Computational Intelligence

Clustering with CI/EC algorithms: Project documentation

Clustering-Based Customer Segmentation

كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

Project Name: Clustering-Based Customer Segmentation with CI/EC Algorithms

Team Information :

- Team number : 29
- Discussion Time : 12:40

	ID	Full Name [In Arabic]	Attendance [Handwritten Signature]	Final Grade
1	20210111	احمد محمد مقبول عبدالهادي		
2	20210281	حسام أحمد صلاح نور الدين		
3	20210121	احمد مصطفى سيد احمد		
4	20210023	احمد اسماعيل سالم		
5	20210597	عمر ايمن فاروق احمد		
6	20210098	احمد محمد بسيونى		

Table of Contents

About the problem

The system aims to perform (Clustring) customer segmentation using **evolutionary and nature-inspired algorithms** for clustering, offering an alternative to traditional clustering methods such as k-means.

Problem formulation

Clustring can be formulated in multiple ways

Optimization problem (single objective)

Goal : Partition a dataset into K clusters to minimize intra-cluster distances.

Given :

- Dataset $X = \{x_1, x_2, \dots, x_n\} \subset \mathbb{R}^d$
- K number of clusters

Find :

- Cluster centers $C = \{c_1, c_2, \dots, c_k\}$
- Assignment function $a: X \to \{1, \dots, K\}$

Objective function :

$$\min_{\{C,a\}} \sum_{i=1}^{n} \|x_i - c_a(x_i)\|^2$$

- This is the classic **K-means** formulation — a **single-objective continuous optimization problem**.

Multi-Objective Optimisation Problem

Clustering often involves conflicting goals, like:

- Compactness (low intra-cluster distance)
- Separation (high inter-cluster distance)

Formulation :

$$min\ f_1 = \sum_{i=1}^{n} \|x_i - c_a(x_i)\|^2$$

$$min\ f_2 = \frac{1}{k(k-1)} \sum_{i=1}^{n} \|c_i - c_j\|^2$$

Free Optimisation Problem

- The number of clusters K is not predefined.
- The solution includes discovering both the optimal number of clusters and the assignments

Formulation :

Let $K \in \mathbb{N},\ K \geq 2\ and$

$$\min_{\{K,C,a\}} [Intra\ Cluster\ Distance - \lambda\ .Inter\ Cluster\ \ Distance]$$

$$\min_{\{K,C,a\}} [\sum_{i=1}^{n} \sum_{j=1}^{K} y_{ik} \|x_i - c_k\|^2 - \ \lambda \sum_{k=1}^{K} \sum_{j=1, j\neq k}^{K} \|c_k - c_j\|^2]$$

- This becomes a model selection + optimisation hybrid.
- One such method is **clustering with adaptive K** using techniques such as **genetic algorithms** or **simulated annealing** to search for the best K that minimizes the objective function.
- Dynamic Adjustment: The number of clusters can change during the optimization process. For example, clusters that contain very few points may be merged, or new clusters may be introduced if the algorithm finds that a new cluster reduces the total distance or improves the separation.
-

Constrained Optimisation Problem

- Add hard or soft constraints to the problem.
- Examples
  - Clusters must have a minimum or maximum number of points
- Formulation :
  - Minimize intra-cluster distance:

$$\min_{\{C,a\}} \sum_{i=1}^{n} \|x_i - c_a(x_i)\|^2$$

  - Subject to:

$$L \leq |C_k| \leq U \quad \forall k \in \{1, \dots, K\}$$

Evaluation metrics for clustering

Silhouette Score

Measures how similar a data point is to its own cluster (cohesion) compared to other clusters (separation).

- Range -1 to 1
  - Closer to 1: better clustering
  - Around 0: overlapping clusters
  - Negative: wrong clustering

$$Silhouette\ Score = \frac{b - a}{\max(a, b)}$$

- a is the average distance between a point and all other points in its cluster (cohesion)
  - average intra-cluster distance
- b is the average distance between a point and all points in the next nearest cluster (separation)
  - average nearest-cluster distance

Calinski-Harabasz

Ratio of between-cluster dispersion to within-cluster dispersion

- Higher is better

$$CH = \frac{tr(B_k)}{tr(W_k)} \frac{n - k}{k - 1}$$

- n number of samples
- k number of clusters
- $tr()$ the trace of a square matrix and it's the sum of its diagonal elements
  - $tr(A) = \sum_{i=1}^{n} A_{ii}$
- $tr(B_k)$ trace of between-group dispersion matrix
  - total inter-cluster variance
  - Between-cluster scatter matrix B_k how far cluster centroids are from the overall mean
- $tr(W_k)$ trace of within-group dispersion matrix
  - total intra-cluster variance
  - Within-cluster scatter matrix W_k how compact the data is around its own cluster center
- Which means higher between-cluster variance and lower within-cluster variance → better clusters.

Inertia

$$Inertia = \sum_{i=1}^{n} \|x_i - c_a(x_i)\|^2$$

Datasets and preprocessing

We are using some datasets as a benchmark across all the implmented CI approaches.

Mall Customer Segmentation Data

This dataset contains information on 200 customers, including attributes like age, gender, annual income, and spending score. It's ideal for practicing clustering algorithms such as K-Means.

Source: https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python

Samples	**200**
Dimensionality	5
Features	CustomerID, Gender, Age, *Annual Income (k$),*Spending Score (1-100)

Customer Segmentation

An automobile company has plans to enter new markets with their existing products (P1, P2, P3, P4 and P5). After intensive market research, they've deduced that the behavior of new market is like their existing market

Source: https://www.kaggle.com/datasets/vetrirah/customer

Samples	**8068 (training)**
Dimensionality	11
Features	ID, Gender ,Ever_Married, Age, Graduated, Profession, Work_Experience, Spending_Score, Family_Size, Var_1, Segmentation(label)
Samples	2627 (testing)
Dimensionality	10
Features	ID, Gender ,Ever_Married, Age, Graduated, Profession, Work_Experience, Spending_Score, Family_Size, Var_1

Preprocessing

The preprocessing pipeline implemented in our code is designed to **systematically clean and transform raw datasets into a format suitable for machine learning algorithms**. It handles both numerical and categorical features through a combination of imputation, scaling, and encoding steps, ensuring that the data is consistent, complete, and fully numeric.

Dataset Loading and Initial Cleaning

The dataset is first loaded from a CSV file using pandas. High-uniqueness columns commonly **identifier fields** such as CustomerID or ID **are automatically dropped**, as they typically do not contribute meaningful information for pattern recognition and can negatively impact model performance.

Feature Type Detection

Features are categorized into numerical (e.g., Age, Annual Income, Spending Score) and categorical (e.g., Gender, Profession, Ever_Married) based on their data types.

