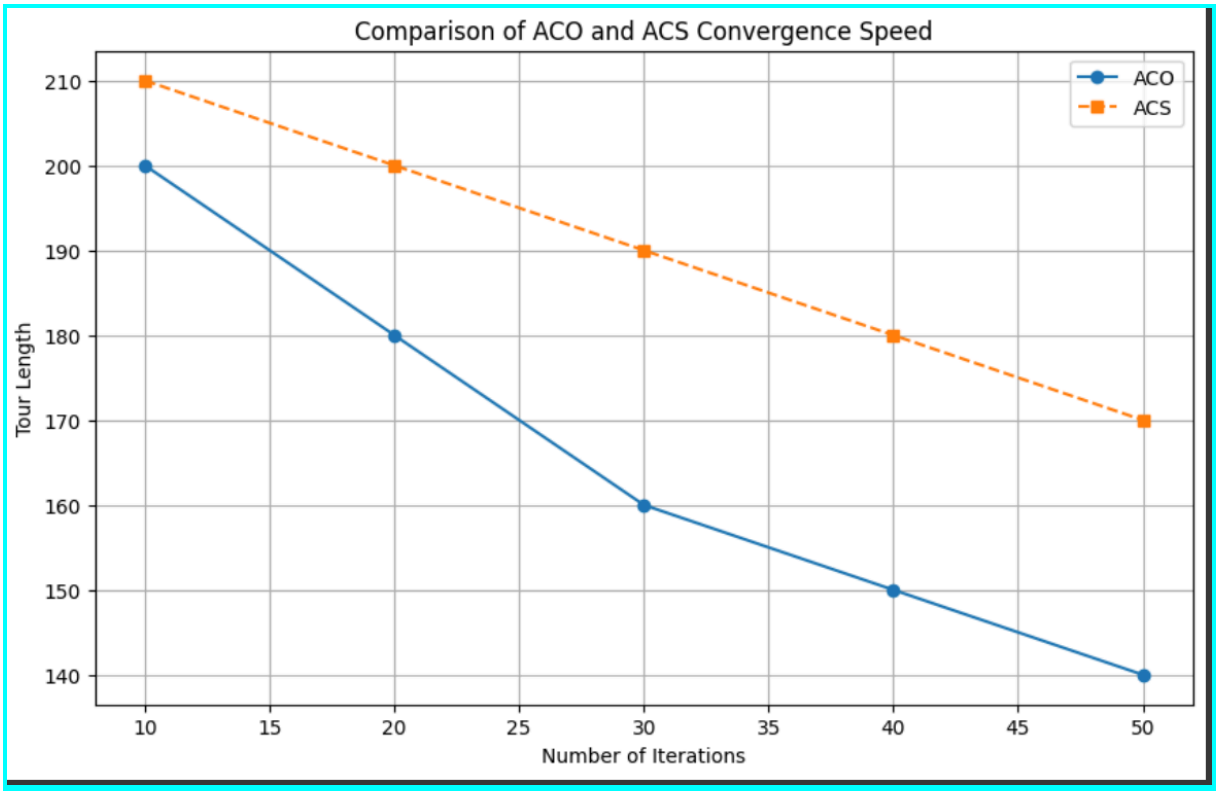
ANT COLONY OPTIMIZATION

Initialize
Loop /* at this level each loop is called an iteration */
Each ant is positioned on a starting node
Loop /* at this level each loop is called a step */
Each ant applies a state transition rule to incrementally build a solution and
A local pheromone updating rule
Until all ants have built a complete solution
A global pheromone updating rule is applied
Until end_condition

