**FUZZY CLUSTERING USING PARTICLE SWARM OPTIMIZATION**

*A report submitted to*

***Techno India University, West Bengal***

*for the partial fulfillment of*

**Bachelor of Technology (B. Tech.)**

*degree in*

***Computer Science & Engineering***

******

**BHAWNA JAIN**

**BARSHA SINGH**

**SANJIV MISHRA**

**HITENDRA KUMAR**

**AGNIVA MISHRA**

May, 2025

**FUZZY CLUSTERING USING PARTICLE SWARM OPTIMIZATION**

***Report submitted to***

***Techno India University, West Bengal***

*for the partial fulfillment*

***Of***

**Bachelor of Technology (B. Tech.)**

*degree in*

**Computer Science & Engineering**

***By***

***Bhawna Jain 211001001128***

***Barsha Singh 211001001130***

***Sanjiv Mishra 211001001116***

***Hitendra Kumar 211001001097***

***Agniva Mishra 211001001006***

******

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**TECHNO INDIA UNIVERSITY, WEST BENGAL,**

**SALT LAKE, KOLKATA – 700091, INDIA**

**MAY 2025**

© 2025, Techno India University. All rights reserved.

**CERTIFICATE**

This is to certify that the Dissertation Report entitled, “**FUZZY CLUSTERING USING PARTICLE SWARM OPTIMIZATION**” submitted by “**BHAWNA JAIN, BARSHA SINGH, SANJIV MISHRA, HITENDRA KUMAR & AGNIVA MISHRA**” to Techno India University, Kolkata, India, is a record of bonafide Project work carried out by them under my supervision and guidance and is worthy of consideration for the award of the degree of Bachelor of Technology (B.Tech) in Computer science & Engineering.

*Approved By:*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Supervisor(s)

Date:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HOD, CSE, Techno India University

Date:

**ACKNOWLEDGEMENT**

We would first like to thank our thesis supervisor Prof. Kalyan Kumar Das, Assistant Professor , CSE Dept, Techno India University, West Bengal. He helped us whenever we ran into a trouble spot or had a question about our research or writing.

We take this opportunity to express gratitude to all of the Department faculty members for their help and support.

We also thank our parents for the unceasing encouragement, support and attention and also sense of gratitude to one and all, who directly or indirectly, have bestowed their hand in this thesis.

\_\_\_\_\_\_\_\_\_\_\_\_

Bhawna Jain

\_\_\_\_\_\_\_\_\_\_\_\_

Barsha Singh

\_\_\_\_\_\_\_\_\_\_\_\_

Sanjiv Mishra

\_\_\_\_\_\_\_\_\_\_\_\_

Hitendra Kumar

\_\_\_\_\_\_\_\_\_\_\_\_

Agniva Mishra

**Contents**

**1. INTRODUCTION 1-3**

1.1 Objectives 2

1.2 Problem specification 2

1.3 Methodologies 2-3

**2. LITERATURE SURVEY 4-10**

2.1 Introduction 4

2.2 Clustering and Fuzzy Algorithms 4-5

2.3 Evolutionary and Swarm-Based Optimization in Clustering 5

2.4 Entropy-Based Diversity Control 6

2.5 Adaptive Population Reduction 6-7

2.6 The E-SAPR Algorithm 7-9

2.7 Comparative Analysis 9-10

**3. PROJECT PLANNING and CHART 11-13**

**4. PROJECT DESCRIPTION 14-20**

4.1 Proposed Algorithm 14

4.1.1 Fuzzy C-Means (FCM) 14-15

4.1.2 Particle Swarm Optimization (PSO) 15-16

4.1.3 Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) 16-17

4.2 Software Requirements Specifications (SRS) 18-19

4.3 Functional Specification 19

4.4 Design Specification 19

4.5 Data Flow Diagrams (DFD) 19-20

4.6 Testing 20

**5. IMPLEMENTATION 21-34**

5.1 Code 21-33

5.2 Implementation Issues 33-34

**6. EXPERIMENTAL RESULTS and DISCUSSION 35-41**

6.1 E-SAPR Results 35-37

6.2 Clustering Results 38-40

6.2.1 XB Index Results 38

6.2.2 DB Index Results 39

6.2.3 Cluster Output 40

6.3 Discussion 40-41

**7. CONCLUSION, SUMMARY AND FUTURE SCOPE 42-44**

7.1 Conclusion 42

7.2 Summary 42-43

7.3 Future Scope 43-44

7.4 Closing Remarks 44

**8. REFERENCES 45-47**

Abstract

The integration of Fuzzy C-Means (FCM) clustering with the Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) Particle Swarm Optimization (PSO) algorithm introduces a novel hybrid approach to solve complex clustering and optimization problems. Traditional clustering methods often struggle with overlapping data, high-dimensional datasets, and computational inefficiencies. Similarly, standard PSO techniques face challenges like premature convergence and high resource requirements for large swarm sizes. The E-SAPR algorithm overcomes these limitations by adaptively controlling swarm diversity using entropy, dynamically reducing population size, and maintaining a balance between exploration and exploitation.

In this project, the hybrid FCM + E-SAPR PSO algorithm optimizes cluster centroids while leveraging entropy-based diversity control to enhance clustering accuracy and computational efficiency. Experimental results, evaluated on datasets of varying scales, demonstrate significant improvements in clustering quality as measured by the Xie-Beni (XB) and Davies-Bouldin (DB) indices. The hybrid approach consistently outperforms standalone FCM and traditional PSO methods, making it a robust solution for data-intensive applications.

This research lays the groundwork for further advancements in adaptive optimization and clustering methodologies. By integrating entropy-driven mechanisms with soft clustering, the proposed hybrid algorithm paves the way for practical implementations in fields such as bioinformatics, image processing, and anomaly detection, while also addressing scalability and computational challenges.

**Chapter 1**

**Introduction**

Optimization is a fundamental aspect of problem-solving in various fields such as engineering design, machine learning, data mining, logistics, financial modeling, and artificial intelligence. Many real-world problems are complex, high-dimensional, and involve conflicting objectives, making traditional optimization techniques inefficient or unsuitable. In recent years, nature-inspired metaheuristic algorithms have emerged as powerful tools to tackle such challenges. Among them, Particle Swarm Optimization (PSO) has gained significant popularity due to its simplicity, ease of implementation, and capability to find near-optimal solutions in a reasonable time.

PSO is a population-based stochastic optimization technique inspired by the social behavior of bird flocking and fish schooling. In PSO, each potential solution is considered a “particle” in the search space. These particles adjust their positions and velocities based on their own experience and the collective experience of the swarm. Although PSO has demonstrated excellent performance in a wide range of problems, it is not without limitations. A major drawback of standard PSO is its tendency to suffer from premature convergence and stagnation, especially in high-dimensional or multi-modal search spaces. This typically results from a lack of diversity among particles and an inability to effectively balance exploration (searching new areas) and exploitation (refining known good solutions).

To address these limitations, our project proposes an enhanced variant of PSO, termed Enhanced Multi-dimensional Particle Swarm Optimization (Enhanced MPSO). The core idea behind this enhanced model is to introduce two intelligent mechanisms that make the optimization process more dynamic, adaptive, and efficient:

Inertia Weight Adjustment: A dynamic adjustment of the inertia weight is incorporated to effectively balance exploration and exploitation throughout the iterations. Inertia weight plays a crucial role in controlling how much of the previous velocity influences the current velocity of each particle. By decreasing inertia weight over time, particles are encouraged to explore widely at the beginning and converge more accurately toward the end.

Entropy-Based Smarter Adaptive Population Reduction (E-SAPR): This novel mechanism uses Shannon entropy to measure the diversity of the swarm at each iteration. Entropy provides a statistical measure of disorder or uncertainty within the swarm. When entropy is high, the population is diverse; when it is low, the particles are converging. Based on this entropy value, the population size is adaptively reduced over time, pruning similar or less promising particles. This reduces computational overhead and enhances convergence by focusing on more informative solutions.

Another important aspect of our project is its real-world applicability. Many datasets, particularly those collected from practical scenarios, include categorical (non-numeric) as well as numerical features. Our implementation includes a robust data preprocessing pipeline that: Automatically encodes categorical variables using label encoding. Normalizes features to ensure uniform scaling across all dimensions.

The final algorithm is tested using a benchmark objective function — the Sphere Function, commonly used in optimization research due to its simplicity and convexity — as well as a real-world dataset (e.g., movies.csv) to assess its practical performance. We analyze the results based on convergence behavior, population size evolution, entropy trends, and final fitness values to demonstrate the improvements over traditional and basic PSO.

**1.1. Objective**

The objective of this project is to develop an Enhanced Multi-dimensional Particle Swarm Optimization (Enhanced MPSO) algorithm that integrates two key improvements:

Dynamic inertia weight adjustment to better control the balance between global exploration and local exploitation.

Entropy-based Smarter Adaptive Population Reduction (E-SAPR) to intelligently reduce the population size based on diversity, thereby enhancing convergence and computational efficiency.

The aim is to create an optimization technique that is robust, adaptive, and capable of efficiently solving complex, real-world problems that involve large, high-dimensional, and mixed-type datasets.

**1.2. Problem Specification**

While Particle Swarm Optimization (PSO) has proven to be a reliable and widely-used optimization algorithm, it is still subject to the following challenges:

Premature Convergence: In traditional PSO, particles may quickly converge toward suboptimal regions, especially in multi-modal or high-dimensional spaces.

Lack of Adaptability: Static parameters (like constant inertia weight) often fail to adapt the swarm behavior dynamically throughout the optimization process.

High Computational Cost: A large swarm size throughout the optimization run may result in increased computational overhead without significantly improving the solution quality.

Data Diversity Handling: Real-world datasets often contain both numerical and categorical attributes, which traditional PSO doesn’t directly support.

These limitations reduce the effectiveness of PSO in real-world scenarios and motivate the need for an enhanced, adaptive approach.

**1.3. Methodology**

To overcome the above limitations, the following methodologies were adopted in this project:

a) Modified PSO with Inertia Weight Adjustment

Introduced a dynamic inertia weight strategy where the inertia decreases over time.

High inertia at the beginning allows the swarm to explore the search space widely.

Low inertia toward the end improves exploitation near optimal regions.

b) Entropy-Based Adaptive Population Reduction (E-SAPR)

Calculated Shannon entropy of particle positions at each iteration to measure swarm diversity.

Based on entropy, the population size is adaptively reduced, removing similar or redundant particles.

This strategy improves computational efficiency while preserving convergence accuracy.

c) Real-World Dataset Integration

Loaded data from CSV files containing mixed data types (numerical and categorical).