Numerical Feature Transformation

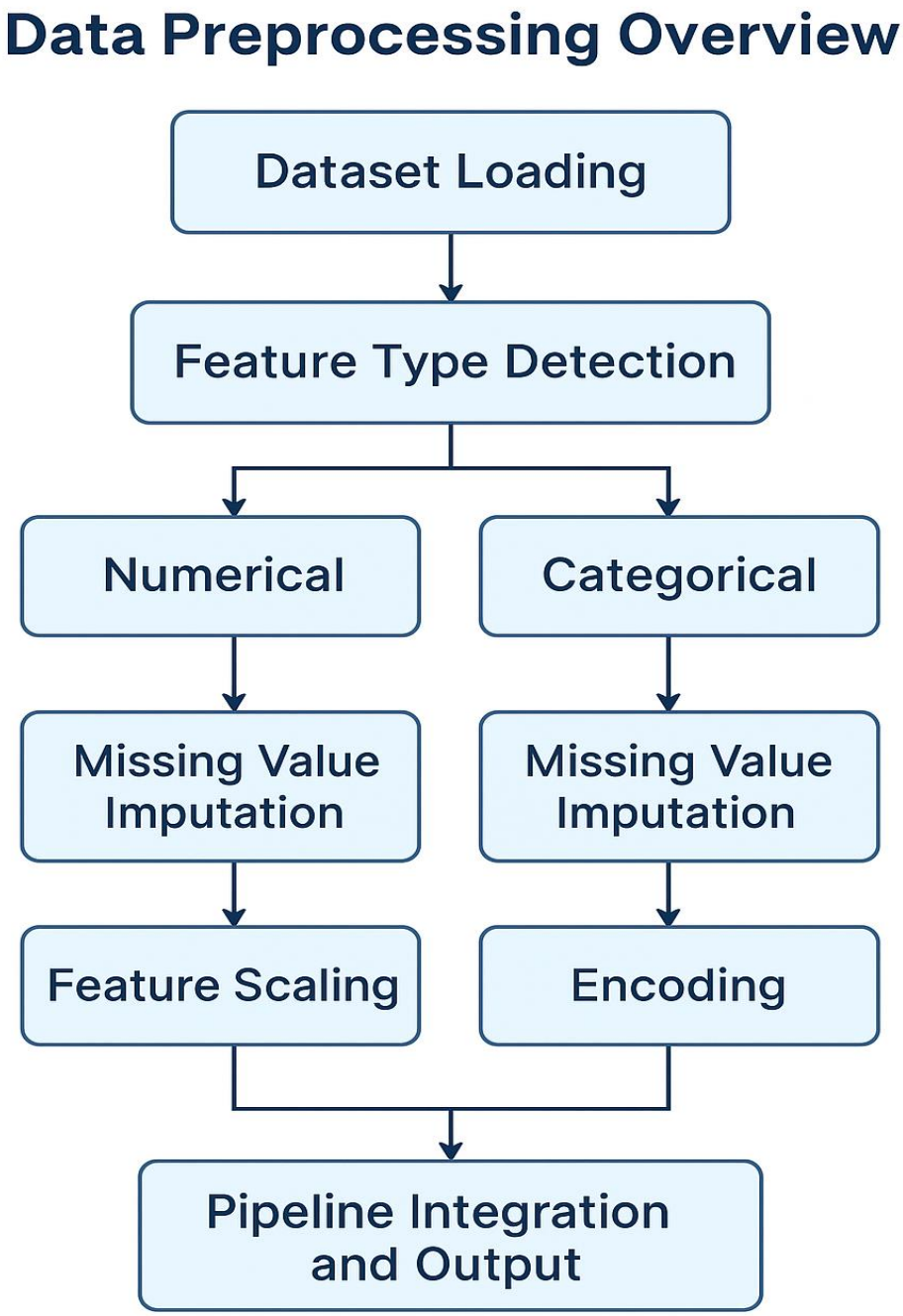
- **Missing Value Imputation:** Numerical columns with missing values are imputed using either the K-Nearest
- **Neighbors (KNN)** strategy for contextual completion or the **mean strategy** for a simpler statistical fill.
- **Feature Scaling:** To ensure numerical consistency, values are scaled using either StandardScaler (standardization with zero mean and unit variance) or MinMaxScaler (normalization to a [0, 1] range), based on user preference.

Categorical Feature Transformation

- Missing Value Imputation: Missing categorical values are filled using the most frequent category within each column.
- Encoding: Categorical variables are then converted to numerical form using a custom label encoding transformer, which applies a LabelEncoder instance to each categorical column independently, preserving the categorical relationships while making the data model-compatible.

Pipeline Integration and Output

- All transformations are encapsulated within a ColumnTransformer, allowing for parallel preprocessing of different feature types. The transformed dataset is returned as a structured DataFrame and optionally saved to a specified path for downstream tasks such as clustering or classification.


Data Preprocessing Overview
Dataset Loading
Feature Type Detection
Numerical
Categorical
Missing Value Imputation
Missing Value Imputation
Feature Scaling
Encoding
Pipeline Integration and Output

Literature review

Literature review of Differential Evolution

Title	**Automatic Clustering Using an Improved Differential Evolution Algorithm**
Year	2008
Authors	Swagatam Das, Ajith Abraham, Senior Member, IEEE, and Amit Konar, Member, IEEE
Keywords	Differential evolution (DE), genetic algorithms (GAs), particle swarm optimization (PSO), partitional clustering
Breif	This paper describes an application of DE to the automatic clustering of large unlabeled data sets. In contrast to most of the existing clustering techniques, the proposed algorithm requires no prior knowledge of the data to be classified. Rather, it determines the optimal number of partitions of the data "on the run." Superiority of the new method is demonstrated by comparing it with two recently developed partitional clustering techniques and one popular hierarchical clustering algorithm. The partitional clustering algorithms are based on two powerful well-known optimization algorithms, namely the genetic algorithm and the particle swarm optimization

Title	**Constrained optimization based on modified differential evolution algorithm**
Year	2012
Authors	Ali Wagdy Mohamed ,Hegazy Zaher Sabry
Keywords	Differential evolution (DE), genetic algorithms (GAs), partitional clustering, Constrained Opmiaztion, evolutionary algorithms
Breif	This paper presents a novel Constrained Optimization based on Modified Differential Evolution algorithm (COMDE). In the new algorithm, a new directed mutation rule, based on the weighted difference vector between the best and the worst individuals at a particular generation, is introduced. The new directed mutation rule is combined with the modified basic mutation strategy DE/rand/1/bin, where only one of the two mutation rules is applied with the probability of 0.5. The proposed mutation rule is shown to enhance the local search ability of the basic Differential Evolution (DE) and to get a better trade-off between convergence rate and robustness. Two new Constrained Optimization are introduced as uniform random variables to improve the diversity of the population and to bias the search direction. Additionally, a dynamic non-linear increased crossover probability is utilized to balance the global exploration and local exploitation. COMDE also includes a modified constraint handling technique based on feasibility and the sum of constraints violations. A new dynamic tolerance technique to handle equality constraints is also adopted.

Title	**Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization**
Year	2019
Authors	A. K. Qin ,V. L. Huang , P. N. Suganthan
Keywords	Differential evolution (DE), genetic algorithms (GAs), self adaptive ,
Breif	we propose a self-adaptive DE (SaDE) algorithm, in which both trial vector generation strategies and their associated control parameter values are gradually self-adapted by learning from their previous experiences in generating promising solutions. Consequently, a more suitable generation strategy along with its parameter settings can be determined adaptively to match different phases of the search process/evolution.

Title	**A Simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm**
Year	2004
Authors	DERVİŞ KARABOĞA SELÇUK ÖKDEM
Keywords	Differential evolution (DE), genetic algorithms (GAs), Cross over, Optmization Algorithm, Mutation
Breif	Differential Evolution (DE) algorithm is a new heuristic approach mainly having three advantages; finding the true global minimum regardless of the initial parameter values, fast convergence, and using few control parameters. DE algorithm is a population based algorithm like genetic algorithms using similar operators; crossover, mutation and selection.

What Is Differential Evolution?