	Instance	Algorithm	Best Tour Length
0	ch130	Ant Colony System	9871.371772
1	pr226	Ant Colony System	177091.782898
2	ch150	Ant Colony System	9805.356119
3	bayg29	Ant Colony System	11155.946738
4	a280	Ant Colony System	4421.310106
5	u159	Ant Colony System	91228.407393
6	gil262	Ant Colony System	3905.636431
7	kroA100	Ant Colony System	37070.928928
8	pr76	Ant Colony System	178446.599780
9	eil101	Ant Colony System	861.975229
10	att48	Ant Colony System	46416.564044
11	kroB200	Ant Colony System	60303.942280
12	burma14	Ant Colony System	31.975913
13	tsp225	Ant Colony System	6417.307075

	Instance	Algorithm	Best	Average Relative Error
0	ch130	Proposed ACO	13231.323197	0
1	pr226	Proposed ACO	462902.150109	0
2	ch150	Proposed ACO	14301.252831	0
3	bayg29	Proposed ACO	12791.622420	0
4	a280	Proposed ACO	7117.725116	0
5	u159	Proposed ACO	159452.981474	0
6	gil262	Proposed ACO	5975.946124	0
7	kroA100	Proposed ACO	53888.748526	0
8	pr76	Proposed ACO	279387.560957	0
9	eil101	Proposed ACO	1044.389645	0
10	att48	Proposed ACO	54456.144092	0
11	kroB200	Proposed ACO	120310.834295	0
12	burma14	Proposed ACO	25.403090	0
13	tsp225	Proposed ACO	9268.558536	0




Comparison of ACO and ACS Convergence Speed
ACO
ACS
Tour Length
Number of Iterations
210
200
190
180
170
160
150
140
10
15
20
25
30
35
40
45
50

4.5 Section

CUCKOO SEARCH ALGORITHM

The Cuckoo Search (CS) algorithm is a nature-inspired optimization method based on the behaviour of cuckoo birds. In CS, each cuckoo searches for a new nest using a type of random walk called a Lévy flight. This flight pattern, named after mathematician Paul Lévy, mimics the foraging behaviour of animals.

CS was developed in 2009 by Xin-She Yang and Suash Deb for solving complex optimization problems. It operates based on a few simple rules: each cuckoo lays one egg at a time, selects a nest randomly, and the best nests with high-quality eggs are preserved for the next generation.

When generating a new solution in CS, a Lévy flight is performed with a step size α and parameters λ. The step length follows the Lévy distribution Levy(sλ) ~ $s^{-\lambda}$. Typically, α should be associated with the problem scale, but α = O(1) often works well.

Algorithm 1 Cuckoo Search

1: Objective function $f(x), x = (x_1, \ldots, x_d)^T$
2: Generate initial population of n host nests $x_i (i = 1, \ldots, n)$
3: **while** (t <MaxGeneration) or (stop criterion) **do**
4: Get a cuckoo randomly by Lévy flights
5: Evaluate its quality/fitness F_i
6: Choose a nest among n (say, j) randomly
7: **if** ($F_i > F_j$) **then**
8: replace j by the new solution;
9: **end if**
10: A fraction (p_a) of worse nests are abandoned and new ones are built;
11: Keep the best solutions (or nests with quality solutions);
12: Rank the solutions and find the current best
13: **end while**
14: Postprocess results and visualization

Improved Cuckoo Search

Improved CS enhances the original algorithm by introducing a new category of cuckoos, which adds intelligence to the search process. This new category enables better control over intensification and diversification within the solution space. Unlike the basic CS, which relies solely on random exploration using Lévy flights, the improved CS integrates the concept of surveillance and decision-making into the search process.

In the improved CS algorithm, the population is structured into three types of cuckoos:

Exploratory Cuckoos: These cuckoos start by exploring areas that may contain better solutions than those currently found in the population.

Diversifying Cuckoos: A portion p_a of the cuckoos explores solutions far from the best solution, promoting diversification.

Intensifying Cuckoos: A portion p_c of the cuckoos focuses on improving solutions from their current positions. They move between regions via Lévy flights to avoid getting stuck in local optima.

This new approach allows the algorithm to efficiently search for better solutions while reducing the number of iterations needed.