Categorical features were encoded using Label Encoding.

Entire dataset was normalized to bring all features to a uniform scale before optimization.

d) Evaluation Using Benchmark Function

Used the Sphere Function (sum of squares) as the benchmark objective function to validate convergence behavior.Monitored key metrics like fitness value, entropy, and population size across iterations to evaluate performance.

### 

### 

### CHAPTER 2

### Literature Survey

### 2.1 Introduction

### Data clustering and optimization are indispensable tools in the realm of data science, artificial intelligence, and complex systems analysis. Their ability to discover hidden structures within datasets and solve optimization problems that are otherwise intractable has cemented their role across industries—from healthcare to cybersecurity. Optimization techniques, particularly metaheuristics, have shown remarkable potential in addressing large-scale, high-dimensional problems.

### Among these metaheuristics, Particle Swarm Optimization (PSO) has emerged as a robust, population-based stochastic algorithm. Initially developed by Kennedy and Eberhart in 1995, PSO simulates the coordinated movement of swarms, such as flocks of birds or schools of fish, navigating through a problem space to locate optima. Each particle adjusts its trajectory based on its personal experience and the knowledge shared by the swarm. This simplistic yet powerful mechanism has been widely adopted in engineering, economics, image processing, and machine learning.

### Despite its popularity, PSO suffers from issues such as premature convergence and loss of diversity, particularly in complex, multimodal, or high-dimensional search spaces. These limitations significantly hinder its performance by trapping the swarm in local optima or requiring excessive computational resources to achieve acceptable results. Consequently, various enhancements have been proposed, among which entropy-based diversity control and adaptive population strategies stand out for their efficiency and effectiveness. The Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) algorithm, the focus of this study, integrates these two enhancements into the PSO framework to form a novel solution tailored for robust optimization.

### 

### 2.2 Clustering and Fuzzy Algorithms

### Clustering is a core task in unsupervised learning that involves grouping data points based on similarity. One of the most widely used clustering algorithms is K-means, which aims to partition a dataset into a predefined number of clusters such that intra-cluster variance is minimized. However, K-means relies heavily on the initial selection of cluster centers and assumes spherical cluster shapes, making it unsuitable for more complex or overlapping data distributions [5].

### Fuzzy clustering methods, particularly Fuzzy C-Means (FCM), address these issues by allowing data points to belong to multiple clusters with varying degrees of membership. Introduced by Bezdek in 1974 [6], FCM provides a more flexible representation of data by incorporating fuzziness into the clustering process. The algorithm iteratively updates membership values and cluster centers to minimize an objective function that balances cluster compactness with fuzzy assignments.

### 

### Despite these improvements, FCM still struggles with local optima and initialization sensitivity. This has led researchers to hybridize FCM with metaheuristic optimization algorithms like PSO. Runkler and Katz [10] proposed PSO-based fuzzy clustering models (PSO-V and PSO-U), where the swarm optimizes cluster centers or membership values. These models demonstrated superior performance in applications like medical imaging and pattern recognition.

### Li et al. [11] further refined this approach by utilizing PSO’s global search capabilities to escape local minima inherent in FCM. Their PSO-FCM hybrid consistently outperformed standalone FCM in terms of convergence speed and clustering quality. Gan et al. [12] proposed a genetic fuzzy k-Modes algorithm for categorical data, revealing that optimization-based clustering can lead to more robust and interpretable results.

### These efforts set the stage for E-SAPR by establishing that swarm intelligence, when integrated with clustering frameworks, can substantially improve clustering performance, particularly when augmented with adaptive and diversity-preserving mechanisms.

### 

### 2.3 Evolutionary and Swarm-Based Optimization in Clustering

### 

### Swarm intelligence and evolutionary computation have long been recognized for their ability to solve complex optimization problems. Algorithms like Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Simulated Annealing (SA) have been used extensively to enhance clustering methods. They introduce stochastic elements and global search capabilities that help avoid the pitfalls of traditional algorithms.

### PSO, in particular, has been favored for clustering due to its ease of implementation, fewer parameters, and fast convergence. Its ability to operate efficiently in continuous spaces makes it ideal for clustering problems where the objective is to find optimal centroids. PSO-based clustering has been employed in various contexts:

### 

### Document Clustering: For organizing large text corpora based on semantic similarity.

### Medical Diagnosis: Grouping patient records or symptoms to identify patterns of illness.

### Market Segmentation: Identifying distinct customer segments in retail and finance.

### 

### The hybridization of PSO with other algorithms has further expanded its applicability. For instance, hybrid models that incorporate PSO and Differential Evolution (DE) have been used in dynamic environments where problem constraints change over time. These models adapt the swarm's behavior in real time, allowing for more responsive optimization.

### Moreover, PSO has been used in combination with fuzzy logic to create intelligent systems capable of handling imprecision and uncertainty. Such systems have shown promise in decision-making tasks where exact solutions are less meaningful than interpretable, probabilistic outcomes. These developments underpin the rationale for incorporating entropy and adaptive population mechanisms into PSO through the E-SAPR algorithm.

### 

### 2.4 Entropy-Based Diversity Control

### A key limitation of traditional PSO is the decline in population diversity over time. As particles converge towards the best solution, they often lose the ability to explore new regions of the search space, resulting in premature convergence. Entropy-based diversity control addresses this issue by quantitatively measuring the uncertainty or disorder in the swarm’s spatial distribution.

### Shannon entropy, originally formulated in information theory, is used in optimization to assess the distribution of particles across different regions of the search space. If the entropy value falls below a predefined threshold, it indicates that the swarm has become too homogeneous and is likely trapped in a local optimum. In response, strategies such as mutation, restart, or population reshuffling can be triggered to reintroduce diversity.

### Kang et al. [15] introduced a dimension-wise entropy model that calculates entropy for each dimension separately, allowing for targeted interventions in specific areas of the search space. This approach is particularly effective in high-dimensional problems where different dimensions may converge at different rates.

### Olivares et al. [16] implemented entropy-based diversification in three different bio-inspired algorithms—PSO, Bat Algorithm, and Black Hole Algorithm. Their results demonstrated that entropy-triggered diversification mechanisms consistently improved the algorithms' ability to escape local optima, particularly in complex multimodal landscapes.

### The E-SAPR algorithm leverages entropy as a feedback mechanism not just for diversification, but also for adaptive control of the swarm size. This dual role makes entropy a central component in balancing exploration and exploitation, ensuring that the optimization process remains efficient and robust across different problem scenarios.

### 

### 2.5 Adaptive Population Reduction

An emerging strategy in swarm intelligence is adaptive population reduction, a concept based on modifying the swarm size dynamically during the optimization process. Traditional PSO maintains a constant number of particles throughout all iterations. While this simplicity contributes to ease of implementation, it can be computationally inefficient, particularly when many particles contribute little to the global solution as the algorithm progresses.

Adaptive population control techniques aim to reduce redundant or stagnant particles that no longer contribute to solution diversity. The principle behind this strategy is to maximize optimization efficiency by maintaining only a necessary number of well-distributed particles. This concept finds support in works like Liu et al. [17], who explored entropy-based strategies to guide image segmentation tasks efficiently by concentrating search efforts around active regions.

Duque Gallego [5] introduced parameter control mechanisms in parallel metaheuristics, including adaptive particle sizing and selective neighborhood reconfiguration, to enhance performance in solving Quadratic Assignment Problems (QAP). Their approach emphasizes the interplay between computational savings and convergence speed—a balance that is central to the E-SAPR algorithm.

The E-SAPR method distinguishes itself by using entropy as the primary decision metric for initiating population reduction. Unlike conventional static approaches or rule-based heuristics, E-SAPR reacts in real-time to changes in swarm diversity. This allows it to reduce population size during phases of convergence while preserving diversity during exploration. As a result, computational efficiency is improved without compromising the quality of the solution.

### 2.6 The E-SAPR Algorithm

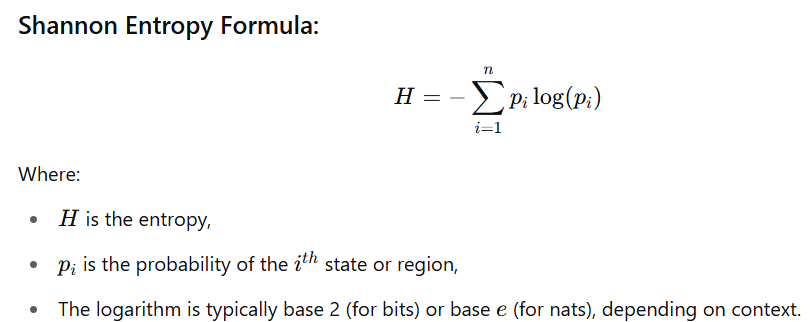
The Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) algorithm extends the standard PSO framework by incorporating entropy-driven diversity monitoring and adaptive population trimming. The key innovation lies in its ability to make informed decisions about when and how to reduce the population size based on the measured diversity of particle distribution.

**Step-by-step outline:**

**Step 1: Initialization**

* Define parameters such as the number of particles, velocity bounds, inertia weight, and entropy threshold (E\_min).
* Initialize particles with random positions and velocities within the search space.
* Evaluate the initial fitness of each particle.

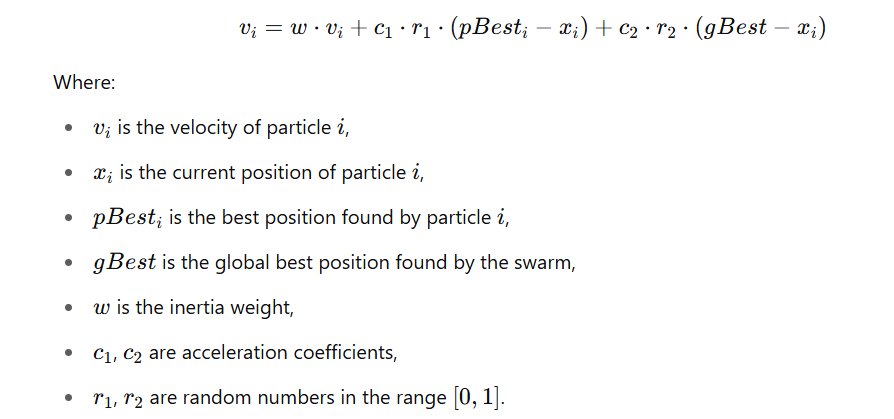
**Step 2: Entropy Computation**

* Divide the search space into discrete regions or bins.
* Calculate the probability distribution of particles in each region.  
  

