417534: Computer Laboratory III

Teaching Scheme:

PR: 02 Hours/Week

Credit 02

Examination Scheme and Marks

Term Work (TW): 50

Marks Practical (PR): 25 Marks

List of Assignments: -

Part I:

1. Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.
2. Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations.
3. Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.
4. Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.
5. Implementation of Clonal selection algorithm using Python.
6. Create and Art with Neural style transfer on given image using deep learning.

Part II:

1. To apply the artificial immune pattern recognition to perform a task of structure damage Classification.
2. Implement DEAP (Distributed Evolutionary Algorithms) using Python.
3. Implement Ant colony optimization by solving the Traveling salesman problem using python
   Problem statement- A salesman needs to visit a set of cities exactly once and return to the original city. The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.
4. Create and Art with Neural style transfer on given image using deep learning.

Course Objectives:

- To understand the fundamentals of a distributed environment in complex application
- To introduce the concepts inspired by the human immune system and their application in problem-solving and optimization
- To make students aware about security issues and protection mechanisms for distributed environments
- To familiarize with various evolutionary algorithms and optimization techniques inspired by natural evolution processes

Course Outcomes:

After completion of the course, learners should be able to-

CO1: Apply the principles on which the internet and other distributed systems are based

CO2: Understand and apply the basic theoretical concepts and algorithms of distributed systems in problem solving

CO3: Apply fuzzy logic techniques to model and solve problems

CO4: Design and implement evolutionary algorithms to solve optimization and search

problems in diverse domains

CO5: Design and implement artificial immune system algorithms to solve complex problems in different domains

Learning Resources

Text Books:

1. C. Manning, P. Raghavan, and H. Schütze, "Introduction to Information Retrieval", Cambridge University Press, 2008
2. Ricardo Baeza-Yates, Berthier Riberio–Neto, "Modern Information Retrieval", Pearson Education, ISBN: 81-297-0274-6
3. C.J. Rijsbergen, "Information Retrieval", 2nd edition, ISBN: 978-408709293
4. Ryan Mitchell, "Web Scraping with Python", O'reilly

Reference Books:

1. S. Buttcher, C. Clarke and G. Cormack, "Information Retrieval: Implementing and Evaluating Search Engines" MIT Press, 2010, ISBN: 0-408-70929-4
2. Amy N. Langville and Carl D. Meyer, "Google's PageRank and Beyond: The Science of Search Engine Rankings", Princeton University Press, ISBN: 9781400830329

e-Books:

1. https://induraj2020.medium.com/implementation-of-ant-colony-optimization-using-pythonsolve-traveling-salesman-problem-9c14d3114475
2. https://blog.tensorflow.org/2018/08/neural-style-transfer-creating-art-with-deep-learning.html

3. https://www.professionalcipher.com/

CO-PO MAPPING

Course Outcome (COs)	Program Outcomes (POs)											
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	1	2	1	1	2						1	
CO2	2	1	1	2	2							
CO3	1	2		1	2						1	
CO4	1		1	2	1							
CO5	1	2	1	1	2						1	

Assignment No. 1

Title:

Distributed Application using RPC for Remote Computation

Problem Statement:

Design a distributed application using RPC for remote computation where client submits an integer value to the server and server calculates factorial and returns the result to the client program.

Theory:

Remote Procedure Call (RPC) is a powerful mechanism for enabling communication between processes or programs across a network. It allows a client program to execute procedures or functions on a remote server as if they were local, abstracting away the complexities of network communication.

RPC Components:

- Client: The program that initiates the RPC call.
- Server: The program that executes the requested procedure.
- Stub: A client-side proxy for the remote procedure, responsible for marshalling parameters and sending requests over the network.
- Skeleton: A server-side proxy for the remote procedure, responsible for unmarshalling parameters and invoking the actual procedure.