Differential Evolution is an evolutionary algorithm that searches for the minimum of a real-valued fitness function $f: Rn \rightarrow R$ by evolving a population of NP candidate vectors over successive generations. Unlike gradient-based methods, DE treats the objective as a black box—only fitness evaluations are needed—making it suitable for non-differentiable, noisy or time-varying problems.

Core Algorithm

A basic DE algorithm consists of the following steps:

1. **Initialization**
   Generate an initial population $\{xi\}i = 1NP$ uniformly at random within the search bounds.
2. **Mutation**
   For each target vector xi , select three distinct vectors $a, b, c \neq x$ and create a mutant vector

   $$\boldsymbol{vi} = \boldsymbol{a} + \boldsymbol{F} \cdot (\boldsymbol{b} - \boldsymbol{c}),$$

   where the differential weight $F \in [0,2]$ controls amplification of the differential variation
3. **Crossover**
   Generate a trial vector ui by mixing components of

   $$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } r_j \leq CR \text{ or } j = R, \\ x_{i,j}, & \text{otherwise,} \end{cases}$$

   where $CR \in [0, 1]$ is the crossover probability, $r_j \sim U(0, 1)$, and R ensures at least one component is taken from $\mathbf{v}_i$
4. **Selection**

   Compare fitnesses: if $f(\mathbf{u}_i) \leq f(\mathbf{x}_i)$, then $\mathbf{x}_i \leftarrow \mathbf{u}_i$; otherwise keep $\mathbf{x}_i$
5. **Termination**
   Repeat mutation, crossover and selection until a stopping criterion is met (e.g., maximum generations or acceptable fitness).

Pseudo-Code

Initialize population X of size NP

While not termination_condition:

For each i in 1...NP:

Select a, b, c ∈ {1...NP}\{i} at random

v ← a + F*(b – c)

u ← crossover(v, x_i, CR)

if f(u) ≤ f(x_i):

x_i ← u

Return best vector in X

Control Parameters

The performance of DE hinges on three main parameters:

- **Population Size (NP):** often set to 10×n10\times n where nn is the problem dimensionality.
- **Differential Weight (F):** typical values range from 0.5 to 0.9; it scales the mutation step.
- **Crossover Probability (CR):** usually between 0.1 and 0.9, balancing exploration and exploitation.

Rules of thumb for parameter tuning were proposed by Storn et al. and further refined by Liu and Lampinen; mathematical convergence analyses have been conducted by Zaharie.

Mutation and Crossover Strategies

Beyond the basic **DE/rand/1** strategy above, other popular schemes include:

- **DE/best/1:** uses the current best vector as the base: $\mathbf{v} = \mathbf{x}_{\text{best}} + F(\mathbf{b} - \mathbf{c})$
- **DE/rand/2:** incorporates two differential pairs: $\mathbf{v} = \mathbf{a} + F(\mathbf{b} - \mathbf{c} + \mathbf{d} - \mathbf{e})$
- **Binomial vs. Exponential Crossover:** alternative crossover schemes can further diversify offspring generation.

Variants and Extensions

Researchers have developed many DE variants to enhance adaptability and convergence:

- **Adaptive DE (JADE, jDE):** parameters FF and CRCR are self-adapted during evolution.
- **Multi-objective DE (MODE):** extends DE to optimize vector-valued objectives.
- **Constrained DE:** employs penalty functions or feasibility rules to handle constraints.
- **Large-Scale DE:** specialized strategies for high-dimensional spaces (e.g., cooperative coevolution).

Applications

Due to its flexibility and simplicity, DE has been successfully applied in:

- **Engineering design:** antenna synthesis, structural optimization.
- **Trajectory planning:** European Space Agency uses DE for fuel-optimal spacecraft trajectories.
- **Machine learning:** hyperparameter tuning of non-differentiable models.
- **Control systems and signal processing:** parameter estimation, filter design.

Practical Considerations

- **Initialization:** ensure coverage of the search space—Latin hypercube sampling can outperform uniform random.
- **Parameter Tuning:** consider parameter sweeps or automated tuning frameworks to identify effective F, CR, NPF, CR, NP for your problem.
- **Stopping Criteria:** balance computational budget against solution quality; common criteria include maximum function evaluations or stagnation detection.

Proposed Modifications to Classical DE

a. Randomized Scaling Factor FF

In classical DE, F is usually fixed (e.g., between 0.4 and 1). This paper proposes:

$$F = 0.5 \cdot (1 + rand(0, 1))$$

- So F now varies **randomly between 0.5 and 1.0**
- This introduces **stochastic diversity** in the population
- Helps to avoid premature convergence and **improves exploration**

This variant is called **DERANDSF** (DE with Random Scale Factor)

b. Time-Varying Crossover Rate CrCr

Instead of using a constant crossover rate, they propose **decreasing it linearly over time**:

$$Cr = (Cr_{max} - Cr_{min}) \cdot \frac{MAXIT - iter}{MAXIT}$$

Where:

- Cr_max = 1.0 (initial value — full exploration)
- Cr_min = 0.5 (final value — focused exploitation)
- iter is the current iteration
- MAXIT is the maximum number of iterations

Why do this?

- Early on: Cr = 1.0 → new solutions inherit more from the mutation — **explore widely**
- Later on: Cr decreases → offspring retain more from parent — **refine (exploit)** current good areas

Why These Changes Help?

- **Random F:** Keeps the search diverse and avoids getting stuck in local optima.
- **Time-Varying Cr:** Smartly balances exploration (early) and exploitation (late).
- Both changes **improve the DE's convergence behavior**, making it more effective in tasks like automatic clustering.

Connection to Clustering

In **automatic clustering**, the DE algorithm searches for:

1. **The number of clusters**
2. **The optimal position of the cluster centroids**

By encoding both in the individual vectors, DE evolves better and better clustering solutions. The improved DE helps:

- Discover cluster structures
- Avoid being stuck in poor partitions
- Optimize both number and structure of clusters simultaneously

Representation

The chromosome in this context must:

- Represent **a set of candidate cluster centers**.
- Allow **adaptive selection** of how many clusters are used (up to KmaxK_{max}).
- Encode **cluster center positions** for use in clustering the data.

Chromosome Structure

Each chromosome is a **real-valued vector** of size:

$$K_{max} + K_{max} \times d$$

Where:

- K_max is the **maximum number of clusters** allowed.
- dd is the **dimension of the data** (e.g., if the data is 3D, then d=3d = 3).