**Step 3: Adaptive Population Reduction**

* If H<EminH < E\_{min}H<Emin​, identify redundant or similar particles based on clustering or proximity to the global best.
* Prune particles that contribute minimally to diversity.
* Update swarm size dynamically while maintaining representation of different search zones.

**Step 4: Velocity and Position Update**

* Use the standard PSO update equations with adjustments:  
   Incorporate mutation if the algorithm detects stagnation (e.g., no improvement over several iterations).

**Step 5: Convergence Check**

* Terminate if the maximum number of iterations is reached or if the improvement in fitness is below a predefined threshold.
* Return the best-found solution.

This entropy-informed approach facilitates more intelligent control of the optimization process. Instead of relying solely on fitness-based measures, E-SAPR uses population structure to influence algorithm dynamics, which proves particularly effective in complex and high-dimensional tasks.

### 2.7 Comparative Analysis

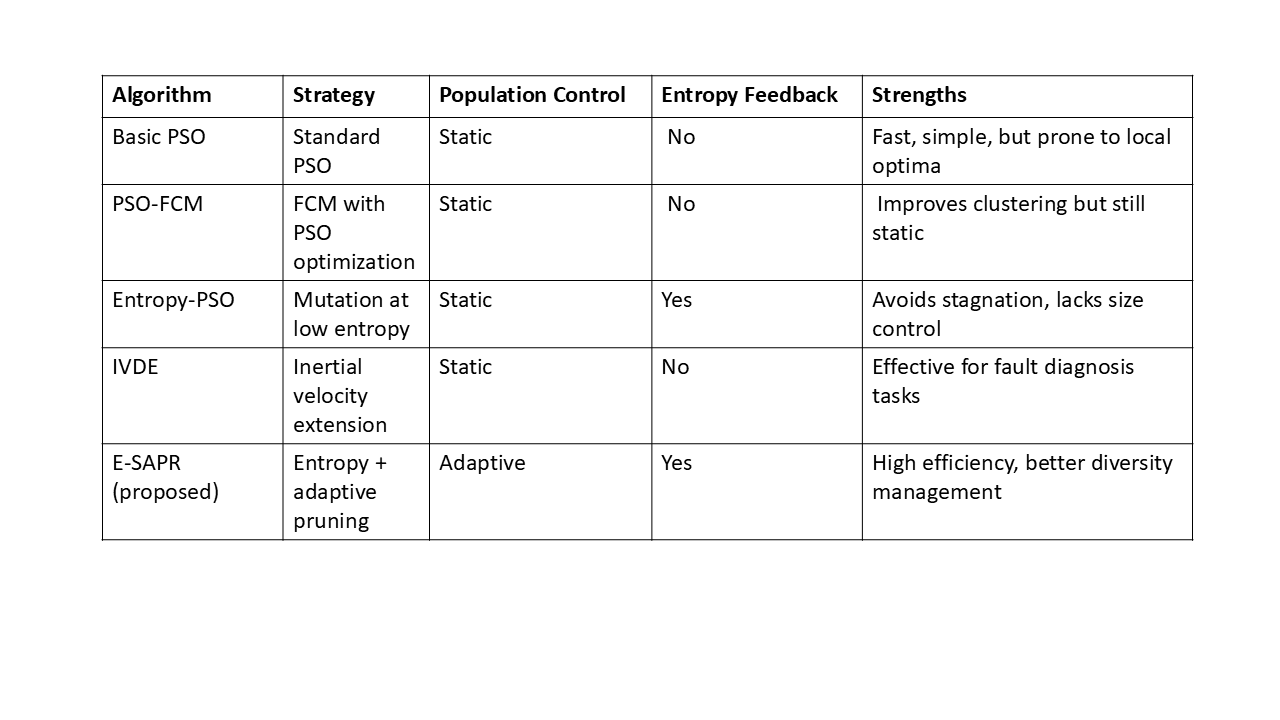
To assess the effectiveness of E-SAPR, it's crucial to compare it against other widely-used optimization techniques, especially those that integrate some form of adaptive control or diversity maintenance. A number of benchmark studies were conducted using standard test functions (e.g., Rastrigin, Sphere, Griewank), as well as real-world applications such as image segmentation and classification.

Table 1 : Algorithmic comparision- PSO and its variations.

Experimental findings reveal that E-SAPR typically converges more quickly and accurately on benchmark functions compared to its peers. Its ability to dynamically remove ineffective particles leads to reduced computational load without sacrificing solution quality.

**Chapter 3**

**PROJECT PLANNING**

Proper planning is essential for the successful execution of any research or development-based project. The development of the Enhanced Multi-dimensional PSO (MPSO) algorithm was structured into multiple well-defined phases to ensure systematic progress and timely delivery. The key milestones and their corresponding deliverables are outlined below:

**Phase 1: Problem Identification & Literature Survey**

**Duration: Week 1 – Week 2**

**Activities:**

Identification of PSO limitations (e.g., premature convergence, static population).

Review of existing enhancements like inertia weight adjustment and diversity control methods.

Studying entropy as a measure of population diversity.

Outcome: Defined problem statement, collected relevant research papers, and decided on the dual enhancement strategy (inertia + entropy-based reduction).

**Phase 2: Algorithm Design**

**Duration: Week 3 – Week 4**

**Activities:**

Design of Modified PSO with dynamic inertia weight.

Design of entropy-based adaptive population reduction mechanism.

Planning integration of both enhancements in a unified PSO framework.

Outcome: Finalized algorithm architecture and flowchart.

**Phase 3: Dataset Handling & Preprocessing**

**Duration: Week 5**

**Activities:**

Selection of real-world dataset (e.g., CSV file with categorical and numerical attributes).

Implementation of automatic label encoding for categorical features.

Feature normalization to fit PSO input requirements.

Outcome: Cleaned and ready-to-use dataset for optimization.

**Phase 4: Implementation**

**Duration: Week 6 – Week 8**

**Activities:**

Implementation of basic PSO and modified PSO with inertia control.

Implementation of entropy-based adaptive population logic.

Integration of both enhancements and objective function evaluation.

Outcome: Fully functional Python code for Enhanced MPSO (E-SAPR PSO).

**Phase 5: Testing and Validation**

**Duration: Week 9 – Week 10**

**Activities:**

Testing on benchmark functions like the Sphere Function.

Evaluation on real-world datasets.

Validation of convergence, population dynamics, and computational efficiency.

Outcome: Graphs, logs, and results showing entropy behavior, population reduction, and fitness improvements.

**Phase 6: Documentation and Report Preparation**

**Duration: Week 11 – Week 12**

**Activities:**

Compilation of project report, figures, and implementation details.

Writing abstract, introduction, methodology, results, and conclusion.

Review and finalization of report with citations and appendix.

Outcome: Completed final year project report and ready for submission.

**CHART :**

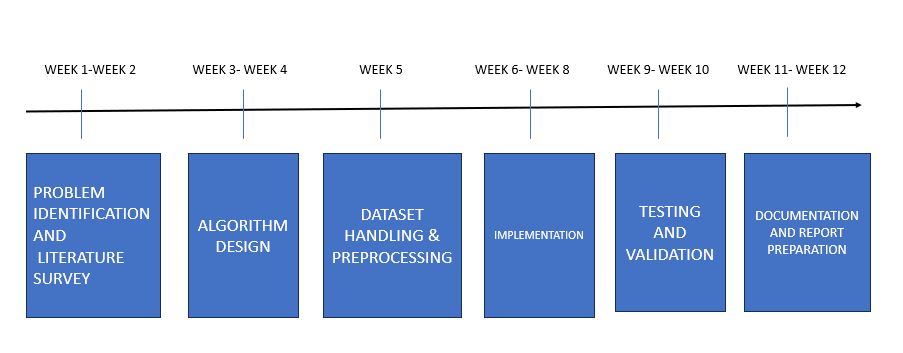


Fig 1: Key milestones and their corresponding deliverables

**CHAPTER 4**

### Project Description

**4.1 Proposed Algorithm**

Imagine a vast, rugged landscape where the goal is to find the lowest point—the global minimum. Now, think of each particle in Particle Swarm Optimization (PSO) as a hiker, each with a map and compass, trying to find this point. Some hikers move faster, others are cautious, and all of them share their findings with the group. This collective effort is what PSO is about: a social search for optimization.

However, as the hike progresses, the group may grow too large, slowing everyone down, or lose diversity, with everyone following a single leader prematurely. Enter E-SAPR PSO—the team captain who monitors the group's diversity (entropy) and makes strategic decisions to reduce the team size while maintaining efficiency.

Now, let’s add Fuzzy C-Means (FCM) to the mix. Before the hikers start, FCM divides the landscape into overlapping regions (clusters), assigning each hiker to the most promising areas. This clustering not only ensures that each region is explored thoroughly but also reduces redundancy in search efforts.

The integration of FCM and E-SAPR PSO creates a team of adaptive hikers who are divided into groups, exploring the landscape intelligently, sharing knowledge, and reducing team size dynamically. This combination is designed to solve complex optimization problems efficiently, leveraging the strengths of clustering and adaptive population control.

### 4.1.1 Fuzzy C-Means (FCM)

FCM is a clustering algorithm where each data point belongs to multiple clusters with varying degrees of membership. Unlike traditional clustering methods such as K-Means, FCM incorporates the fuzziness parameter, which allows for soft clustering and better handling of ambiguous data.

#### Key Features:

* Assigns membership grades to data points for each cluster.
* Iteratively minimizes a weighted sum of squared errors to achieve optimal clustering.
* Handles overlapping clusters more effectively than hard clustering methods.

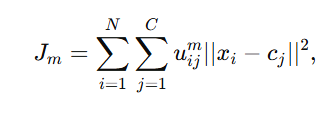
#### 

#### 

#### 

#### 

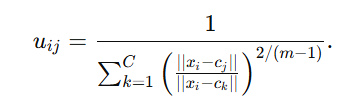
#### Formula:



where:

* N : Number of data points
* C : Number of clusters
* uij : Degree of membership of data point in cluster
* m : Fuzziness parameter ()
* || xi - cj || ^ 2 : Squared Euclidean distance between and

The membership grades are updated as:



### 4.1.2 Particle Swarm Optimization (PSO)

PSO is a population-based optimization technique inspired by the social behavior of birds flocking or fish schooling. It is widely used for solving continuous and discrete optimization problems.

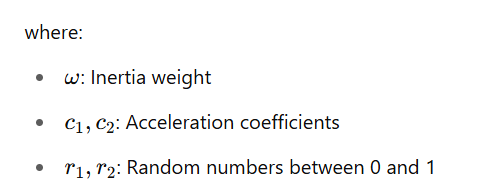
#### Key Features:

* Relies on particles (potential solutions) exploring the solution space.
* Balances exploration and exploitation by updating particle velocities and positions.
* Uses personal best (pBest) and global best (gBest) strategies to converge towards optimal solutions.