RPC Workflow:

- The client invokes a procedure on the stub, passing parameters.
- The stub marshals the parameters into a message and sends it over the network to the server.
- The server's skeleton receives the message, unmarshals the parameters, and invokes the corresponding procedure.
- The procedure executes on the server, producing a result.
- The result is marshalled into a response message and sent back to the client.
- The client's stub receives the response, unmarshals the result, and returns it to the client program.

Procedure:

Designing the RPC Interface:

- Define the interface between the client and server. This includes specifying the method(s) that the client can call on the server.
- For this application, define a method calculate_factorial(n) that takes an integer n as input and returns the factorial of n.

Implementing the Server:

- Set up a server program that listens for incoming RPC requests from clients.
- Implement the calculate_factorial(n) method on the server. Use an appropriate algorithm (e.g., recursive or iterative) to calculate the factorial of n.

- Expose the method to clients by registering it with the RPC framework.

Implementing the Client:

- Create a client program that sends an RPC request to the server.
- Call the calculate_factorial(n) method on the server, passing the desired integer n as an argument.
- Receive and process the result returned by the server.

Testing:

- Test the client-server interaction with various input values to ensure correctness and robustness.
- Handle potential error cases gracefully, such as invalid input or server unavailability.

Conclusion:

Thus, we have successfully implemented a distributed application using RPC for remote computation.

Assignment No. 2

Title:
Fuzzy set Operations & Relations

Problem Statement:
Implement Union, Intersection, Complement and Difference operations on fuzzy sets. Also create fuzzy relations by Cartesian product of any two fuzzy sets and perform max-min composition on any two fuzzy relations.

Theory:

Fuzzy Set Operations

1. **Define Fuzzy Sets:**
   - Choose a universe of discourse (U) representing the range of possible values for your variable.
   - Define two fuzzy sets, A and B, on the universe U using membership functions. Membership functions map elements in U to a degree of membership between 0 and 1. Common functions include triangular, trapezoidal, and Gaussian functions.
2. **Union (OR):**
   - Implement a function that takes two fuzzy sets (A, B) and their membership functions (μ_A, μ_B) as input.
   - For each element x in U, calculate the degree of membership in the union (μ_A U B)(x) using the maximum of individual memberships:
   - μ_A U B(x) = max(μ_A(x), μ_B(x))
3. **Intersection (AND):**
   - Implement a function that takes two fuzzy sets (A, B) and their membership functions (μ_A, μ_B) as input.
   - For each element x in U, calculate the degree of membership in the intersection (μ_A ∩ B)(x) using the minimum of individual memberships:
   - μ_A ∩ B(x) = min(μ_A(x), μ_B(x))
4. **Complement (NOT):**
   - Implement a function that takes a fuzzy set (A) and its membership function (μ_A) as input.
   - For each element x in U, calculate the degree of membership in the complement (μ_A^c)(x) by inverting the membership of A:
   - μ_A^c(x) = 1 - μ_A(x)
5. **Difference (A except B):**
   - Implement a function that takes two fuzzy sets (A, B) and their membership functions (μ_A, μ_B) as input.
   - For each element x in U, calculate the degree of membership in the difference (μ_A \ B)(x) using the membership of A and the complement of B:
   - μ_A \ B(x) = max(μ_A(x), μ_B^c(x))

Fuzzy Relations

1. **Cartesian Product:**
    - Define two fuzzy sets, A and B, on universes U and V respectively.
    - The Cartesian product of A and B creates a new fuzzy relation R on the universe U x V (cartesian product of U and V).
    - The membership function of R, μ_R((u, v)), represents the degree of relationship between element u in U and element v in V.
    - Implement a function that takes two fuzzy sets (A, B) and their membership functions (μ_A, μ_B) as input.
    - For each pair (u, v) in U x V, calculate the degree of membership in the relation using the minimum of individual memberships:
    - μ_R((u, v)) = min(μ_A(u), μ_B(v))
2. **Max-Min Composition:**
    - Define two fuzzy relations, R and S, on universes (U x V) and (V x W) respectively.
    - Max-min composition combines these relations to create a new fuzzy relation T on universe U x W.
    - The membership function of T, μ_T((u, w)), represents the composed relationship between element u in U and element w in W, considering the relationship between u and all elements v in V.
    - Implement a function that takes two fuzzy relations (R, S) and their membership functions (μ_R, μ_S) as input.
    - For each pair (u, w) in U x W, iterate through all elements v in V and calculate the intermediate values:
    - z_uv = min(μ_R((u, v)), μ_S((v, w)))
    - Use the maximum of these intermediate values as the degree of membership in the composed relation:

Conclusion:

Thus, we have successfully implemented a program for Fussy set operations& its relations.

Assignment No. 3

Title:
Load balancing algorithms

Problem Statement:
Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.

Theory:

Part 1: Simulating Client Requests

Procedure:

1. **Define Client Request:**
   - Create a class or data structure to represent a client request.
   - Include attributes like request ID, arrival time, processing time (randomly generated), and any additional relevant data.
2. **Client Arrival Process:**
   - Implement a function to generate client requests with random inter-arrival times. You can use libraries like random to generate random time intervals between requests.
   - Consider using a loop or a timer to simulate requests arriving at specific intervals or continuously over a period.
3. **Request Queue:**
   - Implement a data structure (e.g., list) to store pending client requests waiting for server assignment.

Part 2: Load Balancing Algorithms

Algorithms:

- **Round Robin:**
  - Maintain a list of available servers.
  - For each new request, assign it to the next server in the list and move that server to the end of the list.
  - This ensures each server receives requests in a round-robin fashion.
- **Least Connections:**
  - Maintain a counter for the number of active connections on each server.
  - Assign a new request to the server with the fewest active connections.
  - This distributes requests based on current workload.
- **Weighted Round Robin:**
  - Assign weights to each server based on processing power, memory, or other relevant factors.
  - Modify the round-robin algorithm to select servers based on their weights. Servers with higher weights receive requests more frequently.

Implementation:

1. **Server Management:**
    - Define a class or data structure to represent a server.
    - Include attributes like server ID, processing capacity (optional), and a flag indicating if it's busy processing a request.
2. **Load Balancer Class:**
    - Implement the chosen load balancing algorithm(s) within a load balancer class.
    - The class should have methods to:
        - Add a new request to the request queue.
        - Select a server based on the chosen algorithm.
        - Assign the request to the selected server and update its busy flag.
        - Optionally, track and report metrics like average waiting time, server utilization, etc.

Part 3: Simulation Loop

Procedure:

1. **Main Loop:**
    - Implement a loop that iterates for a set duration or processes a specific number of requests.
    - In each iteration:
        - Generate a new client request using the function from Part 1.
        - Add the request to the queue in the load balancer.
        - Call the load balancer's method to select a server and assign the request.
        - Optionally, update statistics and counters for monitoring performance.
2. **Output:**
    - Print or store relevant data after the simulation, such as:
        - Total number of requests processed.
        - Average waiting time for requests.
        - Server utilization (percentage of time servers were busy).
        - Comparison of results across different load balancing algorithms (if applicable).

Conclusion:

Thus, we have successfully implemented a program to simulate the request using Load Balancing Algorithm.

Assignment No. 4

Title:

Optimization of genetic algorithm parameter.

Problem Statement:

Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.

Software Requirement:

Python 3.9 and Jupiter/PyCharm/Any Equivalent Editor

Input:

Data Set

Output:

Spam and Not Spam Classification.

Theory:

Genetic Algorithm (GA):

Genetic Algorithms are heuristic optimization algorithms inspired by the process of natural selection and genetics. They are widely used for solving optimization problems. In hybrid genetic algorithm-neural network modeling, a genetic algorithm is often used to optimize the weights and biases of a neural network.