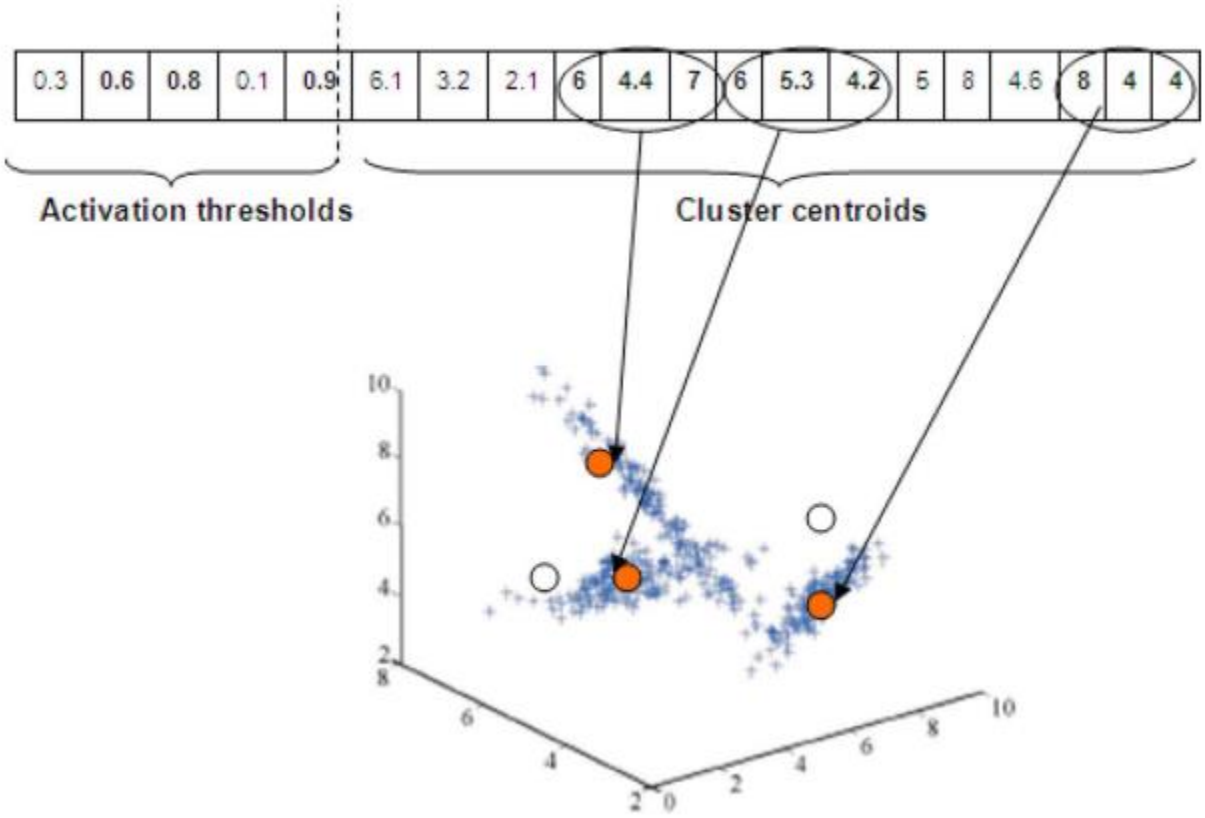
This vector can be divided into **two parts**:

1. **Activation vector** [Ti,1,Ti,2,...,Ti,Kmax]
    - Each value Ti,j∈[0,1]T_{i,j} \in [0, 1] indicates whether the j-th cluster is **active**.
    - Rule:

$$\text{If } T_{i,j} > 0.5 \Rightarrow \text{Cluster } j \text{ is active}$$

$$\text{If } T_{i,j} \leq 0.5 \Rightarrow \text{Cluster } j \text{ is inactive}$$

2. **Cluster center positions** [mi,1,mi,2,...,mi,Kmax]
    - Each mi,j is a **vector of length dd**, representing the **coordinates of cluster center j**.
    - This part of the chromosome is responsible for where clusters are placed in the feature space.



$$\vec{V}_i(t) = \underbrace{\boxed{T_{i,1}}\boxed{T_{i,2}}\boxed{\ldots}\boxed{T_i, K_{\max}}}_{\text{Activation Threshhold}}\underbrace{\boxed{\vec{m}_{i,1}}\boxed{\vec{m}_{i,2}}\boxed{\ldots}\boxed{\vec{m}_{i,K_{\max}}}}_{\text{Cluster Centroids}}$$

Mutation Handling

After DE creates a new chromosome using crossover/mutation, two special cases are handled:

1. **Activation value goes out of bounds**:
    - If Ti,j>1T_{i,j} > 1, set to 1.
    - If Ti,j<0T_{i,j} < 0, set to 0.
    - This **clamps** the value to [0, 1].
2. **No active clusters**:
    - If all activation values Ti,j≤0.5T_{i,j} \leq 0.5, meaning **zero clusters** would be selected,
    - Randomly choose **two indices**, and assign them **random values in [0.5, 1.0]** to ensure at least **two active clusters**.

This ensures:

- A **minimum of 2 clusters** per chromosome (to avoid trivial clustering).
- Sufficient **diversity** in cluster configurations during evolution.

Avoiding Erroneous Chromosomes

During clustering, **some chromosomes** (candidate solutions) might generate **invalid clusterings**. This can happen when:

- **A selected cluster center has no or very few data points assigned to it**.
- A cluster has **fewer than 2 points**, which can lead to:
    - **Division by zero** when computing intra-cluster distances or cluster validity indices.
    - **Meaningless evaluations** of clustering quality.

This usually happens when:

- A cluster center lies **outside the data distribution**, so **no points** get assigned to it during clustering.

Why Is This Dangerous?

If this happens, the computation of **CS** or **DB** index may fail:

- Could raise a runtime error (division by zero).
- Could lead to an incorrect fitness score.
- Could negatively impact the optimization process.

Validation & Repair Strategy:

1. **Check cluster sizes**:
    - If **any cluster** contains **fewer than 2 data points**, it's considered invalid.
2. **Repair the chromosome**:
    - Reinitialize the **cluster center positions** by:
        - Dividing the data into KK parts (where KK is the number of active clusters).

- Assigning about nK\frac{n}{K} data points to each cluster center.
- Each data point is assigned to its **nearest center**.
- Then, **recalculate** each center as the **average of the points assigned to it**.

This ensures:

- All clusters have at least two data points.
- No divisions by zero during evaluation.
- The chromosome becomes valid and usable for further processing.

Literature review for EA approach

Title	**An Improved Evolutionary Multi-Objective Clustering Algorithm Based on Autoencoder**
Year	2024
Authors	Mingxin Qiu, Yingyao Zhang, Shuai Lei, Miaosong Gu
Keywords	Evolutionary multi-objective clustering (EMOC), EMO-KC, autoencoder, dimensionality reduction, IEMO-KC, scaling factor, valid initialization, clustering accuracy, clustering validity, NSGA-II, centroid-based encoding.
Breif	The paper introduces AE-IEMOKC, a clustering algorithm that combines an autoencoder for dimensionality reduction with an improved EMO-KC. It addresses issues in accuracy, optimization (SSDexp), and invalid initializations. Using NSGA-II and a bi-objective model, AE-IEMOKC shows better performance on UCI datasets across metrics like ARI, ACC, SSD, and F1.

Title	**A Review of Evolutionary Multi-objective Clustering Approaches**
Year	2022
Authors	Cristina Y. Morimoto, Aurora Pozo, Shuai Lei, Marc´ılio C. P. de Souto
Keywords	Evolutionary multi-objective clustering (EMOC), multi-objective optimization, multi-objective evolutionary algorithms (MOEAs), clustering, Pareto front, objective functions, clustering validity indices (CVIs), crossover operators, mutation operators, solution selection, review, survey, architecture, applications.
Breif	The paper reviews Evolutionary Multi-objective Clustering (EMOC), mapping its research trends, key topics, and algorithm architectures. It categorizes EMOC methods, surveys objective functions, operators, and solution selection techniques, and outlines applications and future directions.

Title	**Multi-Objective Automatic Clustering Algorithm Based on Evolutionary Multi-Tasking Optimization**
Year	2024
Authors	Ying Wang, Kelin Dang, Rennong Yang, Leyan Li, Maoguo Gong
Keywords	evolutionary algorithm, multi-objective optimization, evolutionary multi-tasking,clustering algorithm, multi-tasking learning
Breif	The paper proposes MFMOCK, a multi-objective clustering algorithm based on evolutionary multi-tasking. It tackles limitations of traditional methods by allowing multiple clustering tasks to share knowledge through a unified population. Using overall deviation and connectivity index as

	objectives, MFMOCK applies multi-task adjacency coding and a relevance-based evolutionary operator to enhance performance. Tests on artificial and UCI datasets show improved accuracy and convergence over existing methods.