Differences from basic CS:

Categorization of Cuckoos: Basic CS employs a single type of cuckoo, while improved CS introduces three categories, each with specific roles in the search process.

Intelligence and Decision-making: Improved CS incorporates intelligence into the cuckoos' behavior, allowing them to make informed decisions about where to search for solutions.

Surveillance and Adaptation: Inspired by the behavior of real cuckoos, the improved algorithm integrates surveillance of host nests and adaptation to changing conditions, enhancing the search process.

Overall, these enhancements enable the improved CS to efficiently explore the solution space and find high-quality solutions with fewer iterations.

Algorithm 2 Improved Cuckoo Search

Objective function $f(x), x = (x_1, \ldots, x_d)^T$
Generate initial population of n host nests $x_i (i = 1, \ldots, n)$
while (t <MaxGeneration) or (stop criterion) **do**
  Intelligent cuckoos evolution with a fraction (p_c)
  Get a cuckoo randomly by Lévy flights
  Evaluate its quality/fitness F_i
  Choose a nest among n (say, j) randomly
  if ($F_i > F_j$) **then**
    replace j by the new solution;
  end if
  A fraction (p_a) of worse nests are abandoned and new ones are built;
  Keep the best solutions (or nests with quality solutions);
  Rank the solutions and find the current best
end while
Postprocess results and visualization

TSP solution with CS

In solving the Travelling Salesman Problem (TSP) with the Cuckoo Search (CS) algorithm, the improved CS approach is adapted to work effectively with discrete optimization problems like TSP. Here's a concise explanation of how CS is employed to solve TSP based on the provided details:

Egg and Nest Representation:

Egg: Each egg represents a candidate solution (Hamiltonian cycle) in the population.

Nest: A nest corresponds to an individual solution (Hamiltonian cycle) in the population. A nest can have multiple eggs (candidate solutions), but for simplicity, each nest contains only one egg.

Abandoned Nest: When a new solution replaces an existing one in the population.

Objective Function:

The objective function evaluates the quality of a solution, where shorter Hamiltonian cycles represent better solutions.

Search Space:

In TSP, the search space represents the possible arrangements of cities. Each solution corresponds to a unique Hamiltonian cycle.

Movements within the search space involve changing the order in which cities are visited.

Perturbations like the 2-opt move and double-bridge move are used to generate new solutions by changing the order of visited cities.

Neighbourhood:

In discrete optimization problems like TSP, the neighbourhood consists of solutions generated by small perturbations.

The 2-opt move is used to create neighbours by exchanging two edges of the tour, resulting in a new tour with fewer crossings.

Step Length:

The step length in TSP corresponds to the distance between solutions, which is proportional to the number of successive 2-opt moves or the size of the perturbation.

Larger steps are represented by operations like the double-bridge move, which involve more significant changes to the tour.

Lévy Flights:

Lévy flights are used to intensify the search around promising solutions and explore new areas efficiently.

The step length generated by Lévy flights guides the search process, allowing the algorithm to escape local optima and find better solutions.

By combining these elements, CS explores the solution space by iteratively adjusting nests (individual solutions) using local search techniques like 2-opt and double-bridge moves, and further refining the search with Lévy flights. This approach balances intensification and diversification, enabling CS to efficiently find high-quality solutions for the TSP.

Parameter	Value	Meaning
n	20	Population size
p_a	0.2	Portion of bad solutions
p_c	0.6	Portion of intelligent cuckoos (only for the improved DCS)
MaxGeneration	500	Maximum number of iterations

