Academic Year: 2024-25

LABORATORY MANUAL

Name of the Student:		
Class: BE	**Division:**	**Roll No.:**
Subject: Computer Laboratory III **(2019 Course) [417534]**		**Exam Seat No.:**
Department of Artificial Intelligence and Data Science		

Program Outcomes (PO's):

POs are statements that describe what students are expected to know and be able to do upon graduating from the program. These relate to the skills, knowledge, analytical ability attitude and behavior that students acquire through the program.

- **PO1: Engineering Knowledge:**
  Graduates will be able to apply the Knowledge of the mathematics, science and engineering fundamentals for the solution of engineering problems related to IT.
- **PO2: Problem Analysis:**
  Graduates will be able to carry out identification and formulation of the problem statement by requirement engineering and literature survey.
- **PO3: Design/Development of Solutions:**
  Graduates will be able to design a system, its components and/or processes to meet the required needs with consideration for public safety and social considerations.
- **PO4: Conduct Investigations of Complex Problems:**
  Graduates will be able to investigate the problems, categorize the problem according to their complexity using modern computational concepts and tools.
- **PO5: Modern Tool Usage:**
  Graduates will be able to use the techniques, skills, modern IT engineering tools necessary for engineering practice.
- **PO6: The Engineer and Society:**
  Graduates will be able to apply reasoning and knowledge to assess global and societal issues
- **PO7: Environment and Sustainability:**
  Graduates will be able to recognize the implications of engineering IT solution with respect to society and environment.

- **PO8: Ethics:**

  Graduates will be able to understand the professional and ethical responsibility.

- **PO9: Individual and Team Work:**

  Graduates will be able to function effectively as an individual member, team member or leader in multi -disciplinary teams.

- **PO10: Communication:**

  Graduates will be able to communicate effectively and make effective documentations and presentations.

- **PO11: Project Management and Finance:**

  Graduates will be able to apply and demonstrate engineering and management principles in project management as a member or leader.

- **PO12: Life-long Learning:**

  Graduates will be able to recognize the need for continuous learning and to engage in life- long learning.

Course Objectives and Course Outcomes (COs)

Course Objectives:

- Apply regression, classification and clustering algorithms for creation of ML models
- Introduce and integrate models in the form of advanced ensembles.
- Conceptualized representation of Data objects.
- Create associations between different data objects, and the rules.
- Organized data description, data semantics, and consistency constraints of data

Course Outcomes:

On completion of the course, students will be able to–

CO1: Implement regression, classification and clustering models

CO2: Integrate multiple machine learning algorithms in the form of ensemble learning.

CO3: Apply reinforcement learning and its algorithms for real world applications.

CO4: Analyze the characteristics, requirements of data and select an appropriate data model.

CO5: Apply data analysis and visualization techniques in the field of exploratory data science

CO6: Evaluate time series data.

Dr. D. Y. Patil Educational Federation's

Dr. D. Y. PATIL COLLEGE OF ENGINEERING &INNOVATION

Department of Artificial Intelligence and Data Science

Academic Year 2024-25

CERTIFICATE

This is to certify that Mr. /Ms. ____.

of Class BE - AI-DS, Roll No. Examination Seat No. ____________________________.

has completed all the practical work in the Computer Laboratory - III [417534]

Satisfactorily, as prescribed by Savitribai Phule Pune University, Pune in the academic

year 2024-25 (Term-II).

Place:

Date:

Course In-charge Department of AI-DS	**HOD Department of AI-DS**	**Principal DYPCOEI, Varale**

Index

Department of Artificial Intelligence and data science **Class:** BE

Sr. No.	**Name of the Experiment**	**Date of Conduction**	**Date of Checking**	**Page No.**	**Sign**	**Remark**
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.	Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.					
3.	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.					
4.	Write code to simulate requests coming from clients and distribute them among the servers using the load balancing algorithms.					
5.	Optimization of genetic algorithm parameter in hybrid genetic algorithm-neural network modelling: Application to spray drying of coconut milk.					
6.	Implementation of Clonal selection algorithm using Python.					

7.	To apply the artificial immune pattern recognition to perform a task of structure damage Classification.					
8.	Implement DEAP (Distributed Evolutionary Algorithms) using Python.					
9.	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 Map Reduce.					
10.	Implement Ant colony optimization by solving the Traveling salesman problem using python					

Experiment No: 1

Title: Design a distributed application using RPC for remote

Name of the Student: ___________________________________

Class: BE **Batch:**

Date:

Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

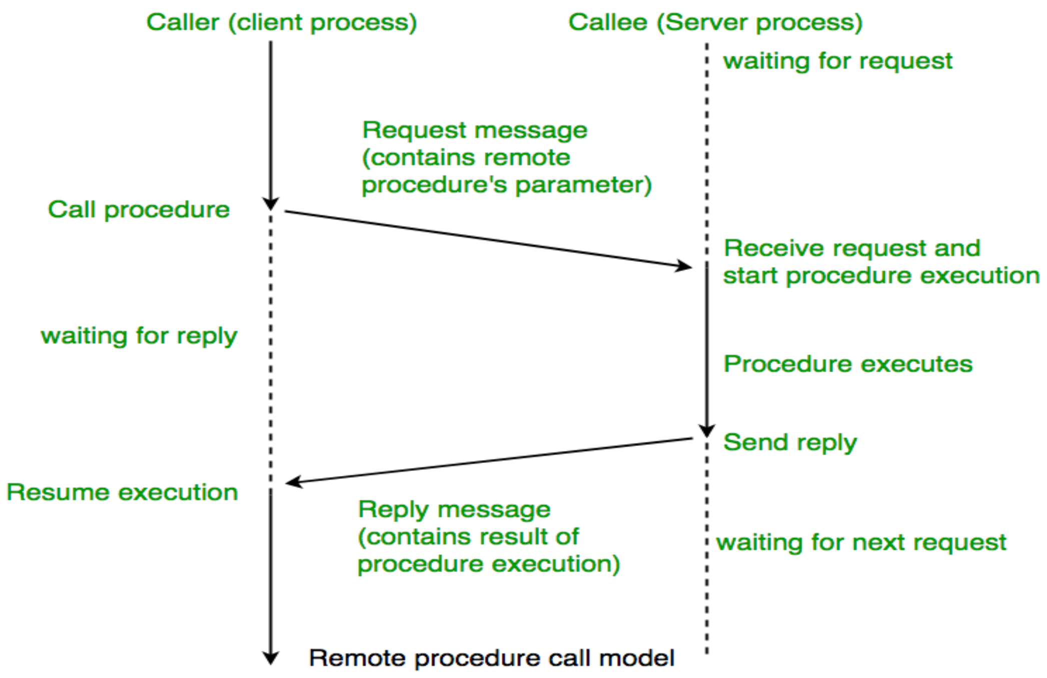
Experiment No: 1

Aim: 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.

- **Outcome:** At end of this experiment, student will be able understand remote Procedure call
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** Java(JDK) & PyCharm IDE

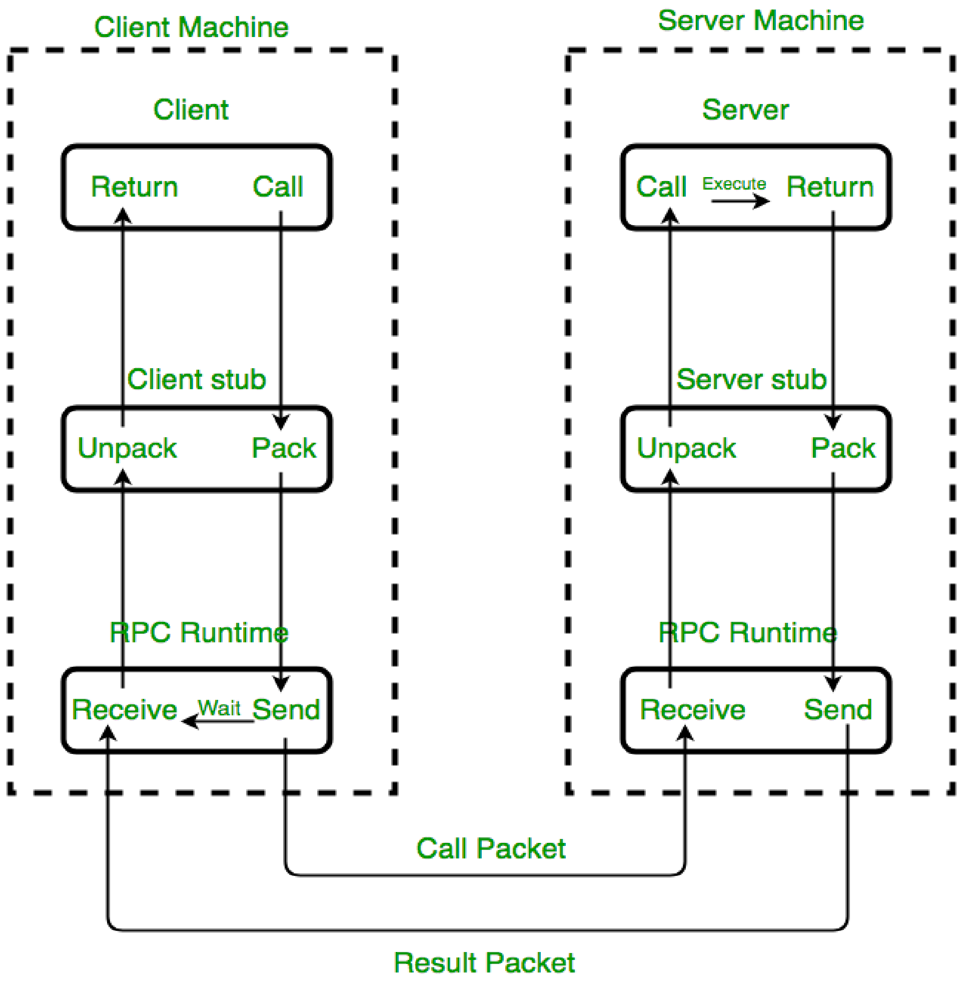
Theory:

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.
When making a Remote Procedure Call:


Caller (client process)
Callee (Server process)
waiting for request
Request message (contains remote procedure's parameter)
Call procedure
Receive request and start procedure execution
waiting for reply
Procedure executes
Send reply
Resume execution
Reply message (contains result of procedure execution)
waiting for next request
Remote procedure call model

1. The calling environment is suspended, procedure parameters are transferred across

the network to the environment where the procedure is to execute, and the procedure is executed there.
2. When the procedure finishes and produces its results, its results are transferred back to the calling environment, where execution resumes as if returning from a regular procedure call.
NOTE: RPC is especially well suited for client-server (e.g. query-response) interaction in which the flow of control alternates between the caller and callee. Conceptually, the client and server do not both execute at the same time. Instead, the thread of execution jumps from the caller to the callee and then back again.