Title	**Genetic K-Means Algorithm**
Year	1999
Authors	K. Krishna, M. Narasimha Murty
Keywords	Clustering, genetic algorithms, global optimization, K-means algorithm, unsupervised learning
Breif	hybrid Genetic K-Means Algorithm that addresses K-means' tendency to get stuck in local minima. By combining the **global search** ability of Genetic Algorithms with the **fast convergence** of K-means (via a custom K-means operator), GKA achieves globally optimal clustering with high reliability. A **distance-based mutation** aids convergence while avoiding empty clusters. The method is theoretically proven to converge globally and outperforms K-means and other evolutionary methods in experiments.

Multiple Objective Optimization (MOO) : optimize multiple conflicting objectives, Instead of seeking a single optimal solution, MOO aims to find a set of compromise solutions known as the **Pareto front** which is a set of solutions that are non-dominated to each other but are superior to the rest of solutions in the search space.

These solutions represent the best trade-offs between the different objectives, where improving one objective would lead to the degradation of at least one other objective.

A Multi-objective Optimization Problem (MOP) involves minimizing or maximizing a vector of objective functions

$$F(\pi) = \left(f_1(\pi), f_2(\pi), \dots, f_z(\pi)\right)$$

Subject to some constraints.

Evolutionary Multi-objective Clustering (EMOC)

EMOC applies the principles of MOO to the problem of data clustering, Traditional clustering algorithms often optimize a single criterion, which might not be suitable for uncovering complex data structures that require considering multiple properties simultaneously.

EMOC addresses this by formulating clustering as a multi-objective optimization problem, where different clustering quality measures (like compactness and separation) are treated as separate objectives to be optimized concurrently.

By optimizing multiple objectives, EMOC can obtain a set of diverse and high-quality clustering solutions in a single run, offering different perspectives on the underlying data structure, In some EMOC algorithms, the number of clusters K itself can even be considered one of the multiple objectives .

Non-dominated Sorting is a ranking method used in Evolutionary Multi-Objective Optimization (EMOO) to classify solutions based on Pareto dominance, where

- A solution is Pareto-dominant if it's no worse than others in all objectives and better in at least one.
- Non-dominated solutions form the Pareto-optimal front, representing the best trade-offs across objectives.

The **non-dominated sorting process:**

1. Classifies all non-dominated individuals into Rank 1.
2. Removes Rank 1 solutions, then finds the next layer of non-dominated solutions → Rank 2.
3. Repeats until all individuals are ranked.

A general framwork for EMOC

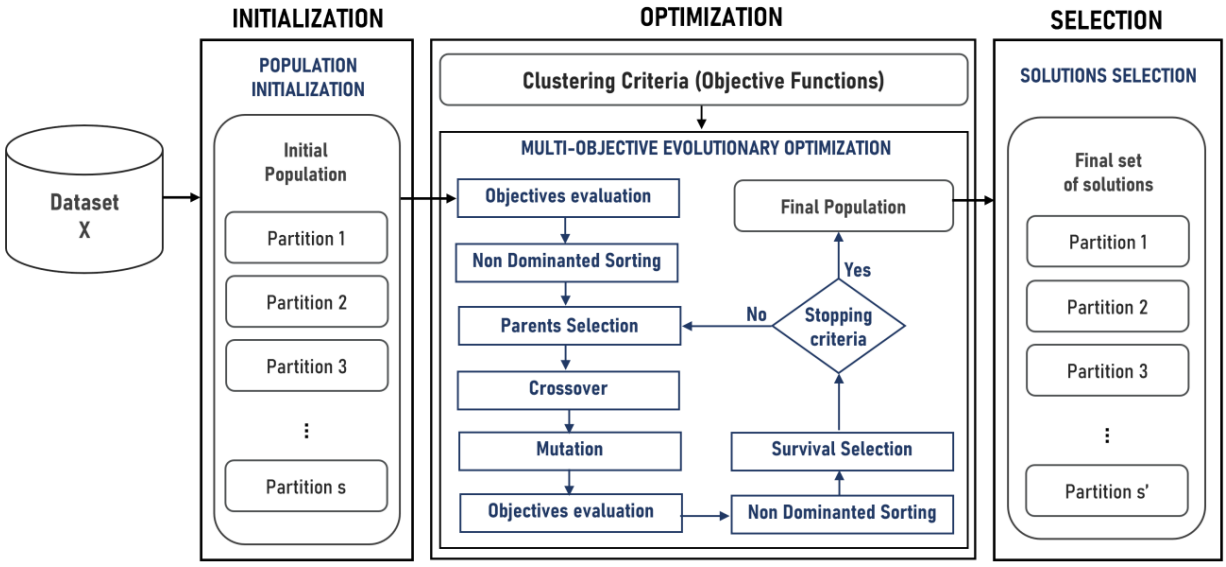


Figure 3: A general architecture of Evolutionary Multi-objective Clustering

How Genotypes are formed ?

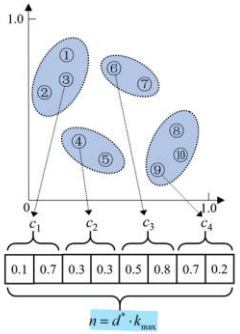
A chromosome or genotype in evolutionary algorithms (EA) is a set of parameters which define a proposed solution of the problem that the evolutionary algorithm is trying to solve.

Prototype-based Encoding	This method, used in centroid-based clustering, represents a clustering solution by the **coordinates of the cluster centers (centroids)**
Label-based Representation	Each gene in the chromosome corresponds to a data object, **and the allele value of the gene represents the cluster label assigned** to that object, The length of the chromosome is equal to **the number of data points in the dataset.**

Locus-based Adjacency Graph (LAG) Representation	This method represents a clustering as a graph, Each data point is represented by a gene (node) in the graph, and **the allele value of the gene defines a connection to another data point, Connected data points are considered to belong to the same cluster**, The number of clusters is determined automatically during the decoding step.
Binary Representation	Some approaches use a binary string as the chromosome. For example, **each chromosome might include n * k bits, where n is the number of data points and k is the number of potential clusters.** Every k bits can represent the cluster assignment of a data point.
Sparse-based Representation	This method considers an input as a linear combination of base elements from an over-complete dictionary to design the genotype.

Example on Prototype-based Encoding (Centroid-based)

- Each chromosome represents coordinates of cluster centroids.
- Assume 2D data (each point has two features), and we're allowing up to 3 clusters.
- Chromosome = [(2.1, 3.5), (7.2, 1.3), (4.0, 5.0)]
- Each tuple is a centroid.
- Or Chromosome = [**2.1, 3.5**, 7.2, 1.3, **4.0, 5.0**]
- The clustering algorithm will assign each data point to the nearest centroid.



Example on Label-based Representation

- Each gene corresponds to a data point, and its value is the cluster ID.
- Chromosome = [1, 2, 1, 0, 2]
- 5 genes (since we have 5 data points).
- Data point 0 → cluster 1, point 1 → cluster 2, etc.

Example on Locus-based Adjacency Graph (LAG)

- Each gene indicates which other data point it connects to, forming clusters as connected components.
- Chromosome = [1_0, 1_1, 0_2, 4_3, 3_4]
- Point (index) 0 connects to 1, point (index) 1 to 1 (self-loop), point (index) 2 to 0, etc.