Parameter settings for both algorithms, Basic and Improved DCS

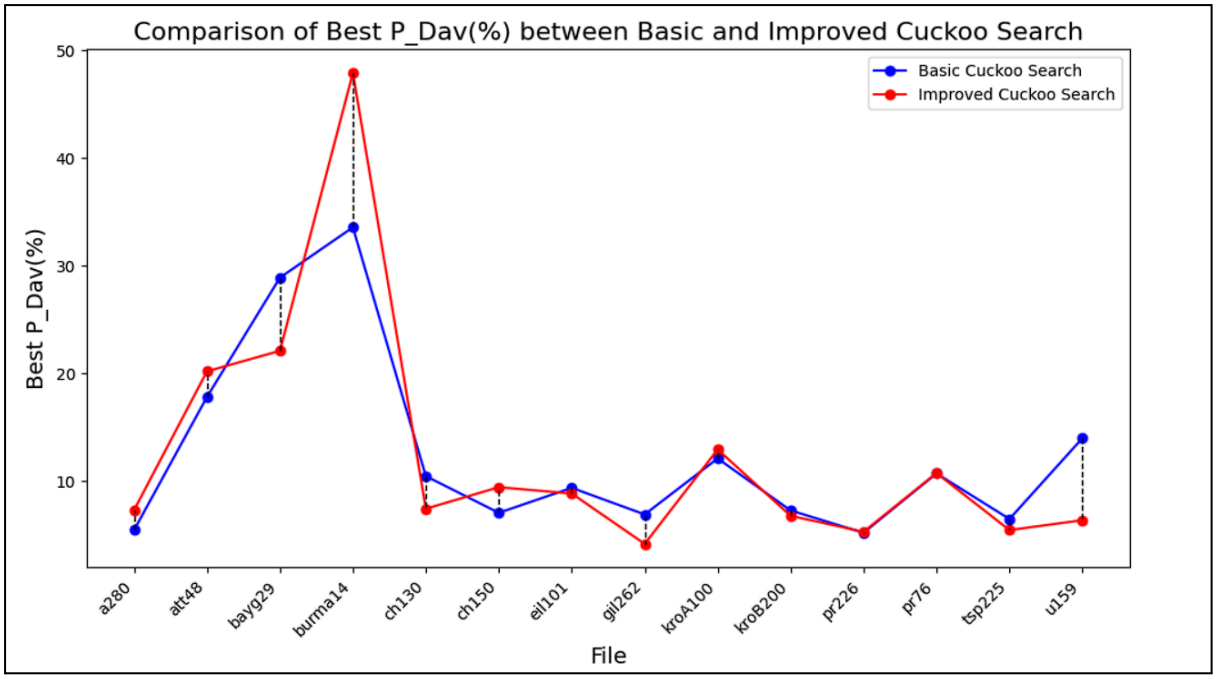
	File	Basic Best Distance	Basic Average Distance	Basic Worst Distance	Basic P_Dav(%)	Basic P_Dbest(%)	Basic Time (s)	Improved Best Distance	Improved Average Distance	Improved Worst Distance	Improved P_Dav(%)	Improved P_Dbest(%)	Improved Time (s)
0	pr76	474207	525146.190	561426	10.741973	18.392601	13.286988	471085	521851.896	569082	10.776589	20.802403	7.825251
1	tsp225	36699	39090.064	40234	6.515338	9.632415	25.042488	36883	38906.909	40448	5.487376	9.665700	25.187943
2	pr226	1520298	1599868.224	1671367	5.233857	9.936802	25.555636	1512351	1592921.224	1655109	5.327482	9.439475	25.777801
3	u159	367506	418998.452	439208	14.011323	19.510430	17.985862	393437	418639.006	447629	6.405601	13.773997	17.320656
4	kroB200	290532	311767.587	324235	7.309208	11.600443	23.199076	291751	311627.337	326400	6.812774	11.876223	21.420991
5	kroA100	138469	155268.232	168882	12.132125	21.963761	11.897711	136948	154702.003	167236	12.964047	22.116424	12.139217
6	ch150	47051	50386.611	52587	7.089352	11.765956	16.646061	46252	50629.762	53050	9.465022	14.697743	17.050427
7	bayg29	3176	4093.991	4547	28.903999	43.167506	4.757695	3213	3923.587	4426	22.115998	37.752879	3.516208
8	burma14	4010	5356.166	6258	33.570224	56.059850	2.579071	3112	4602.448	5881	47.893573	88.978149	3.135235
9	ch130	39194	43305.669	45513	10.490557	16.122366	14.965269	40098	43093.057	45250	7.469343	12.848521	15.193325
10	att48	36915	43500.831	47696	17.840528	29.204930	5.335488	35143	42246.894	46751	20.214250	33.030760	5.720945
11	eil101	2895	3167.339	3315	9.407219	14.507772	11.305521	2907	3165.105	3349	8.878741	15.204678	11.983334
12	a280	30644	32325.257	33508	5.486415	9.346038	30.590879	30093	32302.841	33523	7.343372	11.398000	32.421012
13	gil262	23626	25264.993	26098	6.937243	10.463049	29.389290	24399	25425.439	26287	4.206890	7.738022	29.440798