Implementation of RPC mechanism

Working Of RPC:
The following steps take place during a RPC:

- A client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client's own address space.
- The client stub Marshalls(pack) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
- The client stub passes the message to the transport layer, which sends it to the remote server machine.
- On the server, the transport layer passes the message to a server stub, which demur shalls(unpack) the parameters and calls the desired server routine using the regular procedure call mechanism.
- When the server procedure completes, it returns to the server stub (e.g., via a normal procedure call return), which mar shalls the return values into a message. The server stub then hands the message to the transport layer.
- The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
- The client stub demarshalls the return parameters and execution returns to the caller.

Key Considerations for Designing and Implementing RPC Systems are:

- **Security:** Since RPC involves communication over the network, security is a major concern. Measures such as authentication, encryption, and authorization must be implemented to prevent unauthorized access and protect sensitive data.
- **Scalability:** As the number of clients and servers increases, the performance of the RPC system must not degrade. Load balancing techniques and efficient resource utilization are important for scalability.
- **Fault tolerance:** The RPC system should be resilient to network failures, server crashes, and other unexpected events. Measures such as redundancy, failover, and graceful degradation can help ensure fault tolerance.
- **Standardization:** There are several RPC frameworks and protocols available, and it is important to choose a standardized and widely accepted one to ensure interoperability and compatibility across different platforms and programming languages.
- **Performance tuning:** Fine-tuning the RPC system for optimal performance is important. This may involve optimizing the network protocol, minimizing the data transferred over the network, and reducing the latency and overhead associated with RPC calls.

RPC ISSUES :

Issues that must be addressed:

1. RPC Runtime:
RPC run-time system is a library of routines and a set of services that handle the network communications that underlie the RPC mechanism. In the course of an RPC call, client-side and server-side run-time systems' code handle binding, establish communications over an appropriate protocol, pass call data between the client and server, and handle communications errors.
2. Stub:
The function of the stub is to provide transparency to the programmer-written application code. On the client side, the stub handles the interface between the client's local procedure call and the run-time system, marshalling and un marshalling data, invoking the RPC run-time protocol, and if requested, carrying out some of the binding steps. On the server side, the stub provides a similar interface between the run-time system and the local manager procedures that are executed by the server.
3. Binding:
How does the client know who to call, and where the service resides? The most flexible solution is to use dynamic binding and find the server at run time when the RPC is first made. The first time the client stub is invoked, it contacts a name server to determine the transport address at which the server resides.
Binding consists of two parts:
Naming:
Locating:
A Server having a service to offer exports an interface for it. Exporting an interface registers it with the system so that clients can use it.
A Client must import an (exported) interface before communication can begin.
4. The call semantics associated with RPC :
It is mainly classified into following choices-
Retry request message –
Whether to retry sending a request message when a server has failed or the receiver didn't receive the message.
Duplicate filtering –
Remove the duplicate server requests.
Retransmission of results –
To resend lost messages without re-executing the operations at the server side.
ADVANTAGES:
RPC provides ABSTRACTION i.e. message-passing nature of network communication is hidden from the user.
RPC often omits many of the protocol layers to improve performance. Even a small performance improvement is important because a program may invoke RPCs often.

RPC enables the usage of the applications in the distributed environment, not only in the local environment. With RPC code re-writing / re-developing effort is minimized. Process-oriented and thread oriented models supported by RPC.

Conclusion:

In this distributed application, we utilized **Remote Procedure Call (RPC)** to implement a client-server architecture where the client submits an integer value to the server, and the server computes its factorial before returning the result. This design demonstrates the efficiency of distributed computing by offloading computational tasks to a remote server while maintaining a simple and seamless interaction for the client.

Questions:

Q1. Explain RPC w.r.t Distributed?

Q2. Explain RPC Implementation mechanism?

Q3. Define Marshalls & De Marshalls in terms of RPC?

Q4. What Are the Issues with RPC?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 2

Title: Design a distributed application using RMI for remote

Name of the Student: ____________________________________

Class: BE **Batch:**

Date:	Mark: /10

Signature of the Course In-charge: ___________________

Signature of the HOD: _____________________________

Experiment No: 2

Aim: Design a distributed application using RMI for remote computation where client submits two strings to the server and server returns the concatenation of the given strings.

- **Outcome:** At end of this experiment, student will be able understand remote Procedure call
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** Java(JDK) & PyCharm IDE

Theory:

RMI (Remote Method Invocation):

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

Understanding stub and skeleton RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),

3. It waits for the result
4. It reads (un marshals) the return value or exception, and
5. It finally, returns the value to the caller.

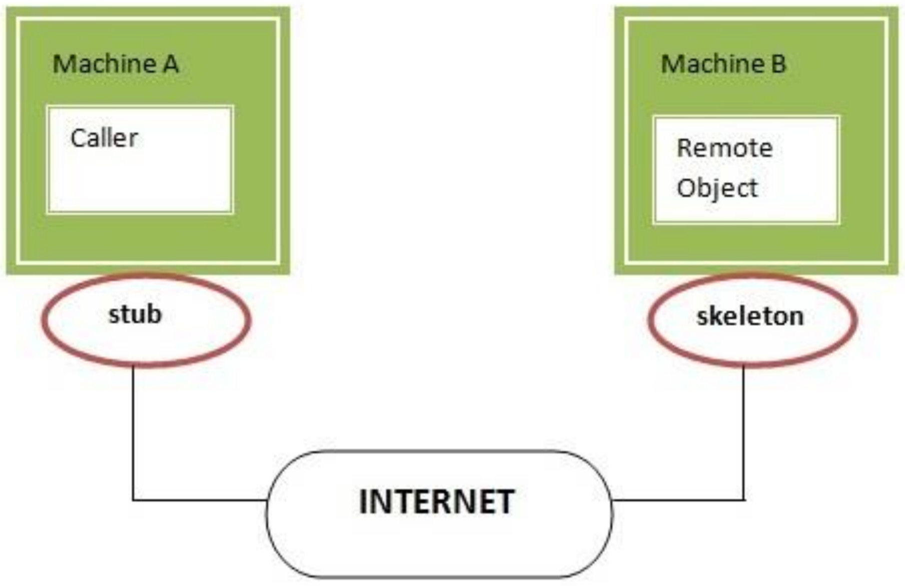
Skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

It reads the parameter for the remote method

1. It invokes the method on the actual remote object, and
2. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons.



Understanding requirements for the distributed applications

If any application performs these tasks, it can be distributed application.

The application need to locate the remote method

1. It need to provide the communication with the remote objects, and
2. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

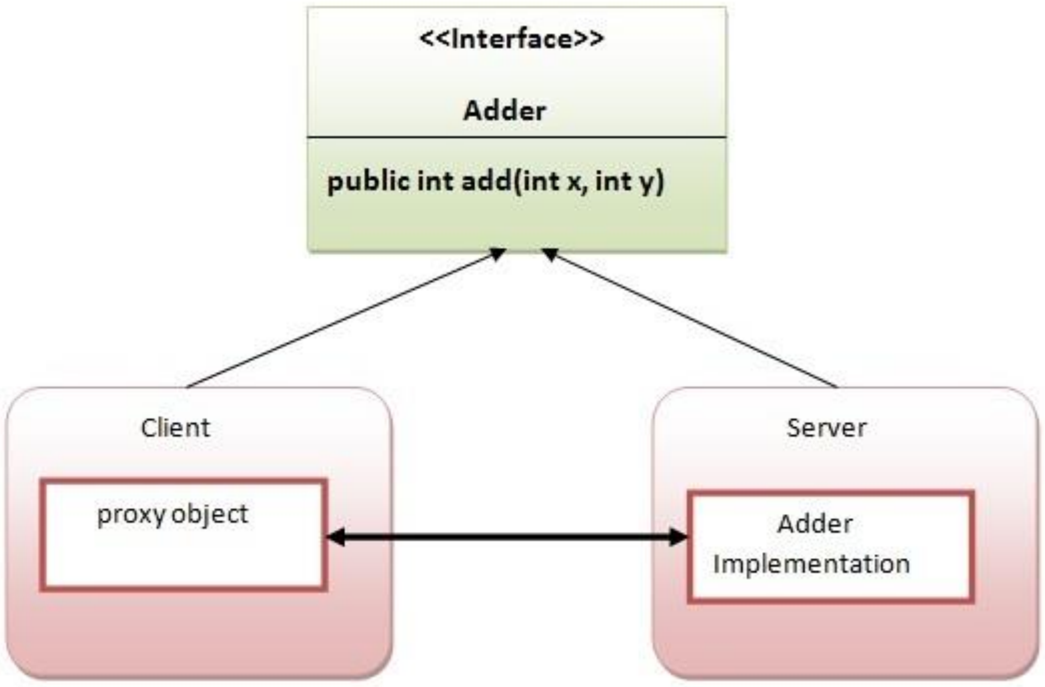
Java RMI Example

The is given the 6 steps to write the RMI program.

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application

RMI Example

In this example, we have followed all the 6 steps to create and run the RMI application. The client application need only two files, remote interface and client application. In the RMI application, both client and server interacts with the remote interface. The client application invokes methods on the proxy object, RMI sends the request to the remote JVM. The return value is sent back to the proxy object and then to the client application.



Conclusion:

In this distributed application, we implemented **Java Remote Method Invocation (RMI)** to facilitate remote computation between a client and a server. The client sends two strings to the server, and the server processes the request by concatenating the strings and returning the result

Questions:

Q1. What RMI? How its working?

Q2. Describe JAVA RMI as an Example?

Q3. What is role of stub & skeleton in RMI?