- Graph edges: (0→1), (1→1), (2→0), (3→4), (4→3)
- Clusters from connected components: {0,1,2}, {3,4}

Example on Binary Representation

- Using a binary string to represent point-cluster assignments.
- Assume k = 3 clusters → need 5×3 = 15 bits.
- Chromosome = [1,0,0, 0,1,0, 0,0,1, 1,0,0, 0,1,0]
- Like one hot-encoding
- Point 0: [1,0,0] → Cluster 0
- Point 4: [0,1,0] → Cluster 1

Example on Sparse-based Representation

- **Each data point is represented as a combination of base elements (dictionary atoms).**
- **Assume 3 dictionary atoms.**
- **Each row is a data point (sum to 1).**
- **Chromosome = [**
  - **[0.8, 0.1, 0.1], # Point 0**
  - **[0.0, 1.0, 0.0], # Point 1**
  - **[0.5, 0.0, 0.5], # Point 2**
  - **[0.9, 0.1, 0.0], # Point 3**
  - **[0.2, 0.8, 0.0], # Point 4**
- **]**

Crossover

One-Point crossover	One crossover point is considered along the length of the parents' chromosomes, and the genes following the crossover point in one parent are swapped with the genes in the other parent
Two-Point crossover	Two crossover points along the length of the chromosome of each parent, such that the interval of genes between these two points are swapped
Shuffle crossover	This operator is similar to one-point crossover, in which a single crossover position is se lected, and before the variables are exchanged, they are randomly shuffled in both parents
Uniform crossover	for each position on the chromosome, a random decision is made on whether the swapping of genes should be done or not
K-Means Operator (KMO)	**one step of the K-means algorithm** applied to a solution string (chromosome), actually it's a search operator used instead of crossover in GKE to fast up the convergence.

Mutation

Simulated binary crossover (SBX)	this operator uses a probability density function that simulates the One Point Crossover in binary-coded representation
Polynomial mutation	a polynomial probability distribution is applied to perturb a solution
Uniform mutation	This operator is similar to one-point crossover, in which a single crossover position is se lected, and before the variables are exchanged, they are randomly shuffled in both parents
Uniform crossover	This operator replaces the value of the chosen particular slot position with a uniform random value selected considering a specified upper and lower bounds for that position
Distance based mutation (DBM)	This mutation operator changes the cluster assignment of a data point (an allele value) based on the **distances between the data point and the cluster centroids,** It is designed such that the probability of assigning a data point to a specific cluster is higher if that cluster's center is closer to the data point

Evolutionary Algorithms for clustring

Genetic K-means (GKA)	GA	
Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D)	MOEA	
MOEA with automatic k (MOCK)	MOEA	
Improved Evolutionary Multi-Objective Clustering Algorithm Based on Autoencoder (AE-IEMOKC)	MOEA	
Artificial Immune system-inspired clustering	CI	
Differential Evolution-based clustering	DE	
Simulated Annealing-based clustering	CI	
Non-dominated Sorting Genetic Algorithm (NSGA)	MOEA	**NSGA-II** is a popular and simple **multi-objective evolutionary algorithm** that ranks solutions using **non-domination sorting** and maintains diversity with **crowding distance**. It's widely used as a base optimizer in many multi-objective clustering methods.

Improved Evolutionary Multi-Objective Clustering Algorithm Based on Autoencoder (AE-IEMOKC)		

An Improved Evolutionary Multi-Objective Clustering Algorithm Based on Autoencoder.

Bi-objective model

- The algorithm simultaneously **minimizes two objectives**, leading to a **Pareto front** of diverse clustering solutions.

$$\min F(H) = \{f_1(H) = \mathrm{SSD}_{\exp},\quad f_2(H) = k\}$$

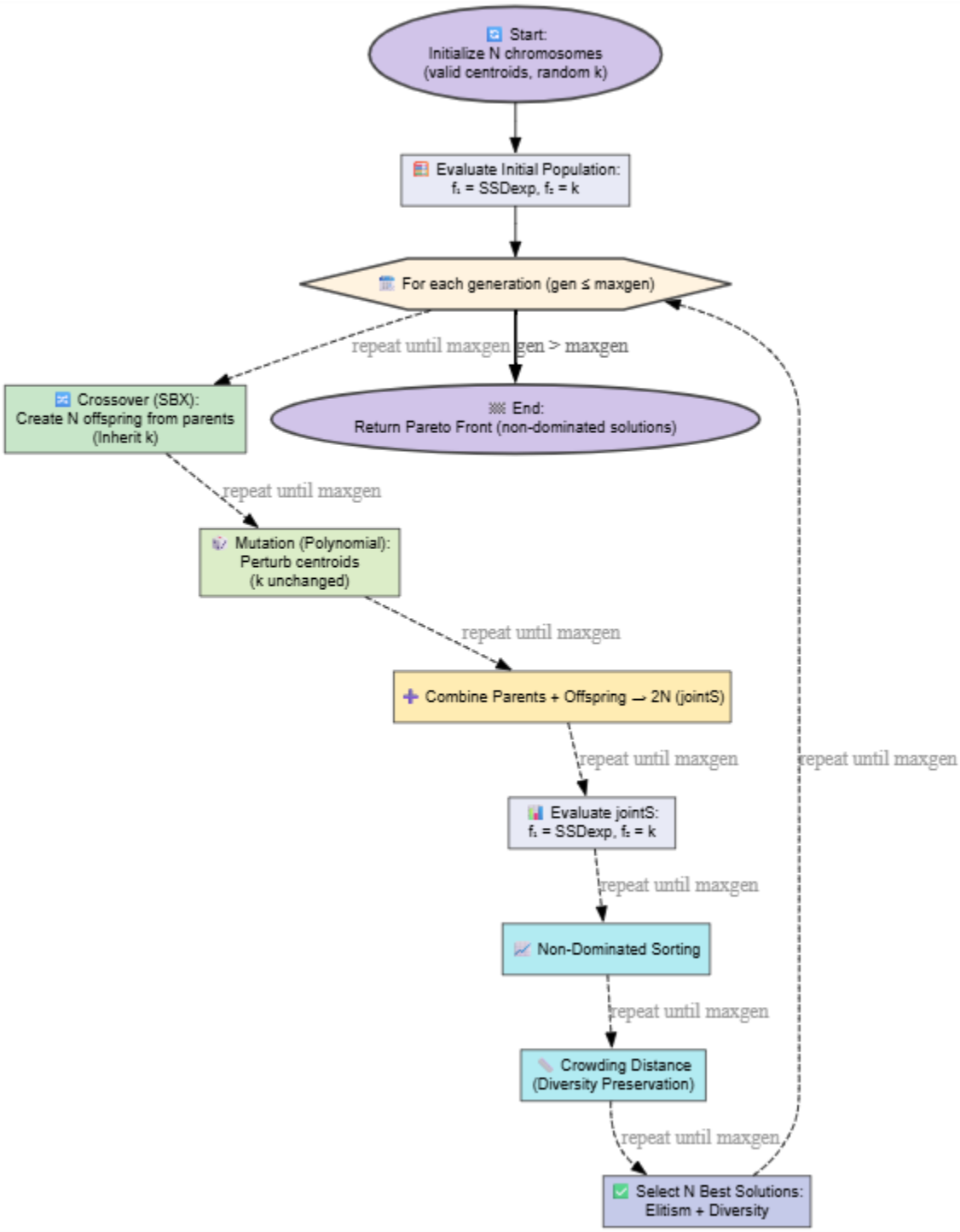
- H is the low-dimensional representation of the data (from the autoencoder)
- SSD_{exp} is a nonlinear transformation of the clustering compactness measure.
- k is the number of clusters.
- Objective 1: $f_1(H) = SSD_{exp}$ → Compact clusters