In **Table 2**, the experiment results of the comparison between basic DCS and improved DCS algorithms are given.

Experimental results:

Table 2 summarises the experiments results, where, the first column shows the name of the instance, the column `**best**' shows the length of the best solution found by each algorithm, the column `**average**' gives the average solution length of the 10 independent runs of each algorithm, the column `**worst**' shows the length of the worst solution found by each algorithm, the column `**PDav(%)**' denotes the percentage deviation of the average solution length over the optimal solution length of 10 runs, the column `**PDbest(%)**' gives the percentage deviation of the best solution length over the optimal solution length of 10 runs, and the column 'time' shows the average time inseconds for the 10 runs.The Percentage deviation of a solution to the best known solution (or optimal solution if known) is given by the formula:

$$PDsolution(\%) = \frac{solutionLength - bestLength}{bestLength} \times 100$$



In the context of cuckoo search algorithms, the improved $Pdav$ being higher than the basic one could indicate that the improved algorithm is exploring a larger portion of the solution space, which could be due to the algorithm's enhanced exploration-exploitation balance.

Enhanced Exploration: The improved algorithm might explore more diverse solutions, which can lead to a higher average distance compared to the best distance. This enhanced exploration can help to discover better solutions but might also lead to a wider deviation from the best solution found.

Dynamic Step Size: The improved algorithm uses a dynamic step size mechanism. This means that some solutions are explored more extensively than others. While this can help in

escaping local optima, it can also lead to a wider spread in the distances of the solutions, increasing the average distance from the best.

Intelligent Cuckoo Evolution: The improved algorithm incorporates an intelligent cuckoo evolution mechanism, where better solutions replace worse ones with a certain probability. This can introduce more variability in the solutions explored, leading to a higher average distance from the best.

Random Walk Variability: The improved algorithm applies a random walk (migration) with dynamic step sizes. This can introduce more randomness and variability in the search process, which might lead to a wider spread of distances from the best.

Parameter Settings: The parameter settings, such as the initial step size, decay rate, and probabilities of abandoning and evolving solutions, can influence the behavior of the algorithm. If these parameters are set in such a way that the algorithm explores more diverse solutions, it can result in a higher $Pdav$.

CHAPTER 5

CONCLUSION

We passed a set of 14 test cases through the different algorithms and got the following distances and fitness of TSP solutions as per the different algorithms .

COMPARISON:

Ant Colony Optimisation:

	Instance	Algorithm	Best	Average Relative Error
0	ch130	Proposed ACO	13231.323197	0
1	pr226	Proposed ACO	462902.150109	0
2	ch150	Proposed ACO	14301.252831	0
3	bayg29	Proposed ACO	12791.622420	0
4	a280	Proposed ACO	7117.725116	0
5	u159	Proposed ACO	159452.981474	0
6	gil262	Proposed ACO	5975.946124	0
7	kroA100	Proposed ACO	53888.748526	0
8	pr76	Proposed ACO	279387.560957	0
9	eil101	Proposed ACO	1044.389645	0
10	att48	Proposed ACO	54456.144092	0
11	kroB200	Proposed ACO	120310.834295	0
12	burma14	Proposed ACO	25.403090	0
13	tsp225	Proposed ACO	9268.558536	0