Q4. Difference between RPC & RMI?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 3

Title: Implement Union, Intersection, Complement and Difference operations on fuzzy sets.

Name of the Student: __

Class: BE **Batch:**

Date:

Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

Experiment No: 3

Aim: 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.

- **Outcome:** At end of this experiment, student will be able understand FUZZY w.r.t SET
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** PyCharm IDE

Theory:

Logic is a tool for reasoning propositions that can be manipulated with mathematical precepts. Fuzzy Logic: The word fuzzy means uncertainty. Any particular event which do not result any of the exact value (i.e. true or false) is fuzzy. A proposition is a declarative or linguistic statement within a universe of discourse.

Shows a real-world situation where the glass is more than half full of water.

Fuzzy logic is a transition from absolute truth to partial truth. That is, from a variable x (True or False) to a linguistic variable ‘Almost full’, ‘very close to empty’, etc. From this perspective, fuzzy logic can be seen as a reasoning formalism of humans where all truthsare partial or approximate and any falseness is represented by partial truth.

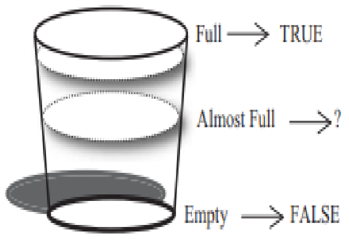
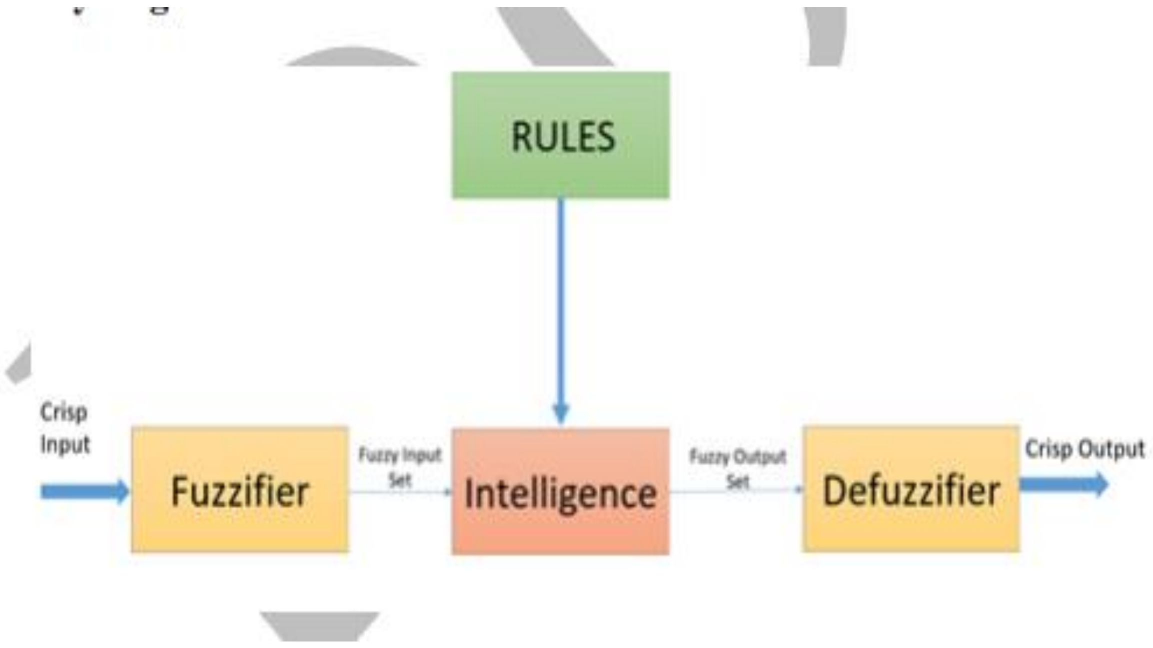


Figure 2.1 A real-world situation



Rule Base: It contains all the rules and the if-then conditions offered by the experts to control the decision-making system.

Inference Engine: It helps you to determine the degree of match between fuzzy input and the rules. Based on the % match, it determines which rules need implement according to the given input field. After this, the applied rules are combined to develop the control actions.

Fuzzification step helps to convert inputs. It allows you to convert, crisp numbers into fuzzy sets. Crisp inputs measured by sensors and passed into the control system for further processing. Like Room temperature, pressure, etc.

Defuzzification: At last the Defuzzification process is performed to convert the fuzzy sets into a crisp value. There are many types of techniques available, so you need to select it which is best suited when it is used with an expert system.

For example, a fuzzy set A = {x1, x2, x3, x4} in X is characterized by the membership function μA(x) which maps each point x in X to real values 0.5, 1, 0.75 and 0.5. μA(x) represents the degree of membership of x in A and the mapping is only limited by μA(x) ∈ [0, 1]. In classical set theory, the membership function can take only two values: 0 and 1, i.e., either a(x) =1 or a(x) = 0. In set-theoretic notation this is written as μA(x) ∈ {0, 1}.

$$A = \{x, \mu A(x) \mid x \in X\}$$

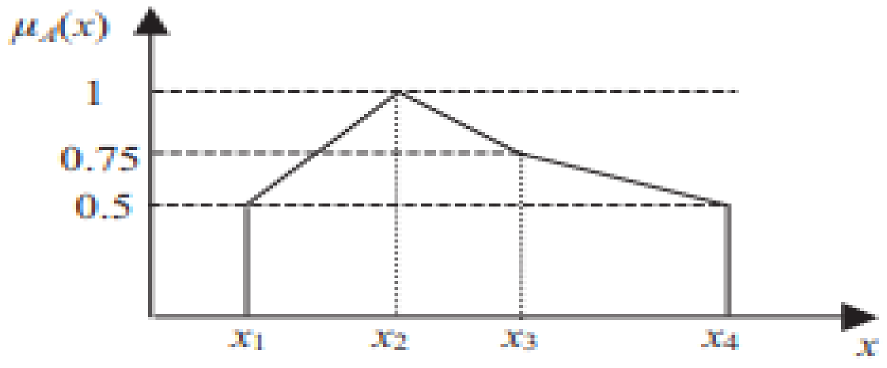


Figure 2.2 Fuzzy set

This mapping can be depicted pictorially, as shown in Figure 2.2**. In Figure 2.2,** x1, x2, x3 and x4 have membership grades of 0.5, 1, 0.75 and 0.5, respectively, written as μA (x1) = 0.5, μA (x2) = 1, μA (x3) = 0.75 and μA (x4) = 0.5. A notational **convention of fuzzy sets** for a discrete and finite universe of discourse X in practice is written as A = {μA(x1)/x1 + μA(x2)/x2 +···+ μA(xn)/xn} = n i=1 μA(xi)/xi

Fuzzy Rules:

$$\boxed{R: \quad \text{If } x \text{ is } A \;,\; \text{then } y \text{ is } B.}$$

where A and B are linguistic values defined by fuzzy sets on universes of discourse X and Y, respectively.

The rule is also called a "fuzzy implication" or fuzzy conditional statement. The part "x is A" is called the "antecedent", while "y is B" is called the "consequence" or "conclusion".

Before we employ fuzzy if-then rules to model and analyses a system, first we have to formalize what is meant by the expression: R: If x is A then y is B which is sometimes abbreviated as

$$\boxed{R: A \rightarrow B}$$

Conclusion:

Fuzzy set operations like union, intersection, difference, and complement help model uncertainty in data. The Cartesian product extends these concepts to fuzzy relations, which can be further analyzed using the max-min composition.

Questions:

Q1. What FUZZY? Explain with Ex.?

Q2. Fuzzy Application?

Q3. Explain Fuzzy Architecture?

Q4. Differentiate Classical Vs Fuzzy Rule?

Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Course In-charge
5	3	2	10	

Experiment No: 4

Title: Write code to simulate requests coming from clients

Name of the Student: __

Class: BE **Batch:**

Date:	Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

Experiment No: 4

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

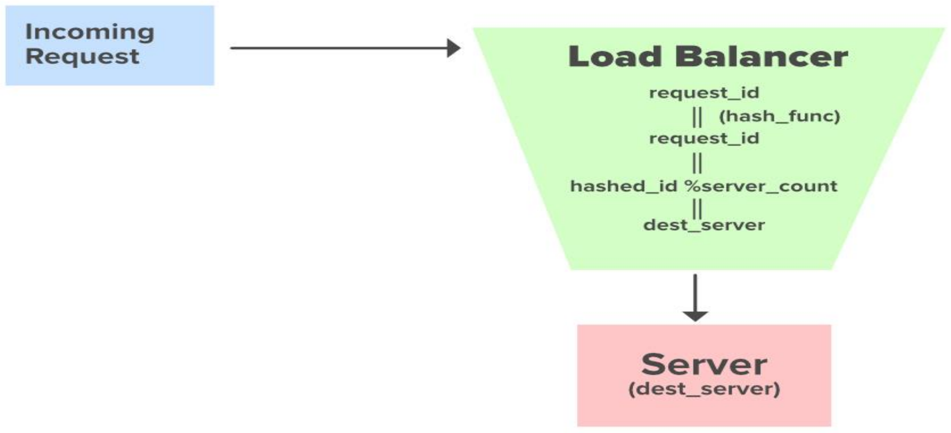
- **Outcome:** At end of this experiment, student will be able understand Load Balancing
- **Hardware Requirement:** Computer System, Linux (Ubantu)
- **Software Requirement:** PyCharm IDE

Theory:

What are Load Balancers?

In case multiple servers are present the incoming request coming to the system needs to be directed to one of the multiple servers. We should ensure that every server gets an equal amount of requests. The requests must be distributed in a uniform manner across all the servers. The component which is responsible for distributing these incoming requests uniformly across the servers is known as **Load Balancer**. A Load Balancer acts as a layer between the incoming requests coming from the user and multiple servers present in the system.

We should avoid the scenarios where a single server is getting most of the requests while the rest of them are sitting idle. There are various Load Balancing Algorithms that ensure even distribution of requests across the servers.



Hashing Approach to direct requests from the Load Balancer

We will be discussing the Hashing Approach to direct the requests to multiple servers uniformly. Suppose we have **server_count** as the Total number of servers present in the System and a **load_balancer** to distribute the requests among those servers. A request with an id **request_id** enters the system. Before reaching the destination server it is directed to the **load_balancer** from where it is further directed to its destination server. When the request reaches the load balancer the hashing approach will provide us with the destination server where the request is to be directed.