$$SSD_{exp} = (1 - e^{-\alpha.SSD}) - k$$

  - The SSD term measures the within-cluster scatter.
  - The exponential term normalizes SSD into the [0, 1) range
  - Subtracting kkk adds **pressure to reduce the number of clusters**, ensuring **conflict** with objective (no conflict = Redundant Objectives).
  - $SSD = \sum_{r=1}^{k} \|H_i - m_r\|^2$
    - m_r is the centroid of cluster r
    - H_i is a data point assigned to cluster r
    - α is a **scaling factor** $\boldsymbol{\alpha = \frac{10}{m * d^*}}$, where m the number of samples and d^* dimension of the embedding space (here 3)
    - α controls **how sensitive** the objective is to SSD
    - If α equals 1 (too large) or (too small) the $SSD_{exp} = (1 - 1 - k) = k$ (optimization collapses to a redundant objective function)
- Objective 2: $f_2(H) = k$ → Fewer clusters
  - Penalizes high numbers of clusters, aiming for simpler, more interpretable solutions.
- In multi-objective optimization, ensuring conflict between objectives is crucial to produce a diverse set of meaningful solutions.

NSGA-II Optimization

- **Non-dominated Sorting Genetic Algorithm**
- **Evolve a population of clustering solutions over generations to optimize two conflicting objectives**


Start:
Initialize N chromosomes
(valid centroids, random k)
Evaluate Initial Population:
f₁ = SSDexp, f₂ = k
For each generation (gen ≤ maxgen)
repeat until maxgen
gen > maxgen
Crossover (SBX):
Create N offspring from parents
(Inherit k)
End:
Return Pareto Front (non-dominated solutions)
repeat until maxgen
Mutation (Polynomial):
Perturb centroids
(k unchanged)
repeat until maxgen
Combine Parents + Offspring → 2N (jointS)
repeat until maxgen
repeat until maxgen
Evaluate jointS:
f₁ = SSDexp, f₂ = k
repeat until maxgen
Non-Dominated Sorting
repeat until maxgen
Crowding Distance
(Diversity Preservation)
repeat until maxgen
Select N Best Solutions:
Elitism + Diversity

NSGA

Autoencoder serves two purposes

1. **Dimensionality Reduction: Projects high-dimensional data into a lower-dimensional latent space while preserving essential features**
2. **Feature Learning: Learns representations that are more suitable for clustering**

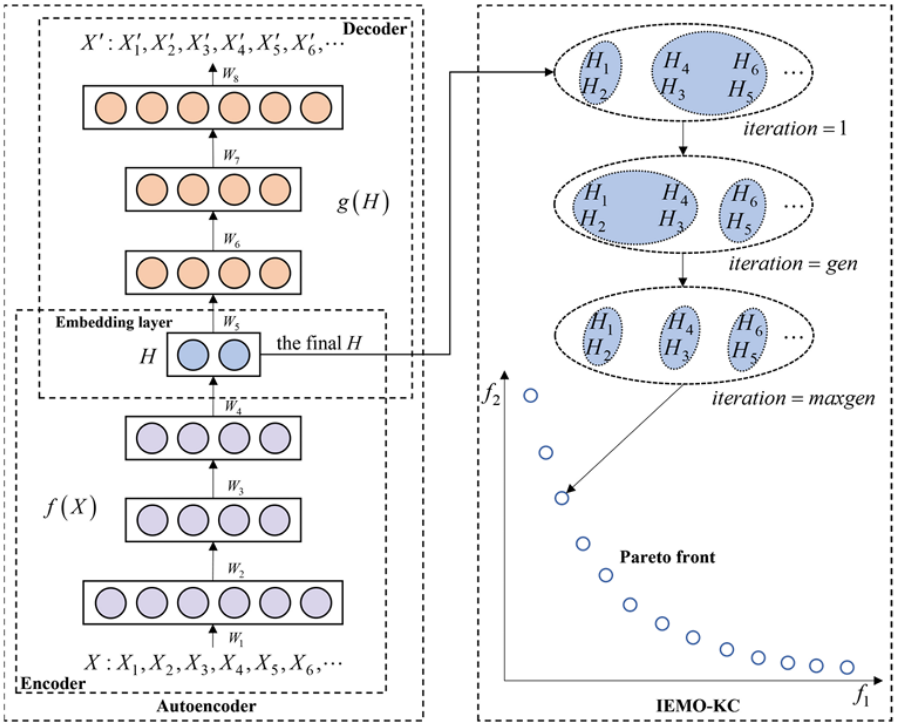
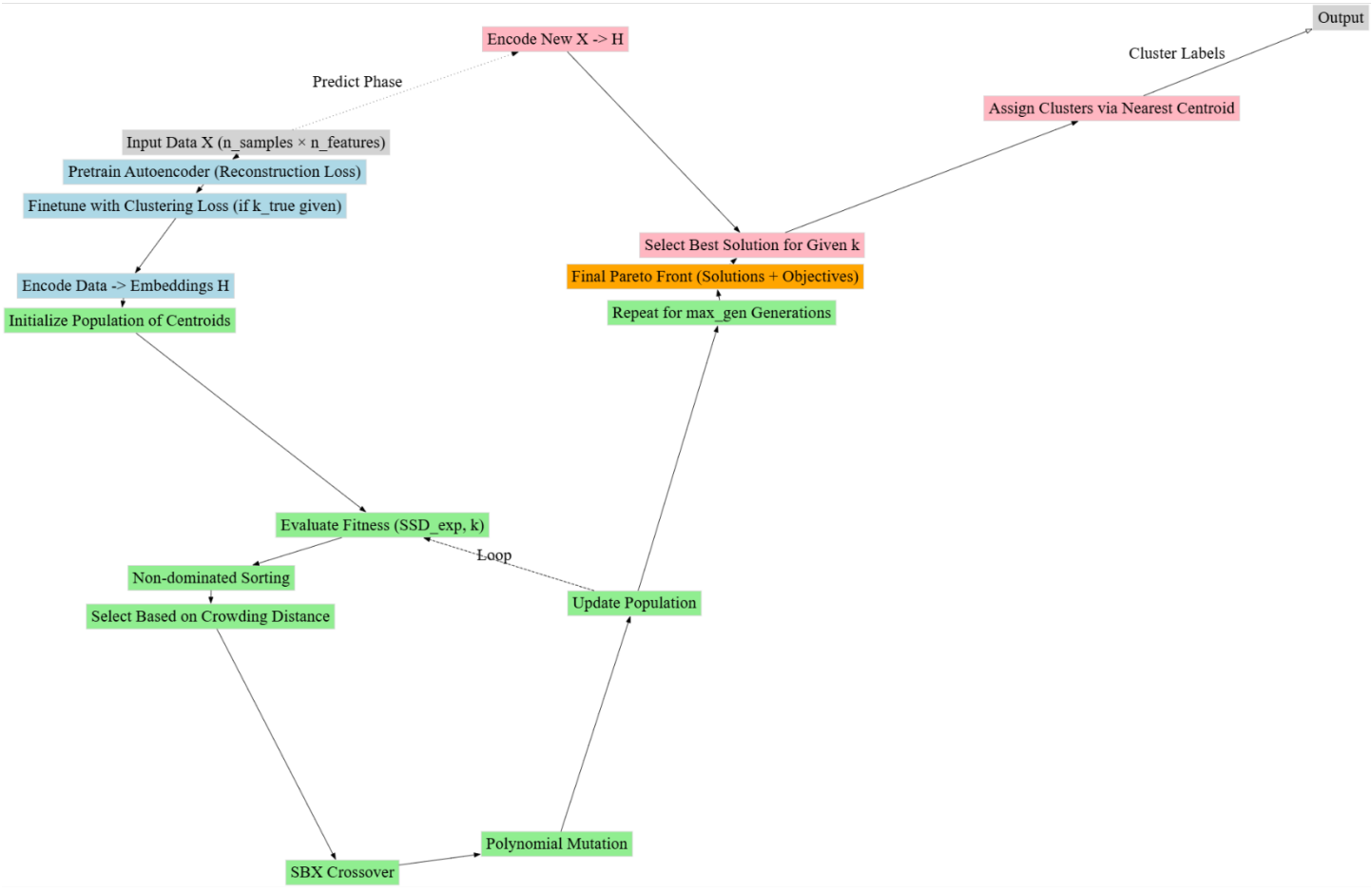