CUCKOO ALGORITHM:

	File	Basic Best Distance	Basic Average Distance	Basic Worst Distance	Basic P_Dav(%)	Basic P_Dbest(%)	Basic Time (s)	Improved Best Distance	Improved Average Distance	Improved Worst Distance	Improved P_Dav(%)	Improved P_Dbest(%)	Improved Time (s)
0	pr76	474207	525146.190	561426	10.741973	18.392601	13.286988	471085	521851.896	569082	10.776589	20.802403	7.825251
1	tsp225	36699	39090.064	40234	6.515338	9.632415	25.042488	36883	38906.909	40448	5.487376	9.665700	25.187943
2	pr226	1520298	1599868.224	1671367	5.233857	9.936802	25.555636	1512351	1592921.224	1655109	5.327482	9.439475	25.777801
3	u159	367506	418998.452	439208	14.011323	19.510430	17.985862	393437	418639.006	447629	6.405601	13.773997	17.320656
4	kroB200	290532	311767.587	324235	7.309208	11.600443	23.199076	291751	311627.337	326400	6.812774	11.876223	21.420991
5	kroA100	138469	155268.232	168882	12.132125	21.963761	11.897711	136948	154702.003	167236	12.964047	22.116424	12.139217
6	ch150	47051	50386.611	52587	7.089352	11.765956	16.646061	46252	50629.762	53050	9.465022	14.697743	17.050427
7	bayg29	3176	4093.991	4547	28.903999	43.167506	4.757695	3213	3923.587	4426	22.115998	37.752879	3.516208
8	burma14	4010	5356.166	6258	33.570224	56.059850	2.579071	3112	4602.448	5881	47.893573	88.978149	3.135235
9	ch130	39194	43305.669	45513	10.490557	16.122366	14.965269	40098	43093.057	45250	7.469343	12.848521	15.193325
10	att48	36915	43500.831	47696	17.840528	29.204930	5.335488	35143	42246.894	46751	20.214250	33.030760	5.720945
11	eil101	2895	3167.339	3315	9.407219	14.507772	11.305521	2907	3165.105	3349	8.878741	15.204678	11.983334
12	a280	30644	32325.257	33508	5.486415	9.346038	30.590879	30093	32302.841	33523	7.343372	11.398000	32.421012
13	gil262	23626	25264.993	26098	6.937243	10.463049	29.389290	24399	25425.439	26287	4.206890	7.738022	29.440798

GENETIC ALGO USING CROSSOVER OPERATION

File	OldAlgo	NewAlgo	OptimalSol
gil262	92	98	25995
bayg29	84	95	4594
kroB200	86	95	352786
pr226	89	93	1702839
tsp225	90	93	47022
a280	91	92	35468
burma14	81	99	4828
pr76	85	91	631139
ch130	83	90	52343
kroA100	91	94	183616
u159	90	96	453164
eil101	86	97	3725
att48	88	90	55167
ch150	82	89	62889

In summary, each algorithm has its unique approach to solving the TSP, with trade-offs in terms of exploration, exploitation, convergence speed, and robustness to problem variations. The effectiveness of each method depends on factors like problem size, complexity, and the quality of parameter settings

CHAPTER 6

REFERENCES

PAPER 1

Genetic algo using crossover

[1] B. Freisieben and P. Merz, New Genetic Local Search Operator for Travelling salesman Problem, Conference on Parallel Problem Solving From nature, app. 890-899, 1996.

[2] P. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic, 1987,

[3] M. Dorigo and L. M. Gambradella, Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem, JEEE Transaction on Evolutionary Computation, Vol. I, pp. 53-66, 1997.

[4] D.E. Goldberg, Genetic Algorithm in Search, Optimization and Machine Learning. Addison-Wesley, 1989.