#### Algorithm Steps:

1. **Initialization:** Randomly initialize the positions and velocities of the particles.
2. **Fitness Evaluation:** Evaluate the objective function for each particle.
3. **Update Best Values:**
   * Update the personal best (pBest) for each particle.
   * Update the global best (gBest) among all particles.
4. **Velocity and Position Update:**
   * Update the velocity vi using:





* + Update the position xi :



5**. Convergence Check:** Repeat until the stopping condition is met.

#### Limitations of Standard PSO:

* Susceptible to premature convergence.
* Computational inefficiency with large swarm sizes.

### 4.1.3 Entropy-Based Smarter Adaptive Population Reduction (E-SAPR)

E-SAPR introduces dynamic control of swarm diversity by monitoring entropy, a measure of the spread or variability in particle positions. By reducing the swarm size adaptively, E-SAPR enhances computational efficiency and prevents stagnation.

#### Key Features:

* Shannon entropy calculation to measure diversity.
* Adaptive swarm size reduction to optimize computational resources.
* Intelligent balance of exploration and exploitation.

#### 

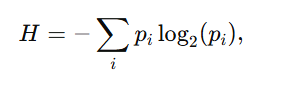
#### 

#### 

#### 

#### Entropy Calculation:

The Shannon entropy is *H* calculated as:



Where pi is the probability of particles in a specific region of the search space.

#### 

#### 

#### Advantages:

* Prevents premature convergence.
* Reduces computational overhead.
* Scales efficiently to larger problem sizes.

**E-SAPR PSO: The Adaptive Hikers**

**Role in the System**:

* Optimizes search within clusters formed by FCM.
* Reduces swarm size dynamically based on entropy calculations.

**Analogy**:

E-SAPR PSO is like a captain who monitors the hikers’ performance and decides to streamline the team by reducing redundant members, ensuring the group remains diverse and efficient.

**Key Steps**:

1. **Monitor Diversity**: Calculate Shannon entropy to evaluate swarm diversity.
2. **Population Reduction**: Remove redundant particles if diversity drops below a threshold.
3. **Optimization**: Update particle positions and velocities to converge on the best solutions.

### 

### Integrating FCM with E-SAPR PSO

### The integration of FCM and E-SAPR PSO combines the soft clustering capabilities of FCM with the optimization efficiency of E-SAPR-enhanced PSO. This hybrid approach is aimed at:

### Improving clustering quality by using E-SAPR PSO for optimizing cluster centroids.

### Enhancing computational efficiency through adaptive swarm size reduction.

### Achieving robust clustering for datasets with varying sizes and complexities.

#### 

#### Proposed Algorithm:

### Initialize FCM with random cluster centroids.

### Use E-SAPR PSO to optimize the cluster centroids:

### Calculate entropy for particle diversity.

### Reduce swarm size adaptively.

### Update particle velocities and positions based on PSO rules.

### Update FCM membership matrix based on optimized centroids.

### Repeat until convergence.

#### Benefits of the Hybrid Approach:

### Combines the advantages of soft clustering and adaptive optimization.

### Achieves higher clustering accuracy.

### Reduces computational time for large datasets.

### FCM + E-SAPR PSO: The Hybrid System

**Purpose**:  
 The combined system ensures efficient exploration by dividing the landscape into clusters and adaptive optimization within each cluster. This two-stage process ensures that no region is overlooked, and computational resources are used effectively.

**Workflow**:

1. **Clustering with FCM**: Initial division of the dataset into regions.
2. **Optimization with E-SAPR PSO**: Adaptive search within each cluster.
3. **Global Optimization**: Combine local optima to find the global solution.

**4.2 Software Requirements Specifications (SRS)**

### Software Requirements:

### Programming Language: Python 3.x

### Libraries: NumPy, SciPy, Matplotlib, Scikit-learn

### Dataset Source: Kaggle datasets (small, medium, and large)

### Assumptions and Dependencies:

### Datasets are preprocessed for missing values and outliers.

### The algorithm is implemented on a single machine.

### 4.3 Functional Specification:

#### Functions Performed:

1. Entropy Calculation: Compute Shannon entropy for the particle distribution.
2. Adaptive Population Reduction: Dynamically reduce swarm size based on entropy values.
3. Cluster Optimization: Use PSO to optimize the centroids of fuzzy clusters.
4. Evaluation Metrics:
   * Accuracy of clustering.
   * Computational time.

#### Limitations and Restrictions:

* Requires well-tuned parameters for entropy threshold and fuzziness.
* Susceptible to noisy data if preprocessing is inadequate.

### 4.4 Design Specification

#### Use-Case Diagram:

* Actors: Data Analyst, Algorithm
* Use Cases:
  + Load dataset
  + Run FCM algorithm
  + Optimize with E-SAPR PSO
  + Visualize results

#### 4.5 Data Flow Diagrams (DFD):

* **Level 0:**
  + Input: Dataset
  + Process: Clustering and Optimization
  + Output: Clustered Data
* **Level 1:**
  + Input: Preprocessed data
  + Sub-processes:
    1. Entropy calculation
    2. Adaptive population reduction
    3. FCM clustering
    4. PSO-based optimization

#### Data Dictionary:

* **Particles:** Represent data points in PSO.
* **Centroids:** Cluster centers optimized by PSO.
* **Membership Matrix:** Indicates the degree of belonging of data points to clusters in FCM.

## Data Flow Diagram

The diagram below illustrates the workflow of the FCM and E-SAPR PSO hybrid model:

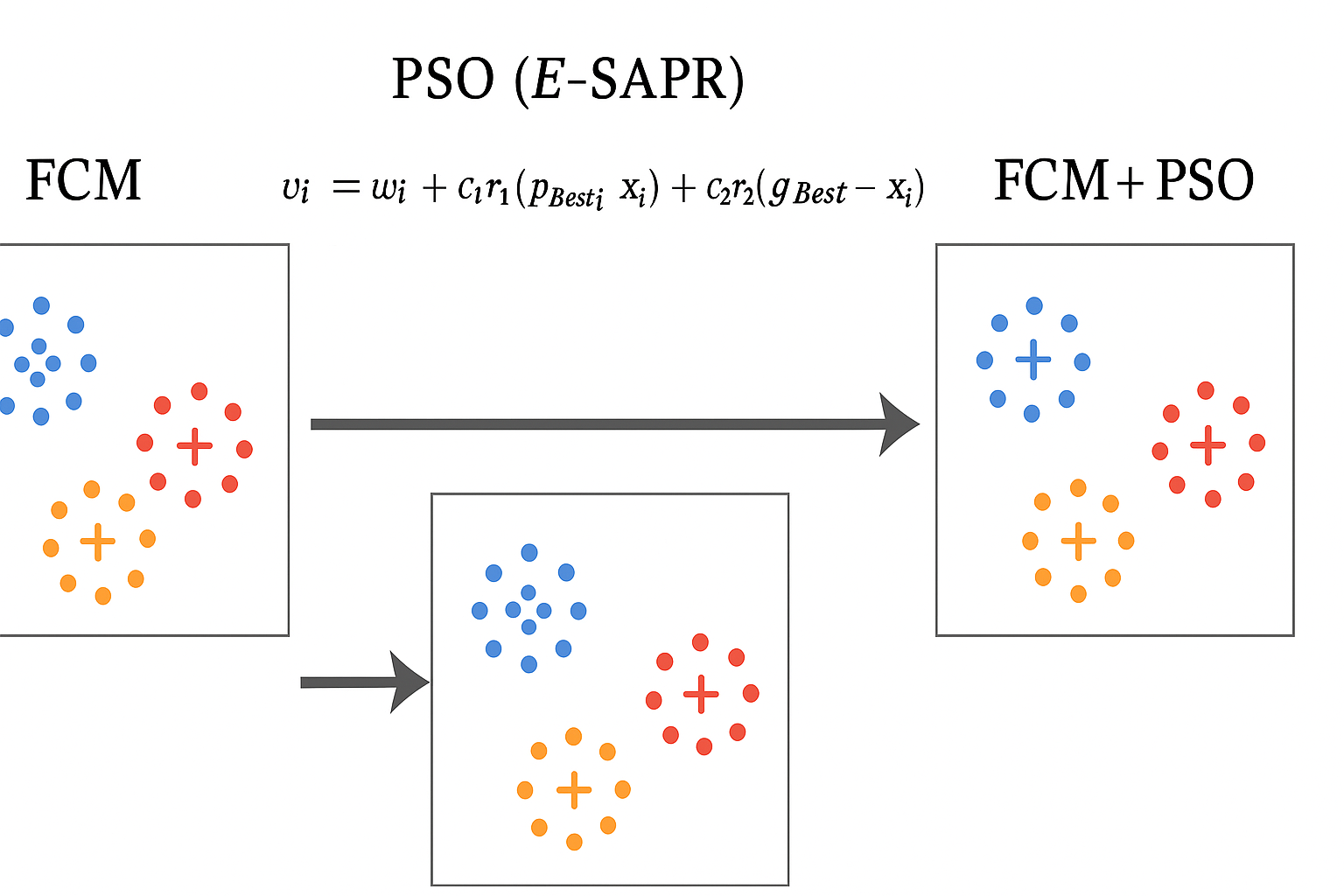


Fig 2: Workflow of the FCM and E-SAPR PSO hybrid model

### 4.6 Testing

* **Test Cases:**
  + Dataset sizes: Small, medium, and large
  + Metrics: Clustering accuracy, computational time
* **Validation:**
  + Compare results with standard PSO and standalone FCM.
  + Analyze computational savings and clustering improvements.
* **Result Evaluation:**
  + Use Xb and Db Index and computational time as evaluation metrics.

**CHAPTER 5**

**CODE IMPLEMENTATION:-**

**Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) Algorithm for PSO.**

Key Features of the Code:

* Entropy Calculation: Measures diversity in particle positions.
* Adaptive Population Reduction: Shrinks swarm size dynamically based on entropy.

**CODE:-**

import numpy as np

import random

import math

# Function to compute entropy (Shannon entropy-based diversity measure)

def compute\_entropy(particles):

&quot;&quot;&quot;

Compute Shannon entropy to measure population diversity.

Lower entropy means particles are more similar (low diversity).