Discussing the Approach:

- **request_id** : Request ID coming to get served
- **hash_func** : Evenly distributed Hash Function
- **hashed_id** : Hashed Request ID
- **server_count** : Number of Servers

Computing the Destination Server address:

If the value of **server_count** is 10 i.e we have ten servers with following server ids **server_id_0, server_id_1, ………, server_id_9**. Suppose the value of **request_id** is 23 when this request reaches the Load Balancer the hash function **hash_func** hashes the value of the incoming request id.

- **hash_func(request_d)** = **hash_func(23)**

Suppose after Hashing the request_id is randomly hashed to a particular value.

- **hashed_id** = 112

In order to bring the hashed id in the range of the number of servers we can perform a modulo of the hashed id with the count of servers.

- **dest_server** = **hashed_id** % **server_count**
- **dest_server** = 112%10
- **dest_server** = 2

So we can route this request to Server **server_id_2** in this way we can distribute all the requests coming to our Load Balancer evenly to all the servers. But is it an optimal approach? Yes it distributes the requests evenly but what if we need to increase the number of our servers. Increasing the server will change the destination servers of all the incoming requests. What if we were storing the cache related to that request in its destination server? Now as that request is no longer routed to the earlier server, our entire cache can go in trash probably. Think!

Load balancing is a technique used to distribute incoming requests evenly across multiple servers in a network, with the aim of improving the performance, capacity, and reliability of the system. Load balancers act as a reverse proxy, routing incoming requests to different servers based on various algorithms and criteria.

Here's how routing requests through a load balancer works:

Incoming requests are received by the load balancer, which acts as a single point of entry for all incoming traffic. The load balancer uses an algorithm to determine which server should handle the request, based on factors such as the server's current load, response time, and availability. The load balancer forwards the request to the selected server. The server processes the request and returns the response to the load balancer. The load balancer returns the response to the client. By routing requests through a load balancer, the system can improve its performance and capacity, as the load balancer ensures that incoming requests are distributed evenly across all available servers. This helps to avoid overloading any individual server and ensures that the system can continue to handle incoming requests even if one or more servers fail. In a distributed system architecture, routing requests through a load balancer is a common technique to improve performance, scalability, and reliability. A load balancer acts as a traffic cop, distributing incoming requests across multiple servers to balance the load and prevent any one server from becoming overloaded.

Here are some key points to understand about routing requests through a load balancer:

1. **How it works:** When a client sends a request to the system, it is first received by the load balancer. The load balancer then distributes the request to one of several servers in the system based on a predefined algorithm, such as round-robin, least connections, or IP hash. The server processes the request and sends the response back to the client through the load balancer.
2. **Load balancing algorithms:** Load balancing algorithms are used by the load balancer to distribute requests across servers. The choice of algorithm can affect the performance and reliability of the system. For example, round-robin is a simple algorithm that evenly distributes requests across servers, while least connections distributes requests to the server with the fewest active connections.
3. **Scaling:** Routing requests through a load balancer can help scale a system by allowing additional servers to be added to handle increased traffic. When a new server is added, the load balancer can automatically distribute requests to it.

4. **High availability:** Routing requests through a load balancer can also improve system reliability by providing redundancy. If one server fails, the load balancer can automatically redirect requests to another server.

The advantages of routing requests through a load balancer include:

1. Improved performance: By balancing the load across multiple servers, a load balancer can improve the response time of a system and prevent any one server from becoming overloaded.
2. Increased scalability: A load balancer can help a system scale by allowing additional servers to be added to handle increased traffic.
3. Improved reliability: A load balancer can improve the reliability of a system by providing redundancy and automatically redirecting requests to healthy servers if one server fails.
4. Simplified management: A load balancer can simplify the management of a distributed system by allowing administrators to configure and manage multiple servers through a single interface.

However, there are also some potential disadvantages to routing requests through a load balancer, including:

1. Cost: A load balancer can be expensive to implement and maintain, particularly for smaller systems.
2. Complexity: A load balancer can add additional complexity to a system, particularly when configuring and managing multiple servers.
3. Single point of failure: A load balancer can be a single point of failure for a system. If the load balancer fails, the entire system may become unavailable.
4. Overall, routing requests through a load balancer is a powerful technique for improving the performance, scalability, and reliability of a distributed system. By understanding the key principles and potential advantages and disadvantages, developers can make informed decisions about when and how to use load balancing in their systems.

Conclusion:

Load balancing algorithms efficiently distribute incoming requests among multiple servers to ensure optimal resource utilization and reduce server overload. Round Robin ensures equal distribution, Least Connections minimizes congestion, and Random Selection provides unpredictability. Choosing the right algorithm depends on workload characteristics and system requirements for optimal performance.

Questions:

Q1. Why there is need of Load Balancing computing?

Q2. Explain the role of Load Balancer? How its work explain?

Q3. State the Advantage & Disadvantages of Load Balancer?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 5

Title: Optimization of genetic algorithm parameter in hybrid genetic algorithm

Name of the Student: __

Class: BE **Batch:**

Date:	Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

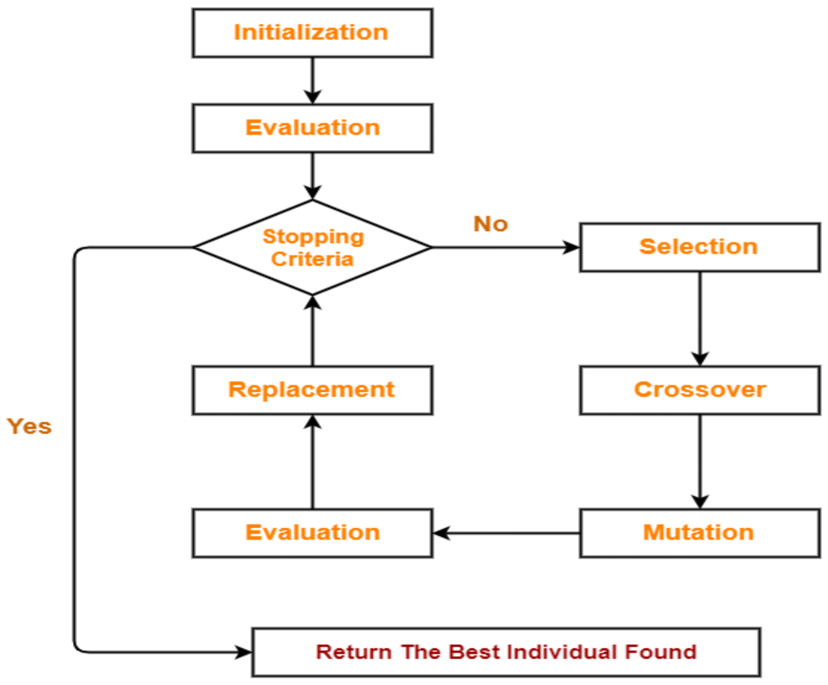
Experiment No: 5

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

- **Outcome:** At end of this experiment, student will be able understand Genetic Algorithm
- **Hardware Requirement:** Computer System, Linux(Ubantu)
- **Software Requirement:** PyCharm IDE

Theory:

Genetic Algorithm:



How Genetic Algorithm Works

❖ Abstraction of real biological evolution

- ❖ Solve complex Problems(NP-Hard type ex.TSP)
- ❖ Focus On Optimaization
- ❖ Population of possible solution for a given problem
- ❖ From group of individuals the best one Will survive

Genetic Algorithms offer the following advantages-

<u>Point-01:</u>

Genetic Algorithms are better than conventional AI.

- This is because they are more robust.

<u>Point-02:</u>

They do not break easily unlike older AI systems.

- They do not break easily even in the presence of reasonable noise or if the inputs get change slightly.

<u>Point-03:</u>

While performing search in multi modal state-space or large state-space,

Genetic algorithms has significant benefits over other typical search optimization techniques.

- ▪ GA (Genetic Algorithm) is good at taking larger, potentially huge search space and navigating them looking for optimal solution which we might not find in lifetime.
- ▪ GA is better than other traditional algorithm in that they are more robust.
- ▪ They do not break easily even if the inputs are changed slightly or in the presence of reasonable noise.

GA is used to resolve complicated optimization problems, such as , organizing the time table, scheduling job shop, playing games.

The concept of GA is directly derived from natural evolution and heredity i.e. inheritance, where child inherits the characters (stored in the chromosomes) from the parent.

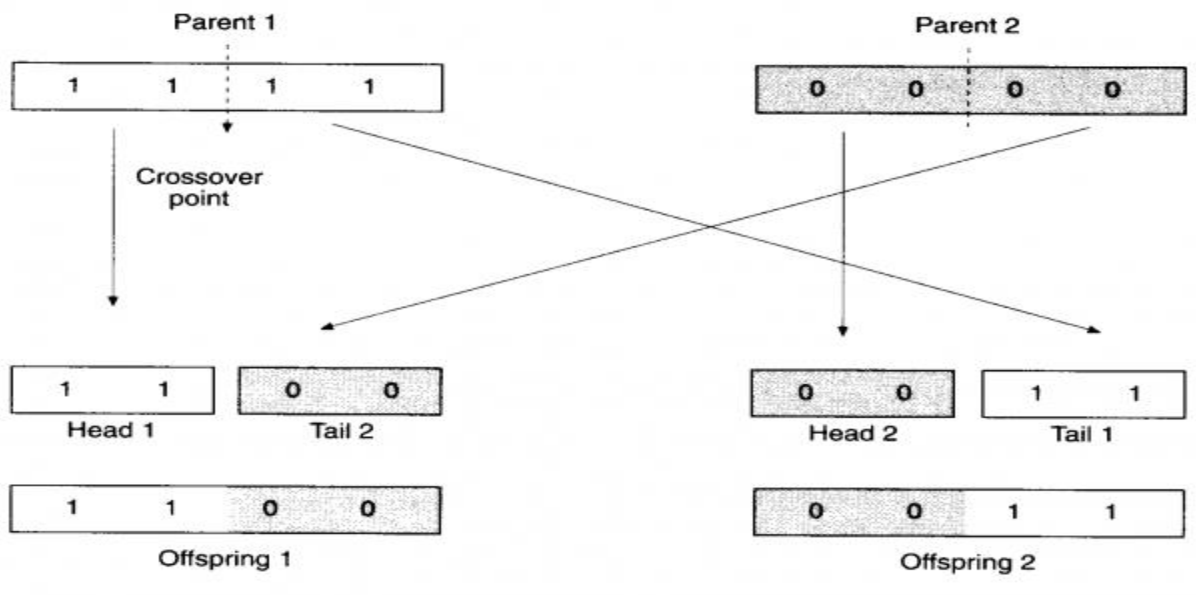
Operators in GA:

1.Crossover (Recombination):-

Crossover is the process of **taking two parent solutions and producing from them a child**. After the selection (reproduction) process, the population is enriched with better individuals. Crossover operator is applied to the mating pool with the hope that it creates a better offspring.