Parameters of Genetic Algorithm:

- Population Size (pop_size): The number of individuals in the population.
- Mutation Rate (mutation_rate): The probability that a gene (parameter) will be mutated during reproduction.
- Crossover Rate (crossover_rate): The probability that two individuals will exchange genetic information during crossover.
- Selection Method: The method used to select individuals for reproduction, such as tournament selection or roulette wheel selection.
- Termination Criteria: The condition under which the genetic algorithm terminates, such as reaching a maximum number of generations or achieving a desired fitness level.

Optimization:

Optimizing genetic algorithm parameters involves finding the combination of parameter values that results in the best performance of the hybrid model. This often requires experimentation and fine-tuning of parameters to achieve optimal results.

Implementation:

Steps for Optimization:

- Initialization: Initialize the genetic algorithm parameters with initial values.
- Create Hybrid Model: Implement the hybrid genetic algorithm-neural network model.

- Parameter Tuning: Experiment with different values of genetic algorithm parameters.
- Evaluation: Evaluate the performance of the hybrid model for each set of parameter values.
- Optimization: Select the parameter values that yield the best performance based on predefined criteria.
- Validation: Validate the optimized parameters on a separate dataset to ensure generalization.

Conclusion:

Thus, we have successfully implemented a program for Optimization of genetic algorithm parameter.

Assignment No. 5

Title:

Clonal selection algorithm

Problem Statement:

Implementation of Clonal selection algorithm using Python.

Theory:

Introduction:

Briefly introduce the concept of CLONALG and its inspiration from the biological immune system's clonal selection process.

Explain how CLONALG utilizes antibodies (candidate solutions) and affinity (fitness function) to iteratively improve solutions.

Core Principles:

- Antibody Representation: Define how you will represent candidate solutions in your implementation. This could be binary strings for binary optimization problems or real-valued vectors for continuous problems.
- Affinity Function: Design a function that evaluates the fitness or quality of each candidate solution (antibody) relevant to your chosen optimization problem.

CLONALG Phases:

- Initialization: Generate a random population of antibodies (candidate solutions).
- Selection: Select high-affinity antibodies (good solutions) for cloning based on their fitness function values.
- Cloning: Create clones of selected antibodies. The number of clones can be proportional to the antibody's affinity.
- Hypermutation: Introduce random mutations to the clones with a low mutation rate to promote diversity.
- Evaluation: Evaluate the affinity of all antibodies (original population and clones) using the fitness function.
- Selection and Replacement: Select the best antibodies (including original population and mutated clones) to form the next generation, replacing low-affinity ones.
- Termination: Stop the process when a termination criterion is met (e.g., reaching a maximum number of iterations or achieving a desired fitness value).

Conclusion:

Thus, we have successfully implemented a program for Clonal selection algorithm.

Assignment No. 6

Title:

Neural style transfer

Problem Statement:

Create and Art with Neural style transfer on given image using deep learning.

Theory:

Introduction:

Neural Style Transfer (NST):

- Neural Style Transfer is a deep learning technique that combines the content of one image (content image) with the style of another image (style image) to create a new image (generated image).
- It leverages convolutional neural networks (CNNs) to extract content and style features from the input images and optimize a generated image to minimize the content distance from the content image and the style distance from the style image simultaneously.

Key Steps of NST:

- Choose Content and Style Images: Select a content image (the image whose content you want to retain) and a style image (the image whose artistic style you want to apply).
- Preprocess Images: Preprocess the content and style images to prepare them for feeding into a CNN.
- Define Loss Functions: Define content loss and style loss functions to measure the difference between the features of the generated image, content image, and style image.
- Optimization: Use gradient descent or other optimization techniques to update the generated image to minimize the combined content and style loss.
- Generate Artistic Image: Generate the final artistic image by optimizing the generated image.

Conclusion:

Thus, we have successfully implemented a program for Neural style transfer.

Assignment No. 7

Title:
Artificial immune pattern recognition

Problem Statement:
To apply the artificial immune pattern recognition to perform a task of structure damage Classification.