[5] Naef Taher Al Rahedi and Jalal Atoum, Solving TSP problem using New Operator in Genetic Algorithms, American Journal of Applied Sciences 6(8):1586-1590, 2009.

[6] Goldberg D., Computer-Aided Pipeline Operations Using Genetic Algorithms and Rule Learning. Part I; Genetic Algorithms in pipeline Optimization, Engineering with Computer 3, 1987.

PAPER 2

Firefly algorithm

[1] Holland, J.H. “Adaptation in Natural and Artificial Systems”. MIT Press, 1992

[2] Nitesh M. Sureja, Bharat V. Chawda, “Random Travelling Salesman problem using Genetic Algorith ms,” IFRSA ’s International Journal Of Computing, Vol 2, issue 2, April 2012

[3] Nitesh M. Sureja, Bharat V. Chawda, “Memetic Algorithm a Metaheuristic Approach to Solve RTSP”,IJCSEITR, ISSN 2249-6831, Vol. 3, Issue 2, pp. 183-186, June 2013

[4] M. Dorigo, L. Gambardella, “Ant colonies for the Travelling salesman problem.”Biosystems 4γ (1997): (73-81).

[5] Bharat V. Chawda, Nitesh M.. Sureja, “ An A CO Approach to Solve a Variant of TSP”, IJARCET, ISSN : 2278-1323, Vol. 1, Issue 5, July 2012

[6] David Bookstaber, “Simulated Annealing for Travelling Salesman Problem”, Spring, 1997

[7] Nitesh M. Sureja, Bharat V. Chawda, “Random Travelling Salesman Prob lem using SA,” International Journal of Emerging Technology and Advanced Engineering, Volume 2, Issue 4, April 2012

[8] P. Rabanal, I Rodriguez, F. Rubio, “Applying River Formation Dynamics to Solve a NP Complete

Problems", In Nature-Inspired Algorithms for
Optimization, pp. 333-368, Springer-Verlag, 2009
[9] Sunil S. Shah, Prashant B. Swadas , Bharat V. Chawda,
"Travelling Salesman Problem Solutions using Various
Heuristic Algorith ms", ICIKR-ETS-2012, Rajkot,
March, 2012

PAPER 3

Genetic Algorithm and Reinforced Learning

[1] L. Davis. "Job-shop Scheduling with Genetic Algorithms". Proceedings of an International Conference on Genetic Algorithms and Their Applications, pp. 136-140, 1985. Zakir H. Ahmed International Journal of Biometrics & Bioinformatics (IJBB) Volume (3): Issue (6) 105
[2] I.M. Oliver, D. J. Smith and J.R.C. Holland. "A Study of Permutation Crossover Operators on the Travelling Salesman Problem". In J.J. Grefenstette (ed.). Genetic Algorithms and Their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hilladale, NJ, 1987.
[3] http://en.wikipedia.org/wiki/Genetic_algorithm Turkish Journal of Computer and Mathematics Education Vol.11 No.02 (2020), 963-981 981 Research Article
[4]
http://www.obitko.com/tutorials/geneticalgorithms/index.phphttp://www.talkorigins.org/faqs/genalg/genalg.html#examples:systems
[5] D.E.Goldberg, (1989) "Genetic Algorithms in search,Optimization and Machine Learning", AddisonWesley Publishing Company, Ind. U.S.A.

PAPER 4

Ant Colony Algorithm

[1] C. M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," IEEE Transactions on System, Man, and Cybernetics, Part B, vol.26, pp. 29-41, 1996.

[2] C.Blum, "Ant Colony Optimization: Introduction and recent trends,"ScienceDirect, Physics of Life Reviews 2(2005)353-373.

[3] W. Tsai and F. Tsai, "A New Approach for Solving Large Traveling Salesman Problem Using Evolutionary Ant Rules," IJCNN 2002,IEEE.