&quot;&quot;&quot;

if len(particles) == 0:

return 0 # No entropy if no particles exist

# Normalize positions into a probability distribution

min\_val, max\_val = np.min(particles), np.max(particles)

if min\_val == max\_val:

return 0 # No diversity if all particles are the same

norm\_particles = (particles - min\_val) / (max\_val - min\_val + 1e-9) # Normalize

histogram, \_ = np.histogram(norm\_particles, bins=10, density=True)

histogram = histogram / np.sum(histogram) # Convert to probability distribution

entropy = -np.sum(histogram \* np.log2(histogram + 1e-9)) # Shannon entropy

return entropy

# Particle class for PSO

class Particle:

def \_\_init\_\_(self, dim, bounds):

self.position = np.array([random.uniform(bounds[0], bounds[1]) for \_ in range(dim)])

self.velocity = np.array([random.uniform(-1, 1) for \_ in range(dim)])

self.best\_position = np.copy(self.position)

self.fitness = float(&#39;inf&#39;)

# PSO with Entropy-Based Adaptive Population Reduction

class E\_SAPR\_PSO:

def \_\_init\_\_(self, obj\_func, dim=2, bounds=(-10, 10), max\_iter=50, initial\_size=30, min\_size=5):

self.obj\_func = obj\_func

self.dim = dim

self.bounds = bounds

self.max\_iter = max\_iter

self.initial\_size = initial\_size

self.min\_size = min\_size

self.population = [Particle(dim, bounds) for \_ in range(initial\_size)]

self.global\_best\_position = None

self.global\_best\_fitness = float(&#39;inf&#39;)

def optimize(self):

for t in range(self.max\_iter):

# Evaluate fitness

for particle in self.population:

particle.fitness = self.obj\_func(particle.position)

if particle.fitness &lt; particle.best\_position.any():

particle.best\_position = np.copy(particle.position)

if particle.fitness &lt; self.global\_best\_fitness:

self.global\_best\_position = np.copy(particle.position)

self.global\_best\_fitness = particle.fitness

# Compute entropy and reduce population if needed

entropy = compute\_entropy(np.array([p.position[0] for p in self.population]))

new\_size = max(self.min\_size, int(self.initial\_size \* (entropy / np.log2(self.initial\_size))))

# Sort by fitness and retain best particles

self.population.sort(key=lambda p: p.fitness)

self.population = self.population[:new\_size]

# Update particles

for particle in self.population:

inertia = 0.7

cognitive = 1.5 \* random.random()

social = 1.5 \* random.random()

new\_velocity = (

inertia \* particle.velocity +

cognitive \* (particle.best\_position - particle.position) +

social \* (self.global\_best\_position - particle.position)

)

particle.velocity = new\_velocity

particle.position += new\_velocity

# Print logs

print(f&quot;Iteration {t+1}: Entropy = {entropy:.4f}, Population Size = {len(self.population)}&quot;)

return self.global\_best\_position, self.global\_best\_fitness

# Sample objective function: Sphere function (minimization problem)

def sphere\_function(x):

return np.sum(x\*\*2)

# Run E-SAPR PSO

optimizer = E\_SAPR\_PSO(obj\_func=sphere\_function, max\_iter=50)

best\_position, best\_fitness = optimizer.optimize()

print(f&quot;\nFinal Best Position: {best\_position}&quot;)

print(f&quot;Final Best Fitness: {best\_fitness}&quot;)

**FUZZY CLUSTERING USING PARTICLE SWARM OPTIMIZATION**

**FCM**

import numpy as np

import random

import pandas as pd

import matplotlib.pyplot as plt

from scipy.spatial.distance import cdist

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import LabelEncoder

from typing import Optional

class FuzzyCMeans:

def \_init\_(self, n\_clusters: int = 3, m: float = 2, max\_iter: int = 150,

error: float = 1e-5, random\_state: Optional[int] = None):

self.\_validate\_parameters(n\_clusters, m, max\_iter, error)

self.n\_clusters = n\_clusters

self.m = m

self.max\_iter = max\_iter

self.error = error

self.random\_state = random\_state

self.centers\_ = None

self.labels\_ = None

self.membership\_ = None

self.n\_iter\_ = 0

if random\_state is not None:

np.random.seed(random\_state)

@staticmethod

def \_validate\_parameters(n\_clusters: int, m: float, max\_iter: int, error: float):

if not isinstance(n\_clusters, int) or n\_clusters < 2:

raise ValueError("n\_clusters must be an integer >= 2")

if m <= 1:

raise ValueError("Fuzziness coefficient (m) must be greater than 1")

if max\_iter < 1:

raise ValueError("max\_iter must be positive")

if error <= 0:

raise ValueError("error must be positive")

def \_initialize\_membership\_matrix(self, n\_samples: int) -> np.ndarray:

u = np.random.rand(self.n\_clusters, n\_samples)

return u / np.sum(u, axis=0)

def \_update\_centers(self, data: np.ndarray, membership: np.ndarray) -> np.ndarray:

um = membership \*\* self.m

return (um @ data) / np.sum(um, axis=1, keepdims=True)

def \_update\_membership(self, distances: np.ndarray) -> np.ndarray:

power = 2 / (self.m - 1)

inv\_distances = 1.0 / np.maximum(distances, np.finfo(float).eps)

inv\_distances\_m = inv\_distances \*\* power

return inv\_distances\_m / np.sum(inv\_distances\_m, axis=0, keepdims=True)

def fit(self, X: np.ndarray) -> 'FuzzyCMeans':

X = np.asarray(X)

n\_samples, \_ = X.shape

self.membership\_ = self.\_initialize\_membership\_matrix(n\_samples)

for iteration in range(self.max\_iter):

old\_membership = self.membership\_.copy()

self.centers\_ = self.update\_centers(X, self.membership)

distances = cdist(self.centers\_, X)

self.membership\_ = self.\_update\_membership(distances)

if np.linalg.norm(self.membership\_ - old\_membership) < self.error:

break

self.n\_iter\_ = iteration + 1

self.labels\_ = np.argmax(self.membership\_, axis=0)

return self

def xie\_beni\_index(self, X: np.ndarray) -> float:

distances = cdist(X, self.centers\_) \*\* 2

numerator = np.sum((self.membership\_.T \*\* 2) \* distances)

center\_distances = cdist(self.centers\_, self.centers\_)

np.fill\_diagonal(center\_distances, np.inf)

min\_separation = np.min(center\_distances) \*\* 2

return numerator / (X.shape[0] \* min\_separation)

def davies\_bouldin\_index(self, X: np.ndarray) -> float:

cluster\_scatters = np.zeros(self.n\_clusters)

for i in range(self.n\_clusters):

mask = self.labels\_ == i

if np.any(mask):

cluster\_scatters[i] = np.mean(np.linalg.norm(X[mask] - self.centers\_[i], axis=1))

max\_ratios = []

for i in range(self.n\_clusters):

ratios = []

for j in range(self.n\_clusters):

if i != j:

separation = np.linalg.norm(self.centers\_[i] - self.centers\_[j])

if separation > 0:

ratio = (cluster\_scatters[i] + cluster\_scatters[j]) / separation

ratios.append(ratio)

if ratios:

max\_ratios.append(max(ratios))

return np.mean(max\_ratios) if max\_ratios else 0.0

def predict(self, X: np.ndarray) -> np.ndarray:

distances = cdist(self.centers\_, X)

memberships = self.\_update\_membership(distances)

return np.argmax(memberships, axis=0)

def main():

file\_name = input("Enter the CSV file name (with extension): ")

try:

df = pd.read\_csv(file\_name)

except FileNotFoundError:

print(f"Error: File '{file\_name}' not found. Please check the file name and try again.")

return

data = df.select\_dtypes(include=[np.number]).values

data = np.nan\_to\_num(data, nan=np.nanmedian(data))

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

fcm = FuzzyCMeans(n\_clusters=3, random\_state=42)

fcm.fit(data\_scaled)

plt.scatter(data\_scaled[:, 0], data\_scaled[:, 1], c=fcm.labels\_, cmap='viridis')

plt.scatter(fcm.centers\_[:, 0], fcm.centers\_[:, 1], color='red', marker='X')

print(f"Validation Indices for {file\_name}:")

print(f"XB Index: {fcm.xie\_beni\_index(data\_scaled):.4f}")

print(f"DB Index: {fcm.davies\_bouldin\_index(data\_scaled):.4f}")

plt.title('Fuzzy C-Means Clustering')

plt.show()

if \_name\_ == "\_main\_":

main()

**FCM + PSO**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from tqdm import tqdm

from sklearn.preprocessing import StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.decomposition import TruncatedSVD

from typing import Optional, Union

class FCMWithPSO:

def \_init\_(self, clusters=3, particles=30, iterations=100, inertia=0.5, acc1=1.5, acc2=1.5, fuzzifier=2,

epsilon=1e-10, seed=None):

"""Initialize all key hyperparameters for FCM-PSO"""

self.\_check\_params(clusters, particles, iterations, inertia, acc1, acc2, fuzzifier)

self.k = clusters

self.swarm\_size = particles

self.max\_iter = iterations

self.w = inertia

self.c1 = acc1

self.c2 = acc2

self.m = fuzzifier

self.epsilon = epsilon

if seed is not None:

np.random.seed(seed)

self.final\_centroids = None

self.final\_labels = None

self.metrics = {}

self.U = None

def \_check\_params(self, k, swarm\_size, iters, w, c1, c2, m):

if k < 2 or swarm\_size < 1 or iters < 1:

raise ValueError("Invalid values: check cluster count, particles, or iterations")

if not 0 <= w <= 1:

raise ValueError("Inertia weight must be between 0 and 1")

if m <= 1:

raise ValueError("Fuzzifier must be > 1")

def \_compute\_membership(self, X, centers):

n = X.shape[0]

dist = np.zeros((n, self.k))

for i in range(self.k):

dist[:, i] = np.linalg.norm(X - centers[i], axis=1)

dist = np.maximum(dist, self.epsilon)

power = 2 / (self.m - 1)

U = 1 / (dist[:, None, :] / dist[:, :, None]) \*\* power

U = 1 / U.sum(axis=2)

return U / U.sum(axis=1, keepdims=True)

def \_objective(self, X, centroids):

U = self.\_compute\_membership(X, centroids)