Theory:

- AIS emerged in the mid-1980s with articles authored by Farmer, Packard and Perelson (1986) and Bersini and Varela (1990) on immune networks.
- In artificial intelligence, artificial immune systems (AIS) are a class of computationally intelligent, rule-based machine learning systems inspired by the principles and processes of the vertebrate immune system. The algorithms are typically modeled after the immune system's characteristics of learning and memory for use in problem-solving.
- The field of artificial immune systems (AIS) is concerned with abstracting the structure and function of the immune system to computational systems, and investigating the application of these systems towards solving computational problems from mathematics, engineering, and information technology.
- AIS is a sub-field of biologically inspired computing, and natural computation, with interests in machine learning and belonging to the broader field of artificial intelligence.
- Artificial immune systems (AIS) are adaptive systems, inspired by theoretical immunology and observed immune functions, principles and models, which are applied to problem solving.
- AIS is distinct from computational immunology and theoretical biology that are concerned with simulating immunology using computational and mathematical models towards better understanding the immune system, although such models initiated the field of AIS and continue to provide a fertile ground for inspiration. Finally, the field of AIS is not concerned with the investigation of the immune system as a substrate for computation, unlike other fields such as DNA computing.

Techniques

The common techniques are inspired by specific immunological theories that explain the function and behavior of the mammalian adaptive immune system.

- Clonal selection algorithm:
    - A class of algorithms inspired by the clonal selection theory of acquired immunity that explains how B and T lymphocytes improve their response to antigens over time called affinity maturation.
    - These algorithms focus on the Darwinian attributes of the theory where selection is inspired by the affinity of antigen–antibody interactions, reproduction is inspired by cell division, and variation is inspired by somatic hypermutation.

    - Clonal selection algorithms are most commonly applied to optimization and pattern recognition domains, some of which resemble parallel hill climbing and the genetic algorithm without the recombination operator.
- Negative selection algorithm:
    - Inspired by the positive and negative selection processes that occur during the maturation of T cells in the thymus called T cell tolerance. Negative selection refers to the identification and deletion (apoptosis) of self-reacting cells, that is T cells that may select for and attack self tissues.
    - This class of algorithms are typically used for classification and pattern recognition problem domains where the problem space is modeled in the complement of available knowledge.
    - For example, in the case of an anomaly detection domain the algorithm prepares a set of exemplar pattern detectors trained on normal (non-anomalous) patterns that model and detect unseen or anomalous patterns.
- Immune network algorithms:
    - Algorithms inspired by the idiotypic network theory proposed by Niels Kaj Jerne that describes the regulation of the immune system by anti-idiotypic antibodies (antibodies that select for other antibodies).
    - This class of algorithms focus on the network graph structures involved where antibodies (or antibody producing cells) represent the nodes and the training algorithm involves growing or pruning edges between the nodes based on affinity (similarity in the problems representation space).
    - Immune network algorithms have been used in clustering, data visualization, control, and optimization domains, and share properties with artificial neural networks.[10]
- Dendritic cell algorithms:
    - The dendritic cell algorithm (DCA) is an example of an immune inspired algorithm developed using a multi-scale approach. This algorithm is based on an abstract model of dendritic cells (DCs).
    - The DCA is abstracted and implemented through a process of examining and modeling various aspects of DC function, from the molecular networks present within the cell to the behaviour exhibited by a population of cells as a whole.
    - Within the DCA information is granulated at different layers, achieved through multi-scale processing.

Conclusion:

Thus, we have successfully implemented a program for Artificial immune pattern recognition.

Assignment No. 8

Title:
DEAP

Problem Statement:
Implement DEAP (Distributed Evolutionary Algorithms) using Python.

Theory:

Distributed Evolutionary Algorithms (DEAP):

- Distributed Evolutionary Algorithms are a class of evolutionary algorithms designed to solve complex optimization problems by distributing computation across multiple processors or computational nodes.
- DEAP offers scalability, efficiency, and parallelism, making it suitable for solving large-scale optimization problems.