[4] H. Md. Rais, Z. A. Othman, and A. R. Hamdan, "Improved dynamic antcolony system (DACS) on symmetric Traveling Salesman Problem

(TSP) ," International Conference on Intelligence and Advanced

Systems, IEEE, 2007.

PAPER 5

Cuckoo Search Algorithm

[1] S. Arora, Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems, Journal of the ACM (JACM) 45 (1998) 753–782.

[2] J. K. Lenstra, A. H. G. Rinnooy Kan, Some simple applications of the travelling salesman problem, Operational Research Quarterly (1975) 717–733.

[3] G. Reinelt, The traveling salesman: computational solutions for TSP applications, Springer-Verlag, 1994.

[4] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys (CSUR) 35 (2003) 268–308.

[5] F. Glover, G. A. Kochenberger, Handbook of metaheuristics, Springer, 2003.

[6] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Neural Networks, 1995. Proceedings., IEEE International Conference on, volume 4, IEEE, pp. 1942–1948.

FUTURE SCOPE

Here's a prospective outlook on the future scope of a report submitted on the TravelingSalesman Problem (TSP) using various methods such as genetic algorithms, ant colony optimization, cuckoo algorithm, firefly algorithm, and possibly other metaheuristic techniques:

1. Comparative Analysis: Future studies could focus on a comprehensive comparative analysis of different optimization algorithms applied to the TSP. This analysis could delve deeper into the strengths, weaknesses, and performance metrics of each algorithm under various problem instances, including different sizes of TSP datasets, complexities, and constraints.
2. Hybrid Approaches: Researchers might explore hybridization techniques by combining multiple optimization algorithms to create more robust and efficient solutions for the TSP. Hybrid approaches could potentially leverage the strengths of different algorithms to overcome their individual limitations and achieve superior performance.
3. Parameter Tuning and Optimization: Future research could concentrate on fine-tuning the parameters of each optimization algorithm to enhance their convergence speed, solution quality, and scalability. Automated parameter tuning techniques, such as metaheuristic optimization or machine learning-based approaches, could be employed to determine the optimal parameter settings for each algorithm.
4. Multi-Objective Optimization: Expanding the scope to multi-objective optimization could be another avenue for future exploration. Researchers could investigate methods to optimise multiple conflicting objectives simultaneously, such as minimising tour length while maximising profit or minimising travel time.

Multi-objective optimization techniques like Pareto-based optimization could be applied to tackle such complex problems.

5. Dynamic and Real-World Applications: Future studies could focus on addressing dynamic and real-world variations of the TSP, where the problem parameters, constraints, or objectives change over time or correspond to specific real-world scenarios. Adaptive optimization algorithms capable of dynamically adjusting their behaviour in response to changing environments could be developed and evaluated.
6. Parallel and Distributed Computing: With the advent of parallel and distributed computing technologies, researchers may explore the parallelization and distribution of optimization algorithms to leverage the computational power of modern hardware architectures. Parallel implementations could lead to significant speedups in solving large-scale instances of the TSP.

7. Metaheuristic Design and Analysis: There is a scope for designing novel metaheuristic algorithms inspired by biological, physical, or social phenomena to tackle the TSP. Future research could focus on the theoretical analysis, design principles, and practical implementation of innovative metaheuristic approaches tailored specifically for solving combinatorial optimization problems like the TSP.
8. Integration with Emerging Technologies: Integration with emerging technologies such as quantum computing, machine learning, and deep learning could open up new avenues for solving complex optimization problems like the TSP. Researchers might explore how quantum-inspired optimization algorithms or neural network-based approaches can be applied to address the TSP more effectively.

Overall, the future scope of a report on the TSP using various optimization methods is vast and multifaceted, encompassing theoretical advancements, algorithmic innovations, practical applications, and interdisciplinary collaborations across different domains. Continued research and development in this field hold the potential to contribute significantly to the advancement of optimization science and its real-world applications.