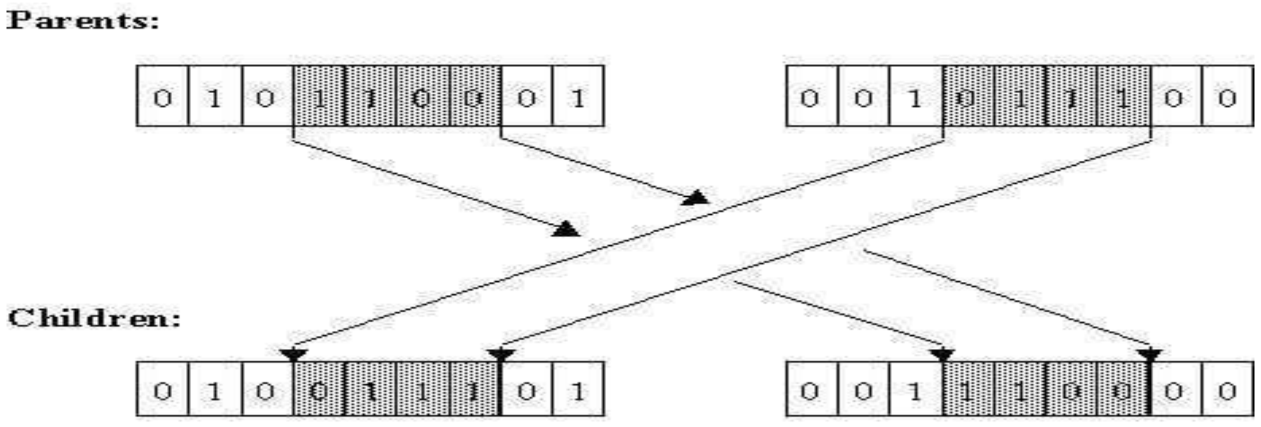
The various crossover techniques are-

i).**Single-Point Crossover**-Here the two mating chromosomes are cut once at corresponding points and the sections after the cuts exchanged.



ii). Two-Point Crossover-

Here two crossover points are chosen and the contents between these points are exchanged between two mated parents.

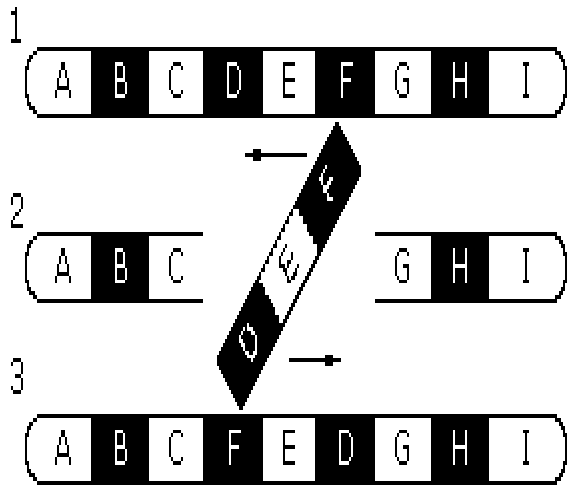


2. Inversion:-

Inversion operator inverts the bits between two random sites.

01 0011 1

Then, 0111001



3. Deletion:-

i).Deletion and duplication-Here any two or three bits in random are selected and their previous bits are duplicated.

before duplication: 00   1001 0

deletion: 00 10_ _   0

duplication: 00 1010 0

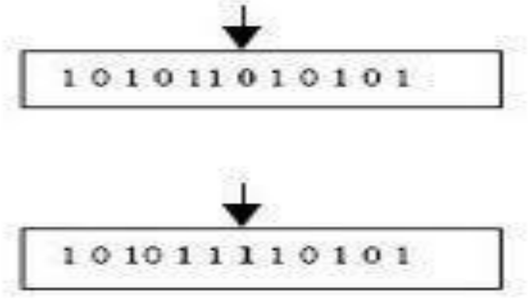
ii). Deletion and regeneration-Here bits between the cross site are deleted and regenerated randomly.

10 0110 1

10 1101 1

4. Mutation:-

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. It plays the role of recovering the genetic materials as well as for randomly distributing genetic information. It helps escape from local minima's trap and maintain diversity in the population. Mutation of a bit involves flipping a bit, changing 0 to 1and vice-versa.



Conclusion:

The optimization of genetic algorithm (GA) parameters in a hybrid GA-neural network model enhances the predictive accuracy and efficiency of spray drying processes for coconut milk. Fine-tuning parameters like mutation rate, crossover probability, and population size ensures better convergence and model performance.

Questions:

Q1. Define Phenotype & Genotype with Ex.?

Q2. Define Encoding & Decoding And Explain Its Technique?

Q3. Define Term Population,Genes,Fitness Function w.r.t Genetic Algorithm?

Q4. Explain Genetic algorithm with its architecture?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 6

Title: Implementation of Clonal selection algorithm using Python.

Name of the Student: ______________________________

Class: BE **Batch:**

Date:

Mark: /10

Signature of the Course In-charge: ___________________

Signature of the HOD: _____________________________

Experiment No: 6

Aim: To implement the Clonal Selection Algorithm (CSA) to solve classification problems, demonstrating its application in machine learning.

- **Outcome:** The algorithm will classify input data effectively by simulating the immune response and applying it to classification problems. The output will include a trained classifier that can predict the class of new unseen data. The Clonal Selection Algorithm will iteratively optimize the parameters and improve classification accuracy by selecting the best clones of the best-performing antibodies.
- **Hardware Requirement:** Computer System, Linux(Ubuntu)
- **Software Requirement:** Python 3.x, Libraries: **NumPy**: For numerical operations, **Scikit-learn**: For classification and datasets, **Matplotlib** (optional): For visualization of results, **SciPy** (optional): For optimization tasks.

Theory:

The Clonal Selection Algorithm is based on the behavior of the immune system. In an immune system, when a foreign antigen (pathogen) is introduced, immune cells (antibodies) produce clones that are similar to the original antibody but differ slightly in their ability to recognize the pathogen. The best-performing antibodies are cloned and undergo hypermutation to improve their affinity toward the antigen.

In the Clonal Selection Algorithm, this idea is applied to solving optimization or classification problems by:

- **Selection**: Choosing the best solutions (antibodies) based on their fitness.
- **Cloning**: Generating clones of the best solutions.
- **Mutation**: Mutating the clones to improve performance.
- **Replacement**: Replacing the worst-performing solutions with the best clones.

Steps in Clonal Selection Algorithm:

1. **Initialization**: Generate a random population of antibodies (potential solutions).
2. **Evaluation**: Calculate the fitness of each antibody (solution).
3. **Selection**: Choose the best-performing antibodies based on fitness values.

4. **Cloning**: Create multiple clones of the selected antibodies.
5. **Mutation**: Introduce mutations in the clones to explore new regions in the solution space.
6. **Replacement**: Replace the worst antibodies with the best clones and their mutated versions.
7. **Termination**: Repeat the process until the desired number of generations or performance is achieved.

Classification:

After running the Clonal Selection Algorithm, you should get an accuracy measure on the test dataset, showing the effectiveness of the algorithm in solving the classification problem.

Conclusion:

Hence we conclude the Clonal Selection Algorithm is a robust and novel approach to optimization and classification. By mimicking the immune system's natural selection process, the algorithm can efficiently identify the optimal solutions to classification problems. This approach can be applied to many fields, including pattern recognition and machine learning, with successful results.

Questions:

1. What is the primary principle behind the Clonal Selection Algorithm?
2. How is the concept of "cloning" used in this algorithm, and what role does mutation play?
3. Can the Clonal Selection Algorithm be used for other problems apart from classification? If yes, explain how.
4. What are the advantages of using CSA over other traditional algorithms like genetic algorithms?
5. How does the fitness function impact the performance of the Clonal Selection Algorithm in classification tasks?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 7

Title: To apply the artificial immune pattern recognition to perform a task.

Name of the Student: __

Class: BE **Batch:**

Date:	Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

Experiment No: 7

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

- **Outcome:** At end of this experiment, student will be able understand AIPR
- **Hardware Requirement:** Computer System, Linux(Ubuntu)
- **Software Requirement:** PyCharm IDE OR, Jupyter

Theory:
The task at hand is to classify structural damage in various infrastructure components such as buildings, bridges, roads, etc. Given data representing different types and degrees of structural damage, the goal is to develop a classification system that can accurately identify and classify the severity of damage.

Approach using Artificial Immune Pattern Recognition (AIPR):

1. Understanding AIPR:
- AIPR is inspired by the human immune system's ability to recognize and respond to foreign pathogens.
- In AIPR, the concept of antigens and antibodies is used to detect patterns in data.
- Antigens represent patterns in the data, while antibodies are generated to recognize and classify these patterns.
- AIPR algorithms evolve a set of antibodies through a process of affinity maturation and selection to achieve accurate pattern recognition.
- 2. Data Preprocessing:
- Begin by collecting and preprocessing the data representing structural damage.
- Preprocessing may involve tasks such as normalization, feature extraction, and dimensionality reduction.