DEAP Components:

- Population: A collection of candidate solutions (individuals) representing potential solutions to the optimization problem.
- Selection: The process of selecting individuals from the population for reproduction based on their fitness.
- Crossover: The process of combining genetic material from selected individuals to create new offspring.
- Mutation: The process of introducing random changes to the genetic material of individuals to maintain diversity.
- Evaluation: The process of evaluating the fitness of individuals in the population based on their performance on the optimization problem.
- Termination Criteria: The condition under which the optimization process terminates, such as reaching a maximum number of generations or achieving a desired fitness level.

Features of DEAP:

- Parallelism: DEAP utilizes parallel processing to evaluate fitness functions and explore the search space efficiently.
- Flexibility: DEAP provides a flexible framework for implementing various evolutionary algorithms, including Genetic Algorithms, Evolutionary Strategies, and Genetic Programming.
- Customization: DEAP allows customization of operators, selection methods, and termination criteria to tailor the algorithm to specific problem domains.

Conclusion:
Thus, we have successfully implemented a program for DEAP.

Assignment No. 9

Title:

Ant colony optimization using Traveling salesman problem

Problem Statement:

Implement Ant colony optimization by solving the Traveling salesman problem using python

Problem statement- A salesman needs to visit a set of cities exactly once and return to the original city. The task is to find the shortest possible route that the salesman can take to visit all the cities and return to the starting city.

Theory:

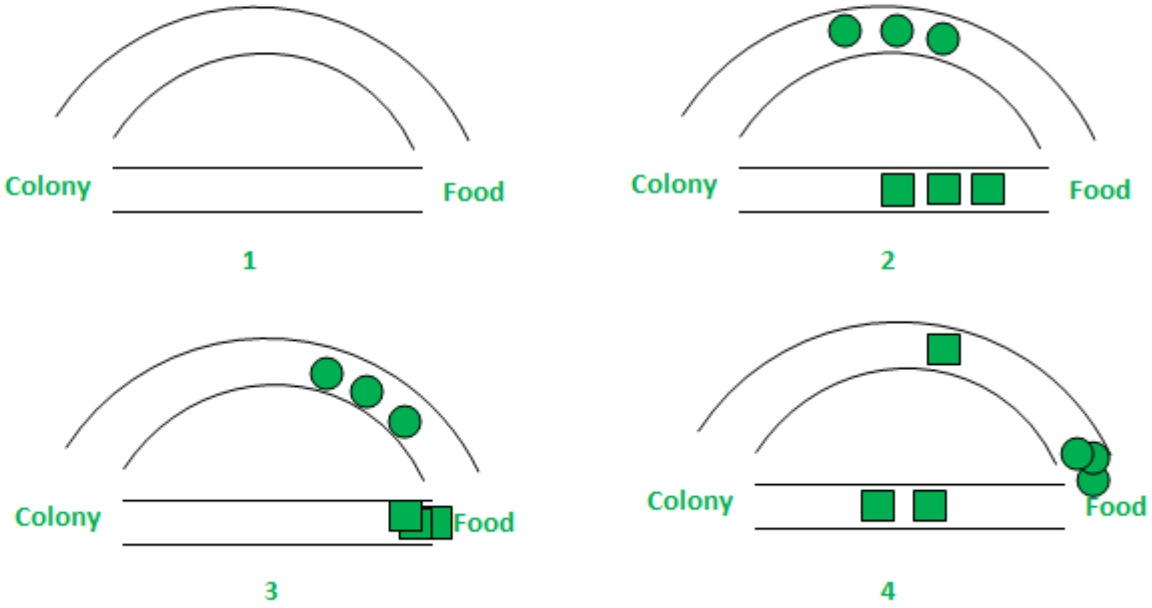
Ant Colony Optimization:

- The algorithmic world is beautiful with multifarious strategies and tools being developed round the clock to render to the need for high-performance computing. In fact, when algorithms are inspired by natural laws, interesting results are observed.
- Evolutionary algorithms belong to such a class of algorithms. These algorithms are designed so as to mimic certain behaviours as well as evolutionary traits of the human genome.
- Moreover, such algorithmic design is not only constrained to humans but can be inspired by the natural behaviour of certain animals as well. The basic aim of fabricating such methodologies is to provide realistic, relevant and yet some low-cost solutions to problems that are hitherto unsolvable by conventional means.
- Different optimization techniques have thus evolved based on such evolutionary algorithms and thereby opened up the domain of metaheuristics. Metaheuristic has been derived from two Greek words, namely, Meta meaning one level above and heuriskein meaning to find.
- Algorithms such as the Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) are examples of swarm intelligence and metaheuristics. The goal of swarm intelligence is to design intelligent multi-agent systems by taking inspiration from the collective behaviour of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or schools of fish.

Background:

- Ant Colony Optimization technique is purely inspired from the foraging behaviour of ant colonies, first introduced by Marco Dorigo in the 1990s.
- Ants are eusocial insects that prefer community survival and sustaining rather than as individual species.
- They communicate with each other using sound, touch and pheromone. Pheromones are organic chemical compounds secreted by the ants that trigger a social response in members of same species.
- These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behaviour of the receiving individuals.

- Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.
- Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path. Initially, ants start to move randomly in search of food around their nests.
- This randomized search opens up multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path.
- Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source. Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone.
- It can also be observed that since the evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.In the above figure, for simplicity, only two possible paths have been considered between the food source and the ant nest.
- The stages can be analyzed as follows:



- Stage 1: All ants are in their nest. There is no pheromone content in the environment. (For algorithmic design, residual pheromone amount can be considered without interfering with the probability)
- Stage 2: Ants begin their search with equal (0.5 each) probability along each path. Clearly, the curved path is the longer and hence the time taken by ants to reach food source is greater than the other.
- Stage 3: The ants through the shorter path reaches food source earlier. Now, evidently they face with a similar selection dilemma, but this time due to pheromone trail along the shorter path already available, probability of selection is higher.

- Stage 4: More ants return via the shorter path and subsequently the pheromone concentrations also increase. Moreover, due to evaporation, the pheromone concentration in the longer path reduces, decreasing the probability of selection of this path in further stages. Therefore, the whole colony gradually uses the shorter path in higher probabilities. So, path optimization is attained.

Conclusion:

Thus, we have successfully implemented a program for Ant colony optimization using Traveling salesman problem.

Assignment No. 10

Title:

Finding the Coolest/Hottest Year using MapReduce

Problem Statement:

Design and develop a distributed application to find the coolest/hottest year from the available weather data. Use weather data from the Internet and process it using MapReduce.

Theory:

MapReduce for Weather Data Analysis:

- MapReduce is a programming model designed for processing large-scale datasets in a distributed computing environment.
- It consists of two main functions:
  - Map Function: Processes input data and generates key-value pairs.
  - Reduce Function: Aggregates the key-value pairs to compute the final result.
- This model enables parallel processing, scalability, and fault tolerance, making it ideal for analysing large weather datasets.

Process of Finding Coolest/Hottest Year using MapReduce:

1. **Data Collection:** Obtain historical weather data from online sources such as NOAA (National Oceanic and Atmospheric Administration).
2. **Preprocessing:** Extract relevant data such as year, temperature readings, and filter out any missing or corrupt values.
3. **Map Phase:**
   - Parse the dataset and extract temperature readings with their corresponding years.
   - Emit key-value pairs in the format (year, temperature).
4. **Shuffle and Sort:** Group temperature records by year.
5. **Reduce Phase:**
   - Compute the minimum and maximum temperature for each year.
   - Identify the coolest and hottest years based on computed values.
6. **Output:** Display the results showing the coolest and hottest years along with their respective temperature values.