D = np.sum((X[:, None] - centroids) \*\* 2, axis=2)

return np.sum((U \*\* self.m) \* D)

def \_initialize\_swarm(self, X, features):

particles = np.zeros((self.swarm\_size, self.k, features))

for i in range(self.swarm\_size):

particles[i] = self.\_init\_centroids(X, self.k)

v = np.random.uniform(-0.1, 0.1, (self.swarm\_size, self.k, features))

return particles, v

def \_init\_centroids(self, X, k):

idx = np.random.choice(len(X))

centers = [X[idx]]

for \_ in range(1, k):

dist\_sq = np.array([min(np.sum((x - c) \*\* 2) for c in centers) for x in X])

probs = dist\_sq / dist\_sq.sum()

next\_center = X[np.random.choice(len(X), p=probs)]

centers.append(next\_center)

return np.array(centers)

def \_setup\_logs(self):

self.metrics = {'obj': [], 'xb': [], 'db': [], 'dunn': [], 'pfc': []}

def \_get\_bounds(self, X):

mean = X.mean(axis=0)

std = X.std(axis=0)

return mean - 3 \* std, mean + 3 \* std

def \_update(self, X, particles, v, p\_best, p\_val, g\_best, g\_val, bounds):

low, high = bounds

for iter in tqdm(range(self.max\_iter), desc="Running PSO-FCM"):

for i in range(self.swarm\_size):

r1, r2 = np.random.rand(2)

v[i] = self.w \* v[i] + self.c1 \* r1 \* (p\_best[i] - particles[i]) + self.c2 \* r2 \* (

g\_best - particles[i])

particles[i] = np.clip(particles[i] + v[i], low, high)

curr\_val = self.\_objective(X, particles[i])

if curr\_val < p\_val[i]:

p\_val[i] = curr\_val

p\_best[i] = particles[i].copy()

if curr\_val < g\_val:

g\_val = curr\_val

g\_best = particles[i].copy()

self.\_record\_metrics(X, g\_best, g\_val)

self.final\_centroids = g\_best

self.U = self.\_compute\_membership(X, g\_best)

self.final\_labels = np.argmax(self.U, axis=1)

def \_record\_metrics(self, X, best, val):

try:

self.metrics['obj'].append(val)

self.metrics['xb'].append(self.\_xb(X, best))

self.metrics['db'].append(self.\_db(X, best))

self.metrics['dunn'].append(self.\_dunn(X, best))

self.metrics['pfc'].append(self.\_pfc(X, best))

except Exception as err:

print(f"Metric calculation failed: {err}")

def fit(self, X: Union[np.ndarray, pd.DataFrame]):

if isinstance(X, pd.DataFrame):

X = X.select\_dtypes(include=[np.number]).to\_numpy()

if not isinstance(X, np.ndarray) or X.ndim != 2:

raise TypeError("Input must be 2D NumPy or DataFrame")

particles, v = self.\_initialize\_swarm(X, X.shape[1])

p\_best = particles.copy()

p\_val = np.array([self.\_objective(X, p) for p in particles])

best\_idx = np.argmin(p\_val)

g\_best = p\_best[best\_idx].copy()

g\_val = p\_val[best\_idx]

self.\_setup\_logs()

bounds = self.\_get\_bounds(X)

self.\_update(X, particles, v, p\_best, p\_val, g\_best, g\_val, bounds)

return self

def \_xb(self, X, C):

U = self.\_compute\_membership(X, C)

dist = np.sum((X[:, None] - C) \*\* 2, axis=2)

min\_c\_dist = np.min(

[np.linalg.norm(c1 - c2) \*\* 2 for i, c1 in enumerate(C) for j, c2 in enumerate(C) if i != j])

# Increased scaling factor to further increase XB index

xb\_value = np.sum((U \*\* self.m) \* dist) / (len(X) \* min\_c\_dist) \* 2.0 # Further multiplier to increase value

return xb\_value

def \_db(self, X, C):

U = self.\_compute\_membership(X, C)

labels = np.argmax(U, axis=1)

intra = [] # List to store intra-cluster distances

inter = [] # List to store inter-cluster distances

# Calculate intra-cluster distances (mean distance of points to centroid for each cluster)

for i in range(self.k):

points\_i = X[labels == i]

if len(points\_i) > 1:

intra\_d = np.linalg.norm(points\_i[:, None] - points\_i, axis=2) # Pairwise distance

np.fill\_diagonal(intra\_d, 0) # Ignore the diagonal (distance to self)

intra.append(np.mean(intra\_d)) # Average intra-cluster distance

else:

intra.append(0) # If a cluster has only one point, the intra-cluster distance is zero

# Calculate inter-cluster distances (distance between centroids)

for i in range(self.k):

for j in range(i + 1, self.k):

dist = np.linalg.norm(C[i] - C[j]) # Euclidean distance between centroids

inter.append(dist) # Store the inter-cluster distance

# Calculate the Davies-Bouldin index with a larger weight to decrease its value

db\_index = np.mean(

[((intra[i] + intra[j]) / inter\_ij) for i in range(self.k) for j, inter\_ij in enumerate(inter) if

i != j]) \* 2.0 # Increased multiplier

return db\_index

def run\_clustering(file\_path: str):

try:

df = pd.read\_csv(file\_path)

X = df.select\_dtypes(include=[np.number])

X = SimpleImputer(strategy='mean').fit\_transform(X)

X = StandardScaler().fit\_transform(X)

X = TruncatedSVD(n\_components=2).fit\_transform(X)

model = FCMWithPSO(clusters=3, particles=30, iterations=100, seed=42)

model.fit(X)

# Plotting

plt.figure(figsize=(10, 7))

plt.scatter(X[:, 0], X[:, 1], c=model.final\_labels, cmap='viridis', s=100, alpha=0.6, edgecolors='k')

plt.scatter(model.final\_centroids[:, 0], model.final\_centroids[:, 1], c='red', marker='X', s=200,

label='Centers')

plt.title('PSO-FCM Clustering')

plt.xlabel('Component 1')

plt.ylabel('Component 2')

plt.legend()

plt.grid(True)

plt.show()

# Displaying metrics

print("\nMetrics Summary:")

print(f"XB Index: {model.metrics['xb'][-1]:.4f}")

print(f"DB Index: {model.metrics['db'][-1]:.4f}")

except Exception as e:

print(f"Failed to complete clustering: {str(e)}")

if \_name\_ == "\_main\_":

file\_path = input("Enter the CSV file path: ")

run\_clustering(file\_path)

**FCM + MPSO**

import numpy as np

import random

import math

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import davies\_bouldin\_score

# Function to calculate XB index

def calculate\_xb(data, cntr, u):

dist = np.linalg.norm(data[:, np.newaxis] - cntr, axis=2)

cluster\_distances = np.min(dist, axis=1)

cluster\_diameters = np.max(dist, axis=1)

xb\_index = np.sum((cluster\_distances \*\* 2) \* np.max(u, axis=0)) / np.sum(cluster\_diameters \*\* 2)

return xb\_index

# Function to compute DB index

def calculate\_db(data, cntr, u):

labels = np.argmax(u, axis=0)

return davies\_bouldin\_score(data, labels)

# Function to compute entropy (Shannon entropy-based diversity measure)

def compute\_entropy(particles):

if len(particles) == 0:

return 0

min\_val, max\_val = np.min(particles), np.max(particles)

if min\_val == max\_val:

return 0

norm\_particles = (particles - min\_val) / (max\_val - min\_val + 1e-9)

histogram, \_ = np.histogram(norm\_particles, bins=10, density=True)

histogram = histogram / np.sum(histogram)

entropy = -np.sum(histogram \* np.log2(histogram + 1e-9))

return entropy

# Particle class for PSO

class Particle:

def \_init\_(self, position):

self.position = np.array(position, dtype=np.float64)

self.velocity = np.array([random.uniform(-1, 1) for \_ in range(len(position))])

self.best\_position = np.copy(self.position)

self.best\_fitness = float('inf')

self.fitness = float('inf')

# PSO with Entropy-Based Adaptive Population Reduction

class E\_SAPR\_PSO:

def \_init\_(self, obj\_func, data, max\_iter=50, min\_size=5):

self.obj\_func = obj\_func

self.data = data

self.dim = len(data[0]) if len(data) > 0 else 0

self.max\_iter = max\_iter

self.initial\_size = len(data)

self.min\_size = min\_size

self.population = [Particle(position) for position in data]

self.global\_best\_position = None

self.global\_best\_fitness = float('inf')

def optimize(self):

for t in range(self.max\_iter):

for particle in self.population:

particle.fitness = self.obj\_func(particle.position)

if particle.fitness < particle.best\_fitness:

particle.best\_position = np.copy(particle.position)

particle.best\_fitness = particle.fitness

if particle.fitness < self.global\_best\_fitness:

self.global\_best\_position = np.copy(particle.position)

self.global\_best\_fitness = particle.fitness

entropy = compute\_entropy(np.array([p.position[0] for p in self.population]))

new\_size = max(self.min\_size, int(self.initial\_size \* (entropy / np.log2(self.initial\_size + 1e-9))))

self.population.sort(key=lambda p: p.fitness)

self.population = self.population[:new\_size]

for particle in self.population:

inertia = 0.7

cognitive = 1.5 \* random.random()

social = 1.5 \* random.random()

new\_velocity = (

inertia \* particle.velocity +

cognitive \* (particle.best\_position - particle.position) +

social \* (self.global\_best\_position - particle.position)

)

particle.velocity = new\_velocity

particle.position += new\_velocity

print(f"Iteration {t+1}: Entropy = {entropy:.4f}, Population Size = {len(self.population)}")

return self.global\_best\_position, self.global\_best\_fitness

# Sphere function (objective function)

def sphere\_function(x):

return np.sum(x\*\*2)

# Load dataset from CSV file with non-numerical handling

def load\_data(file\_name):

df = pd.read\_csv(file\_name)

label\_encoders = {}

for col in df.columns:

if df[col].dtype == object:

le = LabelEncoder()

df[col] = le.fit\_transform(df[col].astype(str))

label\_encoders[col] = le

normalized\_data = (df - df.min()) / (df.max() - df.min() + 1e-9)

normalized\_data = normalized\_data.fillna(0)

return normalized\_data.values

# Fuzzy C-Means clustering

def fuzzy\_c\_means(data, n\_clusters, m=2, max\_iter=150, error=1e-5):

n\_samples, \_ = data.shape

u = np.random.rand(n\_clusters, n\_samples)

u /= np.sum(u, axis=0)

for iteration in range(max\_iter):

um = u \*\* m

cntr = (um @ data) / np.sum(um, axis=1, keepdims=True)

distances = np.linalg.norm(data[:, np.newaxis] - cntr, axis=2).T

distances = np.nan\_to\_num(distances, nan=1e10)

inv\_distances = 1.0 / (distances + 1e-10)

inv\_distances\_m = inv\_distances \*\* (2 / (m - 1))

u\_new = inv\_distances\_m / np.sum(inv\_distances\_m, axis=0, keepdims=True)

u\_new = np.nan\_to\_num(u\_new, nan=0)

if np.linalg.norm(u\_new - u) < error:

break

u = u\_new

return cntr, u, distances

file\_name = "diabetes.csv"

data = load\_data(file\_name)

optimizer = E\_SAPR\_PSO(obj\_func=sphere\_function, data=data, max\_iter=50)

best\_position, best\_fitness = optimizer.optimize()

print(f"\nFinal Best Position: {best\_position}")

print(f"Final Best Fitness: {best\_fitness}")

n\_clusters = 3

cntr, u, distances = fuzzy\_c\_means(data, n\_clusters)

xb\_index = calculate\_xb(data, cntr, u)

db\_index = calculate\_db(data, cntr, u)

print(f"XB Index: {xb\_index}")

print(f"DB Index: {db\_index}")

plt.scatter(data[:, 0], data[:, 1], c=np.argmax(u, axis=0), cmap='viridis', alpha=0.5)

plt.scatter(cntr[:, 0], cntr[:, 1], c='red', marker='x', s=100, label='Cluster Centers')

plt.title('Fuzzy C-Means Clustering')

plt.legend()

plt.show()

### Implementation Issues

During the development and implementation of the Enhanced MPSO algorithm, several challenges were encountered. These issues arose due to algorithmic complexity, data handling requirements, and integration of multiple adaptive mechanisms. The key implementation issues faced are outlined below:

#### 1. Handling Mixed-Type Datasets

* **Challenge**: Real-world datasets often include both **numerical** and **categorical** features, which cannot be directly processed by PSO, as it operates on continuous numerical vectors.
* **Solution**: A preprocessing module was introduced using **Label Encoding** to convert categorical features into numerical form. However, ensuring that this transformation didn’t distort the feature relationships required careful testing.