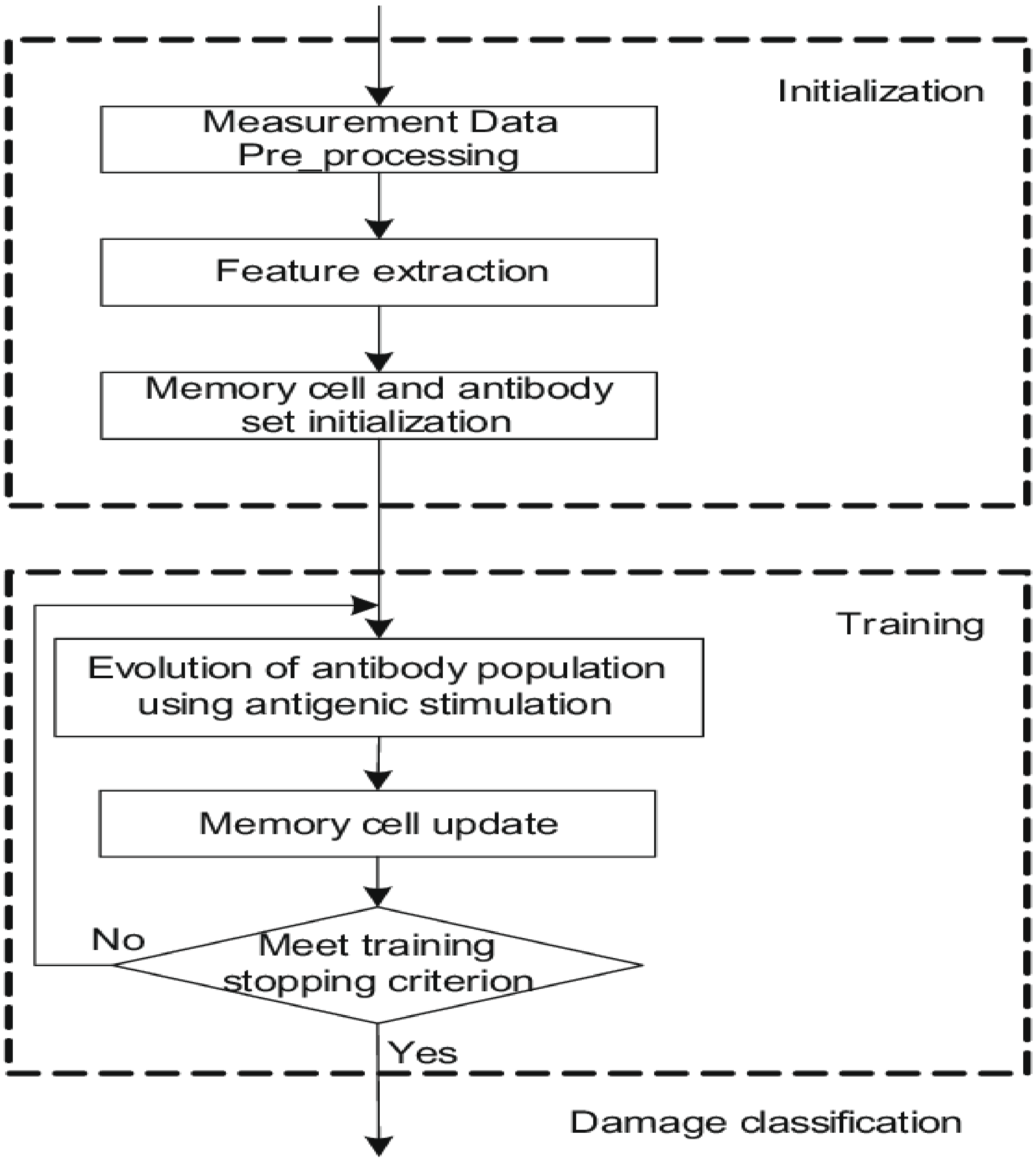
3. Representation of Data:
- Represent the structural damage data as antigens, where each antigen represents a specific pattern or feature in the data.
- Antigens can be represented as vectors or matrices depending on the nature of the data.

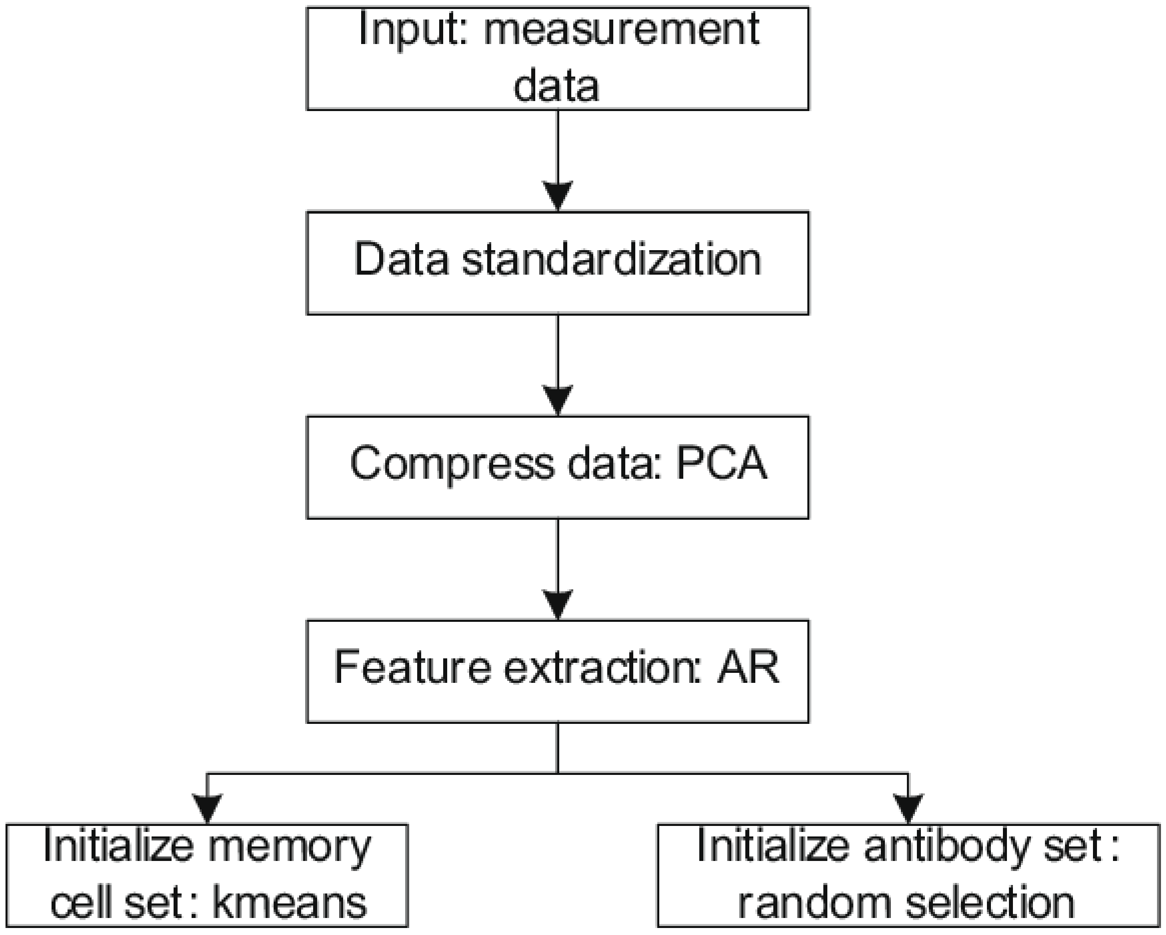
4. Generation of Antibodies:

- Initialize a set of antibodies representing potential classifiers.
- Antibodies are initialized randomly or using a predefined strategy.

5. Affinity Maturation:
   - Apply affinity maturation to evolve the antibodies to better recognize antigens.
   - During affinity maturation, antibodies undergo mutation and selection to improve their affinity towards antigens.

6. Training:
   - Train the AIPR system using the preprocessed data.
   - During training, the antibodies are exposed to antigens, and their affinities are adjusted based on the similarity between antibodies and antigens.
7. Classification:
   - After training, the evolved antibodies can be used for classification.
   - Given a new instance of structural damage, the system can classify its severity based on the affinity of antibodies towards the patterns present in the instance.
8. Evaluation:
   - Evaluate the performance of the AIPR system using metrics such as accuracy, precision, recall, and F1-score.
   - Fine-tune the system parameters and repeat the training and evaluation process as necessary to improve performance.


Initialization
Measurement Data Pre_processing
Feature extraction
Memory cell and antibody set initialization
Training
Evolution of antibody population using antigenic stimulation
Memory cell update
No
Meet training stopping criterion
Yes
Damage classification



[illegible]

Data preprocessing in Artificial Immune Pattern Recognition (AIPR) plays a crucial role in preparing the raw data for effective pattern recognition. Here's how data preprocessing can be conducted in the context of AIPR for the task of structure damage classification:

1. **Data Cleaning:**
    - Remove any noise or irrelevant information from the dataset.
    - Handle missing values by imputation or removal.
2. **Normalization:**
    - Scale the features to a similar range to ensure that they contribute equally to pattern recognition.
    - Common normalization techniques include Min-Max scaling and Z-score normalization.

3. **Feature Extraction:**
    - Extract relevant features from the raw data that capture important characteristics related to structural damage.
    - Feature extraction techniques can include statistical measures, frequency domain analysis, or domain-specific features.
4. **Dimensionality Reduction:**
    - Reduce the dimensionality of the feature space to mitigate the curse of dimensionality and improve computational efficiency.
    - Techniques such as Principal Component Analysis (PCA) or feature selection methods can be employed for dimensionality reduction.
5. **Encoding Categorical Variables:**
    - If the dataset contains categorical variables, encode them into numerical values using techniques such as one-hot encoding or label encoding.
6. **Data Balancing (if applicable):**
    - If the dataset is imbalanced, where certain classes of structural damage are underrepresented, apply techniques such as oversampling, undersampling, or synthetic data generation to balance the dataset.
7. **Data Partitioning:**
    - Split the preprocessed dataset into training, validation, and test sets for model development, evaluation, and testing purposes, respectively.
8. **Data Augmentation (optional):**
    - Augment the dataset by generating additional samples through transformations such as rotation, translation, or scaling.
    - Data augmentation can help improve model generalization and robustness, especially when dealing with limited data.
9. **Data Representation:**
    - Represent the preprocessed data in a suitable format for AIPR algorithms to process.
    - Data representation may involve converting the data into antigenic form, where each instance is represented as an antigen with specific features.