Features of MapReduce:

- Parallel Processing: Distributes computations across multiple nodes to improve efficiency.
- Scalability: Capable of handling massive datasets across distributed systems.
- Fault Tolerance: Automatically recovers from failures using replication mechanisms.
- Flexibility: Can be applied to various data processing tasks, including weather data analysis.

Conclusion:

Thus, we have successfully designed and developed a distributed application to determine the coolest and hottest years from weather data using MapReduce.

Loading Data

```
import pandas as pd
import numpy as np
```

```
from google.colab import files
data = files.upload()
```

Choose Files Mumbai_19…ntacruz.csv

- **Mumbai_1990_2022_Santacruz.csv**(text/csv) - 319190 bytes, last modified: 3/31/2025 - 100% done

```
Saving Mumbai_1990_2022_Santacruz.csv to Mumbai_1990_2022_Santacruz (1).csv
```

```
weather_data = pd.read_csv("/content/Mumbai_1990_2022_Santacruz (1).csv")
weather_data.head()
```

	time	tavg	tmin	tmax	prcp
0	01-01-1990	23.2	17.0	NaN	0.0
1	02-01-1990	22.2	16.5	29.9	0.0
2	03-01-1990	21.8	16.3	30.7	0.0
3	04-01-1990	25.4	17.9	31.8	0.0
4	05-01-1990	26.5	19.3	33.7	0.0

Next steps: Generate code with weather_data View recommended plots New interactive sheet

Mapping Function

```
def mapper():
    file_path = "/content/Mumbai_1990_2022_Santacruz (1).csv"
    df = pd.read_csv(file_path, parse_dates=["time"], dayfirst=True)  # Parse date
    df["year"] = df["time"].dt.year  # Extract year
    df = df[["year", "tavg"]].dropna()  # Keep only year and tavg, remove NaNs

    return df
```

Reducing Function

```
def reducer(mapped_data):
    grouped = mapped_data.groupby("year")["tavg"].apply(list)  # Group by year, keep all tavg values
    avg_temps = mapped_data.groupby("year")["tavg"].mean()  # Compute mean tavg per year

    result_df = pd.DataFrame({"tavg_values": grouped, "avg_tavg": avg_temps})
    return result_df
```

Processing using MapReduce

```
if __name__ == "__main__":
    df = weather_data
    print("Weather Data loaded\n")

    mapped_data = mapper()
    print("Mapped Data loaded\n")

    reduced_result = reducer(mapped_data)
    print("Reduced Data loaded\n")
    print(reduced_result.head())

    hottest_year = reduced_result["avg_tavg"].idxmax()
hottest_year = reduced_result["avg_tavg"].idxmax()
coldest_year = reduced_result["avg_tavg"].idxmin()

print(f"Hottest Year: {hottest_year} with avg temp {reduced_result.loc[hottest_year, 'avg_tavg']:.2f}
print(f"Coldest Year: {coldest_year} with avg temp {reduced_result.loc[coldest_year, 'avg_tavg']:.2f}
```

```
Weather Data loaded

Mapped Data loaded

Reduced Data loaded

                                             tavg_values   avg_tavg
year
1990  [23.2, 22.2, 21.8, 25.4, 26.5, 25.1, 26.0, 26....  27.076944
1991  [18.4, 17.9, 18.8, 20.5, 22.2, 22.3, 22.0, 20....  26.933791
1992  [22.8, 24.1, 23.6, 22.2, 23.7, 22.3, 20.3, 21....  27.109836
1993  [24.9, 24.7, 24.1, 24.8, 25.7, 26.3, 23.9, 23....  27.175549
1994  [25.6, 24.7, 23.9, 26.1, 25.1, 25.6, 26.3, 25....  26.939118
Hottest Year: 2018 with avg temp 28.76°C
Coldest Year: 1991 with avg temp 26.93°C
```