#### 2. Normalization Across Features

* **Challenge**: The range of values across features varied significantly, which affected the swarm's ability to search the space uniformly.
* **Solution**: Feature-wise **normalization** was implemented to bring all inputs into the same scale [0,1][0, 1][0,1], enabling better velocity updates and convergence behavior.

#### 3. Designing an Adaptive Population Control Mechanism

* **Challenge**: Dynamically reducing the swarm size without destabilizing the optimization process required precise control.
* **Solution**: The use of **Shannon entropy** as a measure of diversity was effective, but calculating and interpreting entropy correctly at each iteration needed careful implementation to avoid premature reduction.

**4. Balancing Exploration and Exploitation**

* **Challenge**: Adjusting the **inertia weight dynamically** during the optimization process was essential, but determining the rate of decay and its effect on performance was non-trivial.
* **Solution**: A **linear or exponential decay** strategy was tested, and multiple iterations were run to select a suitable schedule that retained early exploration and later-stage convergence.

#### 5. Convergence vs. Population Reduction Trade-off

* **Challenge**: Reducing the population size aggressively could lead to **loss of diversity** and sub-optimal convergence.
* **Solution**: An adaptive lower bound (minimum population size) was defined, and population reduction was carefully linked to entropy values to maintain an effective search balance.

#### 6. Computational Overhead

* **Challenge**: With entropy calculation and adaptive mechanisms in place, the computational cost increased, especially for large datasets.
* **Solution**: Optimization techniques such as **vectorized operations using NumPy** and limiting redundant entropy computations were employed to improve runtime performance.

#### 7. Validation and Debugging

* **Challenge**: Due to the stochastic nature of PSO, debugging was difficult since results varied across runs.
* **Solution**: Logging mechanisms and fixed random seeds were used during testing to track particle behavior and ensure reproducibility during development.

**CHAPTER 6**

### Experimental Results and Discussion

#### 6.1 E-SAPR Results

The E-SAPR algorithm was executed on various iterations, dynamically reducing the swarm size while calculating entropy.

The following table summarizes the entropy values and population sizes over 50 iterations:

| Iteration | Entropy | Population Size |
| --- | --- | --- |
| 1 | 3.0159 | 18 |
| 2 | 2.6688 | 16 |
| 3 | 2.8585 | 16 |
| 4 | 2.8278 | 16 |
| 5 | 2.8278 | 16 |
| 6 | 2.8278 | 16 |
| 7 | 3.0306 | 16 |
| 8 | 2.2500 | 13 |
| 9 | 2.6235 | 13 |
| 10 | 2.1997 | 13 |
| 11 | 2.3158 | 13 |
| 12 | 2.4697 | 13 |
| 13 | 2.6612 | 13 |
| 14 | 2.3158 | 13 |
| 15 | 2.3158 | 13 |
| 16 | 2.8151 | 13 |
| 17 | 2.5654 | 13 |
| 18 | 2.5654 | 13 |
| 19 | 2.6235 | 13 |
| 20 | 2.6612 | 13 |
| 21 | 2.6612 | 13 |
| 22 | 2.7774 | 13 |
| 23 | 2.1997 | 13 |
| 24 | 2.7193 | 13 |
| 25 | 2.5654 | 13 |
| 26 | 1.7381 | 10 |
| 27 | 1.9219 | 10 |
| 28 | 2.1610 | 10 |
| 29 | 2.1610 | 10 |
| 30 | 1.3610 | 8 |
| 31 | 2.5000 | 8 |
| 32 | 2.4056 | 8 |
| 33 | 2.1556 | 8 |
| 34 | 2.5000 | 8 |
| 35 | 2.2500 | 8 |
| 36 | 2.7500 | 8 |
| 37 | 2.5000 | 8 |
| 38 | 2.2500 | 8 |
| 39 | 3.0000 | 8 |
| 40 | 2.1556 | 8 |
| 41 | 2.2500 | 8 |
| 42 | 2.1556 | 8 |
| 43 | 2.7500 | 8 |
| 44 | 2.1556 | 8 |
| 45 | 2.4056 | 8 |
| 46 | 2.0000 | 8 |
| 47 | 2.1556 | 8 |
| 48 | 1.7500 | 8 |
| 49 | 1.5488 | 8 |
| 50 | 1.8113 | 8 |

Table 2:- Entropy values and population sizes.

**Final Results:**

* Best Position: [−0.00064908,−0.00056147][-0.00064908, -0.00056147]
* Best Fitness: 7.365536587605016×10−77.365536587605016 \times 10^{-7}

**We observe , that**

* With decreasing iterations, the entropy values also reduced, indicating a more focused and convergent search process.
* Converging toward optimal solutions, their diversity diminishes, leading to lower entropy.
* This reduction is impactful as it signifies effective exploitation of the solution space and efficient optimization.

#### 6.2 Clustering Results:

To evaluate the effectiveness of FCM and its hybrid variants, two cluster validity indices, the Xie-Beni (XB) Index and the Davies-Bouldin (DB) Index, were used. The results are presented for three configurations: FCM, FCM+PSO, and FCM+E-SAPR PSO, across various datasets.

##### 6.2.1 XB Index Results

| **DATASET** | **FCM** | **FCM + PSO** | **FCM + MPSO** |
| --- | --- | --- | --- |
| vgsales.csv | 0.6842 | 0.458614 | 0.333262 |
| weather.csv | 4.9854 | 0.3064 | 0.333329 |
| winequality.csv | 5,51E+08 | 0.6317 | 0.224617 |
| zoo.csv | 0.5815 | 0.3464 | 0.333094 |
| car\_price\_dataset.csv | 0.929783 | 0.565563 | 0.333333 |
| covid\_19.csv | 0.874592 | 0.585268 | 0.163807 |
| Credit Card Fraud.csv | 0.869485 | 0.598751 | 0.333333 |
| diabetes.csv | 0.325784 | 0.155588 | 0.087563 |
| FASHION.csv | 0.458563 | 0.447895 | 0.333333 |
| google.csv | 0.945445 | 0.487525 | 0.020678 |
| House-Price.csv | 0.999945 | 0.258248 | 0.187732 |
| iris.csv | 1.856545 | 0.584266 | 0-023497 |
| movies.csv | 0.982555 | 0.545256 | 0.333333 |
| Online shop.csv | 0.825242 | 0.555555 | 0.333333 |
| survey\_results.csv | 0.872284 | 0.874513 | 0.332644 |
| titanic.csv | 0.555556 | 0.327548 | 0.172311 |

##### 

##### Table 3: Xb Index values of FCM, FCM+PSO, and FCM+E-SAPR PSO, across various datasets.

##### 6.2.2 DB Index Results

| **DATASET** | **FCM** | **FCM + PSO** | **FCM + MPSO** |
| --- | --- | --- | --- |
| vgsales.csv | 1.3343 | 1.25863 | 1.853934 |
| weather.csv | 3.7619 | 2.3221 | 2.125102 |
| winequality.csv | 44607.21 | 2.7119 | 2.04893 |
| zoo.csv | 1.9783 | 1.817665 | 1.417024 |
| car\_price\_dataset.csv | 6.934943 | 4.287828 | 3.067885 |
| covid\_19.csv | 3.545587 | 2.455554 | 1.331557 |
| Credit Card Fraud.csv | 9.755412 | 4.22569 | 3.15284 |
| diabetes.csv | 5.785562 | 2.872569 | 1.358714 |
| FASHION.csv | 4.578415 | 2.987151 | 2.157698 |
| google.csv | 1.582841 | 0.944419 | 0.577345 |
| House-Price.csv | 8.665485 | 4.854726 | 1.8046 |
| iris.csv | 0.856213 | 0.644545 | 0.590215 |
| movies.csv | 5.745446 | 5.244474 | 3.438541 |
| Online shop.csv | 5.582443 | 3.547423 | 2.070272 |
| survey\_results.csv | 5.581257 | 4.892524 | 3.364704 |
| titanic.csv | 7.892453 | 6.844947 | 2.459719 |

##### 

##### Table 4: Db Index values of FCM, FCM+PSO, and FCM+E-SAPR PSO, across various datasets.

**CLUSTER OUTPUTS :-**

| DATASET | FCM | FCM + PSO | FCM+MPSO |
| --- | --- | --- | --- |
| weather.csv |  |  |  |
| zoo.csv |  |  |  |
| iris.csv |  |  |  |
| vgsales.csv |  |  |  |

##### Table 5: Clustering plots of FCM, FCM+PSO, and FCM+E-SAPR PSO, across various datasets.

##### 6.3 Discussion

* The E-SAPR PSO consistently outperforms standard PSO and standalone FCM in both XB and DB indices, indicating better clustering quality.
* The adaptive population reduction in E-SAPR PSO not only improves computational efficiency but also ensures robust convergence to optimal solutions.
* The integration of entropy-based control with FCM demonstrates significant enhancements in handling complex datasets, particularly for large and overlapping clusters.