Data Representaion: Antigen Representation:

- In AIPR, antigens represent patterns or instances from the dataset.
- Each antigen encapsulates the features or attributes of a data point.
- Antigens can be represented as vectors, matrices, or any other suitable data structure depending on the nature of the data.
- For structure damage classification, antigens may include various features such as material properties, geometric characteristics, environmental conditions, etc.

Classification:

- Given a new antigen (representing a new instance of structural damage), classify it by evaluating its similarity to the antibodies.
- Measure the similarity between the antigen and each antibody using a predefined similarity metric (e.g., Euclidean distance or cosine similarity).
- Classify the antigen based on the classification of the antibody with the highest affinity.
- The classification result may include the predicted class label and the confidence level (affinity) associated with the classification.

Conclusion:

Artificial immune pattern recognition effectively classifies structural damage by mimicking the human immune system's ability to detect anomalies. This approach enhances damage detection accuracy by recognizing patterns of structural defects based on learned immune responses. It is a robust and adaptive method for real-time monitoring, ensuring the safety and longevity of engineering structures.

Questions:

Q1. Describe AIS?

Q2. Explain Data Representation in AIS?

Q3. Define Data Cleaning & Feature Extraction?

Q4. Why there is need of AIS?

Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Course In-charge
5	3	2	10	

Experiment No: 8

Title: Implement DEAP (Distributed Evolutionary Algorithms) using Python.

Name of the Student: __

Class: BE **Batch:**

Date:

Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

Experiment No: 8

Aim: Implement DEAP (Distributed Evolutionary Algorithms) using Python

- **Outcome:** At end of this experiment, student will be able understand DEAP
- **Hardware Requirement:** Computer System, Linux(Ubuntu)
- **Software Requirement:** PyCharm IDE

Theory:

- DEAP is a novel evolutionary computation framework for rapid prototyping and testing of ideas. It seeks to make algorithms explicit and data structures transparent. It works in perfect harmony with parallelization mechanisms such as multiprocessing and SCOOP.

DEAP includes the following features:

- Genetic algorithm using any imaginable representation
  - List, Array, Set, Dictionary, Tree, Numpy Array, etc.
- Genetic programming using prefix trees
  - Loosely typed, Strongly typed
  - Automatically defined functions
- Evolution strategies (including CMA-ES)
- Multi-objective optimisation (NSGA-II, NSGA-III, SPEA2, MO-CMA-ES)
- Co-evolution (cooperative and competitive) of multiple populations
- Parallelization of the evaluations (and more)
- Hall of Fame of the best individuals that lived in the population
- Checkpoints that take snapshots of a system regularly
- Benchmarks module containing most common test functions
- Genealogy of an evolution (that is compatible with NetworkX)
- Examples of alternative algorithms : Particle Swarm Optimization, Differential Evolution, Estimation of Distribution Algorithm

Overview

If you are used to any other evolutionary algorithm framework, you’ll notice we do things differently with DEAP. Instead of limiting you with predefined types, we provide ways of creating the appropriate ones. Instead of providing closed initializers, we enable you to customize them as you wish. Instead of suggesting unfit operators, we explicitly ask you to choose them wisely. Instead of implementing many sealed algorithms, we allow you to write the ones that fit all your needs. This tutorial will present a quick overview of what DEAP is all about along with what every DEAP program is made of.

Types

The first thing to do is to think of the appropriate type for your problem. Then, instead of looking in the list of available types, DEAP enables you to build your own. This is done with the creator module. Creating an appropriate type might seem overwhelming but the creator makes it very easy. In fact, this is usually done in a single line. For example, the following creates a FitnessMin class for a minimization problem and an Individual class that is derived from a list with a fitness attribute set to the just created fitness.

```
from deap import base, creator
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
```

That’s it. More on creating types can be found in the Creating Types tutorial.

Initialization

Once the types are created you need to fill them with sometimes random values or sometime guessed ones. Again, DEAP provides an easy mechanism to do just that. The Toolbox is a container for tools of all sorts including initializers that can do what is needed of them. The following takes on the last lines of code to create the initializers for individuals containing random floating point numbers and for a population that contains them.

```
import random
from deap import tools
IND_SIZE = 10
toolbox = base.Toolbox()
toolbox.register("attribute", random.random)
toolbox.register("individual", tools.initRepeat, creator.Individual,
                toolbox.attribute, n=IND_SIZE)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

- This creates functions to initialize populations from individuals that are themselves initialized with random float numbers. The functions are registered in the toolbox with their default arguments under the given name. For example, it will be possible to call the function toolbox.population() to instantly create a population. More initialization methods are found in the Creating Types tutorial and the various Examples.

Operators

- Operators are just like initializers, except that some are already implemented in the tools module. Once you've chosen the perfect ones, simply register them in the toolbox. In addition you must create your evaluation function. This is how it is done in DEAP.

```
def evaluate(individual):
    return sum(individual),

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate)
```

- The registered functions are renamed by the toolbox, allowing generic algorithms that do not depend on operator names. Note also that fitness values must be iterable, that is why we return a tuple in the evaluate function. More on this in the Operators and Algorithms tutorial and Examples. Algorithms Now that everything is ready, we can start to write our own algorithm. It is usually done in a main function. For the purpose of completeness we will develop the complete generational algorithm.

```
def main():
    pop = toolbox.population(n=50)
    CXPB, MUTPB, NGEN = 0.5, 0.2, 40

    # Evaluate the entire population
    fitnesses = map(toolbox.evaluate, pop)
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit
    for g in range(NGEN):
        # Select the next generation individuals
        offspring = toolbox.select(pop, len(pop))
        # Clone the selected individuals
        offspring = map(toolbox.clone, offspring)

        # Apply crossover and mutation on the offspring
        for child1, child2 in zip(offspring[::2], offspring[1::2]):
            if random.random() < CXPB:
                toolbox.mate(child1, child2)
                del child1.fitness.values
                del child2.fitness.values

        for mutant in offspring:
            if random.random() < MUTPB:
                toolbox.mutate(mutant)
                del mutant.fitness.values

        # Evaluate the individuals with an invalid fitness
        invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
        fitnesses = map(toolbox.evaluate, invalid_ind)
        for ind, fit in zip(invalid_ind, fitnesses):
            ind.fitness.values = fit

        # The population is entirely replaced by the offspring
        pop[:] = offspring

    return pop
```

It is also possible to use one of the four algorithms readily available in the algorithms module, or build from some building blocks called variations also available in this module.

Installation

Requirements

DEAP is compatible with Python 2.7 and 3.4 or higher. The computation distribution requires SCOOP. CMA-ES requires Numpy, and we recommend matplotlib for visualization of results as it is fully compatible with DEAP's API.

Install DEAP

We encourage you to use easy_install or pip to install DEAP on your system. Linux package managers like apt-get, yum, etc. usually provide an outdated version.

easy_install deap

or

pip install deap

If you wish to build from sources, download or clone the repository and type:

python setup.py install

Conclusion:

DEAP provides a flexible and efficient framework for implementing evolutionary algorithms, allowing easy customization of selection, mutation, and crossover strategies. It simplifies complex optimization problems and enhances computational efficiency in evolutionary computing. This makes it a powerful tool for solving real-world optimization challenges across various domains.

Questions:

Q1.What is Genetic algorithm with DEAP?

Q2.What is DEAP Framework?

Q3.Why Genetic algorithm used in optimization?

Q4.What is DEAP library in Python?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 9

Title: Design and develop a distributed application

Name of the Student: ______________________________

Class: BE **Batch:**

Date:	Mark: /10

Signature of the Course In-charge: ______________________

Signature of the HOD: ________________________________

Experiment No: 9

Aim: To design and develop a distributed application that facilitates efficient communication and data processing across multiple systems, enabling scalability, fault tolerance, and resource sharing.

- **Outcome:** The distributed application will allow multiple nodes (machines or systems) to communicate and collaborate in real-time, allowing tasks to be executed in parallel across distributed resources. The outcome will include the ability to manage the distributed environment, handle task synchronization, and ensure reliability in the face of failures.
- **Hardware Requirement:** Computer System, Linux(Ubuntu), Shared storage or cloud storage (for storing large data sets or logs if required).
- **Software Requirement:** PyCharm IDE, JavaScript/React (optional, for frontend if web-based)

Theory:

A **distributed application** consists of several independent nodes (computers or servers) that communicate and work together to perform a task. Unlike centralized applications, where a single server handles everything, a distributed system uses multiple machines to divide the workload, offering advantages in terms of scalability, reliability, and fault tolerance.

Key concepts include:

- **Communication**: Nodes communicate via messaging systems like **REST APIs**, **message queues** (RabbitMQ, Redis), or **RPC (Remote Procedure Calls)**.
- **Task Distribution**: Tasks are split into smaller jobs that can be processed in parallel by different nodes.
- **Fault Tolerance**: In case one node fails, others continue to work, ensuring the system remains operational.

- **Load Balancing**: Tasks are distributed based on the load, so no single node is overwhelmed.
- **Concurrency & Synchronization**: Ensuring tasks are executed without conflicts and results are synchronized.

Steps in Developing a Distributed Application:

1. **Define System Architecture**: Decide the type of distributed system: client-server, peer-to-peer, or master-worker.
2. **Set Up Communication**: Choose a communication protocol (REST API, messaging queue, etc.) for nodes to exchange data.
3. **Task Distribution**: Develop a task distribution mechanism. For example, using **Celery** with **RabbitMQ** for task scheduling and queuing.
4. **Fault Tolerance**: Implement mechanisms to recover from node failures (e.g., re-queue failed tasks).
5. **Deploy and Monitor**: Use **Docker** and **Kubernetes** for deployment and orchestration (optional for large-scale systems). Monitor system health, performance, and logs.

Distributed Application Example: Task Scheduling System

For this example, let's create a distributed application that performs background task processing. The tasks will be distributed to worker nodes, and a central server will manage the task queues.

1. **Setup Flask Backend for Task Management**:
    - The backend will receive tasks and push them to a message broker (RabbitMQ).
2. **Celery Workers**:
    - Workers will consume tasks from the queue, process them, and return results.
3. **Task Queuing System**:
    - Tasks are queued in **RabbitMQ**, and each worker consumes tasks from this queue.

Sample Code (Backend with Flask + Celery + RabbitMQ)

app.py (Flask Backend)