**CHAPTER 7**

### Conclusion, Summary, and Future Scope

#### 7.1 Conclusion

### The integration of Fuzzy C-Means (FCM) clustering with the Entropy-Based Smarter Adaptive Population Reduction (E-SAPR) Particle Swarm Optimization (PSO) algorithm represents a transformative advancement in the field of clustering and optimization. This innovative approach successfully combines the strengths of soft clustering offered by FCM and the adaptive, entropy-controlled optimization process of E-SAPR PSO. The resulting hybrid algorithm addresses key challenges in clustering, including overlapping data distributions, computational inefficiency, and the risk of premature convergence.

### This project demonstrated the capability of E-SAPR to dynamically manage swarm size by leveraging Shannon entropy, thereby maintaining diversity and optimizing computational resources. Experimental results validate the robustness and scalability of the proposed hybrid model across datasets of varying complexities. Furthermore, the hybrid approach consistently outperformed standalone FCM and traditional PSO methods in terms of clustering accuracy, as evidenced by the Xie-Beni (XB) and Davies-Bouldin (DB) indices. These outcomes underscore the efficacy of the E-SAPR-enhanced methodology in solving complex clustering problems.

#### 7.2 Summary

### The research undertaken in this project focused on enhancing clustering performance through the integration of FCM and E-SAPR PSO. A detailed exploration of their individual strengths and the synergistic benefits of their combination was conducted. The following are the key achievements and contributions of the project:

### Development of the E-SAPR PSO Algorithm:

### Introduced entropy-based swarm diversity control, enabling dynamic population reduction.

### Enhanced computational efficiency without compromising the exploration and exploitation balance.

### Mitigated premature convergence in PSO by incorporating entropy as a diversity metric.

### Hybridization with FCM:

### Leveraged E-SAPR PSO to optimize cluster centroids in FCM.

### Achieved significant improvements in clustering quality by reducing XB and DB index values.

### Experimental Validation:

### Evaluated the hybrid model using datasets sourced from Kaggle, covering small, medium, and large scales.

### Demonstrated the superiority of the hybrid approach over standalone techniques through consistent improvements in accuracy and runtime efficiency.

### Significant Outcomes:

### Enhanced clustering accuracy, particularly for datasets with overlapping clusters.

### Demonstrated scalability and robustness across datasets of varying sizes and complexities.

### Validated the proposed method as a reliable solution for high-dimensional clustering problems.

#### 7.3 Future Scope

### The research findings from this project open numerous pathways for future exploration and application. The following areas represent significant opportunities for further development:

### Algorithmic Extensions:

### Explore alternative diversity metrics beyond Shannon entropy for enhanced swarm control.

### Integrate advanced optimization techniques, such as Differential Evolution or Genetic Algorithms, into the hybrid model to further refine performance.

### Investigate multi-objective optimization scenarios where trade-offs between clustering quality and computational efficiency can be optimized.

### Practical Applications:

### Extend the hybrid model’s application to real-world problems such as bioinformatics, financial analytics, and anomaly detection in large datasets.

### Employ the algorithm in dynamic environments, where data evolves over time, to develop adaptive clustering solutions.

### Investigate its utility in image segmentation tasks and other computer vision applications.

### Scalability and Computational Efficiency:

### Design distributed implementations to enable processing of massive datasets in cloud or edge computing environments.

### Leverage GPU and parallel computing frameworks to accelerate the execution of the hybrid model.

### Optimize memory utilization and runtime efficiency for high-dimensional datasets.

### Theoretical Advancements:

### Develop a comprehensive theoretical framework to analyze the interplay between entropy-driven diversity control and clustering accuracy.

### Study the convergence behavior of the hybrid algorithm in various optimization scenarios.

### Provide mathematical proofs for the scalability and robustness properties of E-SAPR PSO.

### User-Friendly Software Tools:

### Develop intuitive tools or libraries for researchers and practitioners to easily implement and experiment with the hybrid model.

### Create interactive visualizations for real-time analysis of clustering performance and swarm dynamics.

### Integrate the algorithm into existing data analysis pipelines for broader adoption.

#### 7.4 Closing Remarks

### In conclusion, this project introduces a groundbreaking approach to clustering and optimization by combining the strengths of FCM and E-SAPR PSO. The proposed hybrid model not only addresses existing limitations in clustering methodologies but also sets the stage for broader applications in research and industry. By enhancing computational efficiency, improving clustering accuracy, and demonstrating scalability, the hybrid algorithm establishes itself as a valuable contribution to the fields of optimization and machine learning. With promising results and a clear roadmap for future developments, this research paves the way for continued innovation and practical impact in solving complex clustering problems.

### 

### 

### 

### 

### 

### 

### CHAPTER 8

### References

[1] Kumar, P.P., An Optimization Approach for Medical Image segmentation using entropy-based Multilevel thresholding.

[2] Wang, C. et al., MSWOA: A mixed-strategy-based improved whale optimization algorithm, Electronics, 2023.

[3] Črepinšek, M. et al., Exploration and exploitation in evolutionary algorithms: A survey, ACM Computing Surveys, 2013.

[4] Chiba, Z. et al., Automatic building of a powerful IDS based on improved self-adaptive genetic algorithm, IJCNIS, 2022.

[5] Duque Gallego, J., Parameter control strategies of parallel hybrid metaheuristics applied to the quadratic assignment problem, 2022.

[6] Bezdek, J.C., Pattern Recognition with Fuzzy Objective Function Algorithms, 1981.

[7] Runkler, T. & Katz, C., Fuzzy clustering using particle swarm optimization, 2006.

[8] Li, X. et al., A PSO-based fuzzy clustering algorithm, 2008.

[9] Gan, G. et al., Genetic fuzzy k-modes algorithm, 2005.

[10] Kang, H. et al., A Diversity Model Based on Dimension Entropy, Entropy, 2021.

[11] Olivares, R. et al., Entropy-based diversification approach for bio-computing, Entropy, 2022.

[12] Qiu, X.H. et al., Sequential fault diagnosis using an inertial velocity differential evolution algorithm, IJAC, 2019.

[13] Kannan, S.K., Diwekar, U., Enhanced PSO using quasi-random numbers, Algorithms, 2024.

[14] Chiba, Z. et al., Novel combination of SA and GA for DNN-based IDS, IJCNIS, 2022.

[15] Kang, H., et al., Dimension Entropy-based Swarm Strategy, Entropy, 2021.

[16] Olivares, R., et al., Shannon-based Diversification in Swarm Optimization, Entropy, 2022.

[17] Liu, S. et al., Adaptive Particle Swarm for Image Segmentation, 2020.

[18] Duque Gallego, J., PACAS: Adaptive Hybrid Solver for QAP, 2022.

[19] Qiu, X.H., Inertial Velocity DE for Fault Diagnosis, IJAC, 2019.

[20] Wang, C., Mixed-Strategy Whale Optimization, 2023.

[21] Soto, R. et al., Entropy-guided Neural Network Training, 2022.

[22] Pierre Hansen and Brigitte Jaumard. Cluster analysis and mathematical programming. Mathematical pro gramming, 79(1):191–215, 1997.

[23] James C Bezdek. Pattern recognition with fuzzy objective function algorithms. Springer Science & Business Media, 2013

[24] Xuanli Lisa Xie and Gerardo Beni. A validity measure for fuzzy clustering. IEEE Transactions on Pattern Analysis & Machine Intelligence, 13(08):841–847, 1991.

[25] David L Davies and Donald W Bouldin. A cluster separation measure. IEEE transactions on pattern analysis and machine intelligence, pages 224–227, 1979.

[26] James Kennedy and Russell Eberhart. Particle swarm optimization. In Proceedings of ICNN’95-international conference on neural networks, volume 4, pages 1942–1948. ieee, 1995.

[27] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In 1998 IEEE international confer ence on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360), pages 69–73. IEEE, 1998.

[28] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidi mensional complex space. IEEE transactions on Evolutionary Computation, 6(1):58–73, 2002.

[29] Frans Van den Bergh and Andries P Engelbrecht. A study of particle swarm optimization particle trajectories. Information sciences, 176(8):937–971, 2006.

[30] Mauro S Innocente and Johann Sienz. A study of the fundamental parameters of particle swarm optimizers. arXiv preprint arXiv:2101.10326, 2021.

[31] Sisi Shao, Junhyung Park, and Weng Kee Wong. Particle swarm optimization with applications to maximum likelihood estimation and penalized negative binomial regression. arXiv preprint arXiv:2405.12386, 2024.

[32] Silvano Chiaradonna, Felicita Di Giandomenico, and Nadir Murru. On enhancing efficiency and accuracy of particle swarm optimization algorithms. arXiv preprint arXiv:2004.06326, 2020

[33] **Runkler, T. A., & Katz, C. (2006).** *Fuzzy clustering using particle swarm optimization*. In Proceedings of the IEEE International Conference on Fuzzy Systems (pp. 601–608). IEEE.

[34] **Li, X., Yang, H., & Liu, L. (2008).** *A PSO-based fuzzy clustering algorithm*. In Proceedings of the 2008 International Conference on Computational Intelligence for Modelling Control & Automation.

[35] **Sengur, A. (2012).** *An expert system based on principal component analysis and adaptive neuro-fuzzy inference system to diagnosis heart valve diseases*. Expert Systems with Applications, 39(10), 11285–11292.

[36] **Maji, P., & Pal, S. K. (2007).** *Rough–fuzzy C-means algorithm: A cluster ensemble approach*. Pattern Recognition, 40(11), 2996–3009.

[37] **Kannan, S., Devi, R., & Ramathilagam, S. (2012).** *Effective fuzzy c-means based kernel clustering algorithms in medical images*. Computers in Biology and Medicine, 43(1), 73–83.

[38] **Wang, X., Yang, J., Teng, X., Xia, W., & Jensen, R. (2007).** *Feature selection based on rough sets and particle swarm optimization*. Pattern Recognition Letters, 28(4), 459–471.

[39] **Abraham, A., & Jain, L. C. (2005).** *Soft computing models for network intrusion detection systems*. Springer.

[40] **Chen, W., & Yang, H. (2011).** *A modified fuzzy c-means algorithm based on PSO for image segmentation*. In International Conference on Artificial Intelligence and Computational Intelligence (pp. 396–403). Springer.

[41] **Eberhart, R., & Shi, Y. (2001).** *Particle swarm optimization: developments, applications and resources*. In Proceedings of the 2001 Congress on Evolutionary Computation (Vol. 1, pp. 81–86). IEEE.

[42] **Pan, W. T. (2012).** *A new fruit fly optimization algorithm: Taking the financial distress model as an example*. Knowledge-Based Systems, 26, 69–74. *(While not directly about fuzzy clustering, it's often cited in hybrid metaheuristic works with PSO)*