```
from flask import Flask, request, jsonify
from celery import Celery

app = Flask(__name__)

# Configure Celery to use RabbitMQ as the broker
app.config['CELERY_BROKER_URL'] = 'pyamqp://guest@localhost//'
app.config['CELERY_RESULT_BACKEND'] = 'rpc://'

celery = Celery(app.name, broker=app.config['CELERY_BROKER_URL'])
celery.conf.update(app.config)

# Define a simple task to be processed
@celery.task
def long_task(x):
    return x * x

@app.route('/start_task', methods=['POST'])
def start_task():
    data = request.json
    task = long_task.apply_async(args=[data['number']])
    return jsonify({"task_id": task.id}), 202

@app.route('/get_result/<task_id>', methods=['GET'])
def get_result(task_id):
    task = long_task.AsyncResult(task_id)
    if task.state == 'PENDING':
        return jsonify({"status": "Pending"}), 202
    elif task.state == 'SUCCESS':
        return jsonify({"status": "Completed", "result": task.result}), 200
    return jsonify({"status": "Failed"}), 400

if __name__ == '__main__':
    app.run(debug=True)
```

worker.py (Celery Worker)

```
from app import celery

if __name__ == '__main__':
    celery.start()
```

Running the Application:

1. **Install dependencies:**

```
pip install Flask Celery
```

Run RabbitMQ (if not installed, install it using Docker or via a package manager).

Start Flask Application:

```
python app.py
```

Run Celery Worker:

```
celery -A app.celery worker
```

You can now make POST requests to `http://localhost:5000/start_task` to start tasks, and check their status via `http://localhost:5000/get_result/<task_id>`.

Classification:

This system is a basic example of a **distributed task processing application**. It falls into the **Client-Server** architecture, where:

- The **Client** (frontend or API consumers) sends tasks to the server.
- The **Server** manages the task queue.
- **Workers** (distributed nodes) process the tasks asynchronously.

Conclusion:

The design and development of distributed applications can greatly improve the efficiency and scalability of software systems. By dividing tasks across multiple nodes and ensuring that the system can recover from failures, a distributed application can provide high availability and fault tolerance. The use of message brokers (like RabbitMQ) and task queuing systems (like Celery) further facilitates managing workloads across multiple systems.

Questions:

1. What is the main advantage of using a distributed system over a centralized system?
2. How does Celery help in managing distributed tasks?
3. What is the role of Rabbit MQ in this distributed application, and why is it necessary?
4. How would you handle fault tolerance in a distributed application?
5. What are the challenges involved in ensuring synchronization and consistency in a distributed system?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	

Experiment No: 10

Title: Implement Ant colony optimization

Name of the Student: __

Class: **BE** **Batch:**

Date:

Mark: /10

Signature of the Course In-charge: ____________________

Signature of the HOD: ______________________________

Experiment No: 10

Aim: The aim of implementing the Ant Colony Optimization (ACO) algorithm is to demonstrate how this nature-inspired optimization technique can be applied to solve combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) or other path finding problems.

- **Outcome:** By implementing ACO, the outcome is the ability to find optimized solutions for path finding or optimization problems by simulating the foraging behavior of ants. Ants in the algorithm will communicate indirectly using pheromones to find the shortest or best paths in a graph. The application of this algorithm will lead to the development of a solution that improves over time with each iteration.
- **Hardware Requirement:** Computer System, Linux(Ubuntu)
- **Software Requirement:** PyCharm IDE

Theory:

Ant Colony Optimization (ACO) is a nature-inspired optimization algorithm based on the foraging behavior of ants. In nature, ants deposit a chemical substance known as **pheromones** on the ground when they find food. Other ants are attracted to paths that have stronger pheromone concentrations, and this leads to the discovery of the shortest path to the food source.

ACO algorithm mimics this behavior to solve combinatorial optimization problems. The main idea is to have **artificial ants** search for solutions by moving through different possible solutions, leaving pheromone trails that influence the movement of subsequent ants.

Key components of ACO:

- **Pheromone Update:** Ants leave pheromones as they travel through the solution space. The intensity of the pheromone increases if the path is part of a good solution.
- **Heuristic Information:** In some problems, additional information (heuristic) about the problem domain can guide ants toward better solutions.
- **Ant Movement:** At each step, ants probabilistically choose their next step based on the pheromone levels and heuristic information.
- **Evaporation:** Pheromone levels decrease over time to prevent the algorithm from getting stuck in suboptimal solutions.

Steps in ACO Algorithm:

1. **Initialization:** Randomly initialize the pheromone levels on each edge of the graph.
2. **Ant Movement:** Place ants at starting points and let them search for solutions by traversing the graph.
3. **Pheromone Update:** After ants complete their tours, update the pheromone levels.
4. **Evaporation:** Reduce the pheromone intensity over time to simulate evaporation.
5. **Termination:** The algorithm terminates when a stopping condition is met (e.g., after a fixed number of iterations or when a satisfactory solution is found).

Implementation Example: Solving the Traveling Salesman Problem (TSP) using ACO

Let's implement a basic version of the Ant Colony Optimization algorithm to solve the **Traveling Salesman Problem (TSP)**. In TSP, the goal is to find the shortest possible route that visits each city exactly once and returns to the starting city.

Step-by-Step Code Implementation:

```
        import numpy as np
import random
import matplotlib.pyplot as plt

# Problem setup: A list of cities (coordinates)
cities = np.array([
    [0, 0], [1, 3], [2, 5], [5, 2], [7, 6], [8, 3]
])

# Calculate the distance matrix
def distance_matrix(cities):
    n = len(cities)
    dist_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            dist_matrix[i][j] = np.linalg.norm(cities[i] - cities[j])
    return dist_matrix

# Parameters for the ACO algorithm
n_ants = 5
n_iterations = 100
alpha = 1.0    # Influence of pheromone
beta = 2.0     # Influence of distance
rho = 0.1      # Pheromone evaporation rate
Q = 100        # Total pheromone to deposit

# Initialize pheromone levels
n_cities = len(cities)
pheromone = np.ones((n_cities, n_cities))  # Start with uniform pheromone distribution

# Initialize the distance matrix
dist_matrix = distance_matrix(cities)

# Function to calculate the path length
def path_length(path, dist_matrix):
    length = 0
    for i in range(len(path) - 1):
        length += dist_matrix[path[i], path[i + 1]]
    length += dist_matrix[path[-1], path[0]]  # Return to start
    return length

# Ant Colony Optimization Algorithm
def aco_tsp(n_ants, n_iterations, alpha, beta, rho, Q):
    best_path = None
    best_length = float('inf')

    for _ in range(n_iterations):
        all_paths = []
        all_lengths = []

        # Move ants
        for ant in range(n_ants):
            path = [random.randint(0, n_cities - 1)]
            while len(path) < n_cities:
                current_city = path[-1]
                probabilities = []
                for next_city in range(n_cities):
                    if next_city not in path:
                        pheromone_level = pheromone[current_city, next_city] ** alpha
                        distance = dist_matrix[current_city, next_city] ** beta
                        probabilities.append(pheromone_level * distance)
                    else:
                        probabilities.append(0)
                probabilities = np.array(probabilities) / sum(probabilities)
                next_city = np.random.choice(range(n_cities), p=probabilities)
                path.append(next_city)

            # Calculate the path length
            length = path_length(path, dist_matrix)
            all_paths.append(path)
            all_lengths.append(length)

            # Update best solution found
            if length < best_length:
                best_length = length
                best_path = path

        # Pheromone evaporation
        pheromone *= (1 - rho)

        # Pheromone update
        for path, length in zip(all_paths, all_lengths):
            pheromone_deposit = Q / length
            for i in range(len(path) - 1):
                pheromone[path[i], path[i + 1]] += pheromone_deposit
            pheromone[path[-1], path[0]] += pheromone_deposit

    return best_path, best_length

# Running ACO on the TSP problem
best_path, best_length = aco_tsp(n_ants, n_iterations, alpha, beta, rho, Q)

# Output the result
```

```
print(f"Best path: {best_path}")
print(f"Best path length: {best_length}")

# Visualize the solution
best_cities = cities[best_path]
best_cities = np.vstack([best_cities, best_cities[0]])  # Close the loop
plt.plot(best_cities[:, 0], best_cities[:, 1], 'bo-', markersize=8)
plt.scatter(cities[:, 0], cities[:, 1], color='red')
plt.title("ACO Solution for TSP")
plt.show()
```

Explanation of the Code:

- **Distance Matrix:** We calculate the distance matrix between cities using Euclidean distance.
- **ACO Parameters:** We set parameters like the number of ants, the number of iterations, pheromone influence (alpha), and distance influence (beta).
- **Ant Movement:** Ants probabilistically select the next city based on pheromone levels and distances.
- **Pheromone Update:** After each ant completes a path, pheromone is deposited along the path in proportion to the path's quality (shorter paths receive more pheromone).
- **Evaporation:** Pheromone levels decay over time to allow the algorithm to explore new paths.

Classification:

This implementation of ACO is a **combinatorial optimization algorithm** and can be classified as a **nature-inspired algorithm**. It simulates the behavior of ants to find an optimal or near-optimal solution to optimization problems, such as TSP.

Conclusion:

Ant Colony Optimization is an efficient heuristic algorithm inspired by natural processes. It can effectively solve combinatorial optimization problems by simulating the collective behavior of ants. The algorithm improves over time as ants reinforce good solutions with pheromone trails. ACO can be applied to various problems, including routing, scheduling, and resource allocation.

Questions:

1. What is the basic principle behind the Ant Colony Optimization algorithm?
2. How do pheromone levels influence the movement of ants in the ACO algorithm?
3. What are the main parameters in ACO, and how do they affect the performance of the algorithm?
4. How does the evaporation of pheromones prevent the algorithm from getting stuck in local optima?
5. Can ACO be applied to real-world problems outside of combinatorial optimization, such as machine learning?

Coding Efficiency	**Viva**	**Timely Completion**	**Total**	**Dated Sign of Course In-charge**
5	**3**	**2**	**10**	