Figure 1. Architecture of the proposed AE-IEMOKC.

Loss Functions of AE

- The autoencoder uses a combined loss with two components
  - Reconstruction Loss
    - Ensure the embeddings preserve the original data's information
    - MSE between input and reconstructed data
    - $L_{rec}(X) = \frac{1}{m}\sum_{i=1}^{m}\left||X_i - X'_i|\right|^2$
  - Clustering Loss
    - $L_{cl}(H) = \alpha \sum_{i=1}^{k^*}\sum_{H_i \in C_r}\|H_i - m_r\|^2$
      - k^* actual number of clusters
      - Encourages compact clusters in latent space
      - Scaled version of within-cluster variance
  - Total loss
    - $L(X, H) = L_{rec(X)} + \lambda\, L_{cl}(H)$
    - λ balances the two objectives


Output
Cluster Labels
Encode New X -> H
Predict Phase
Assign Clusters via Nearest Centroid
Input Data X (n_samples × n_features)
Pretrain Autoencoder (Reconstruction Loss)
Finetune with Clustering Loss (if k_true given)
Select Best Solution for Given k
Final Pareto Front (Solutions + Objectives)
Encode Data -> Embeddings H
Repeat for max_gen Generations
Initialize Population of Centroids
Evaluate Fitness (SSD_exp, k)
Loop
Non-dominated Sorting
Update Population
Select Based on Crowding Distance
Polynomial Mutation
SBX Crossover

Literature review for Swarm Intellegence Approach

Title	**Augmented weighted K-means grey wolf optimizer: An enhanced metaheuristic algorithm for data clustering problems**
Year	2024
Authors	Manoharan Premkumar , Garima Sinha , Manjula Devi Ramasamy , Santhoshini Sahu ,Chithirala Bala Subramanyam , Ravichandran Sowmya , Laith Abualigah Bizuwork Derebew
Keywords	K-means Clustering-based Grey Wolf Optimizer (KCGWO), Grey Wolf Optimizer (GWO), K-means, Swarm Intelligence
Brief	The KCGWO algorithm is a hybrid optimization technique that combines the Grey Wolf Optimizer (GWO) with K-means clustering. It leverages the strengths of both GWO and K-means to improve the optimization process. The algorithm starts with a random population of wolves and iteratively updates their positions using a weighted average of the alpha, beta, and delta wolves. With a 50% probability, K-means clustering is applied to the wolf population to identify promising regions and guide the search. This hybrid approach enhances exploration, robustness, and flexibility, making KCGWO suitable for various optimization problems, including clustering and machine learning tasks.

Title	**Hybrid Ant Swarm-Based Data Clustering**
Year	2021
Authors	Md Ali Azam, Md Abir Hossen, Md Hafizur Rahman
Keywords	Hybrid Ant Colony Algorithm (hACA),Swarm Intelligence,Data Clustering,Ant-based Clustering,Local Density
Brief	The KCGWO algorithm is a hybrid optimization technique that combines the Grey Wolf Optimizer (GWO) with K-means clustering. It leverages the strengths of both GWO and K-means to improve the optimization process. The algorithm starts with a random population of wolves and iteratively updates their positions using a weighted average of the alpha, beta, and delta wolves. With a 50% probability, K-means clustering is applied to the wolf population to identify promising regions and guide the search. This hybrid approach enhances exploration, robustness, and flexibility, making KCGWO suitable for various optimization problems, including clustering and machine learning tasks.

K-means Clustering-based Grey Wolf Optimizer (KCGWO) Algorithm

The KCGWO algorithm is a novel optimization technique that integrates the Grey Wolf Optimizer (GWO) with K-means clustering. This hybrid approach aims to leverage the strengths of both GWO and K-means to improve the optimization process.

Grey Wolf Optimizer (GWO)

The GWO algorithm is a meta-heuristic optimization technique inspired by the hunting behavior of grey wolves. In GWO, the population of wolves is divided into four categories based on their fitness values:

1. **Alpha (α)**: The best solution found so far.
2. **Beta (β)**: The second-best solution.
3. **Delta (δ)**: The third-best solution.
4. **Omega (ω)**: The remaining wolves.

The GWO algorithm uses the following equations to update the positions of wolves:

1- Distance calculation:

$$\vec{D} = | \vec{C} \cdot \vec{Xp}(t) - \vec{X}(t) |$$

Where $\vec{Xp}(t)$ is the prey's position, $\vec{X}$(t) s the wolf's position and $\vec{C}$ is a coefficient vector.

2- Position update:

$$\vec{X}(t+1) = \vec{Xp}(t) - \vec{A} \cdot \vec{D}$$

where $\vec{A}$ is another coefficient vector.

3- Coefficient vectors:

$$\vec{A} = 2a \cdot \vec{r1} - a$$

where a is a parameter that decreases linearly from 2 to 0 over the iterations and $\vec{r1}$ and r2 are random vectors in the range $[0,1]$.

K-means Clustering

1. K-means is a widely used unsupervised clustering algorithm that partitions the data into K clust

ers based on their similarities. The K-means algorithm minimizes the sum of squared distances between the data points and their assigned cluster centroids.

The K-means objective function is given by:

$$J = \sum_{i=1}^{C} \sum_{x_j \in c_i} \|x_j - \mu_i\|^2$$

where C is the number of clusters, c_i is the i-th cluster, x_j is a data point, and μ_i is the centroid of the i-th cluster.

KCGWO Algorithm

The KCGWO algorithm combines the GWO and K-means clustering techniques. The algorithm starts by initializing a population of wolves randomly in the search space. Then, it applies the following steps:

2. **Evaluate fitness**: Evaluate the fitness of each wolf using the objective function.**Identify alpha, beta, and delta**: Identify the alpha, beta, and delta wolves based on their fitness values.

3. **K-means clustering**: With a probability of 0.5, apply K-means clustering to the wolf population and identify the centroids of the clusters.

4. **Update positions**: Update the positions of wolves using the weighted position update rule:

$$\vec{X}(t+1) = \frac{3\vec{X}_1 + 2\vec{X}_2 + \vec{X}_3}{6}$$

where $\vec{X}_1$, $\vec{X}_2$, $\vec{X}_3$ are the updated positions based on alpha, beta, and delta wolves, respectively.

The KCGWO algorithm repeats these steps until the termination criterion is met.

KCGWO Flow Chart

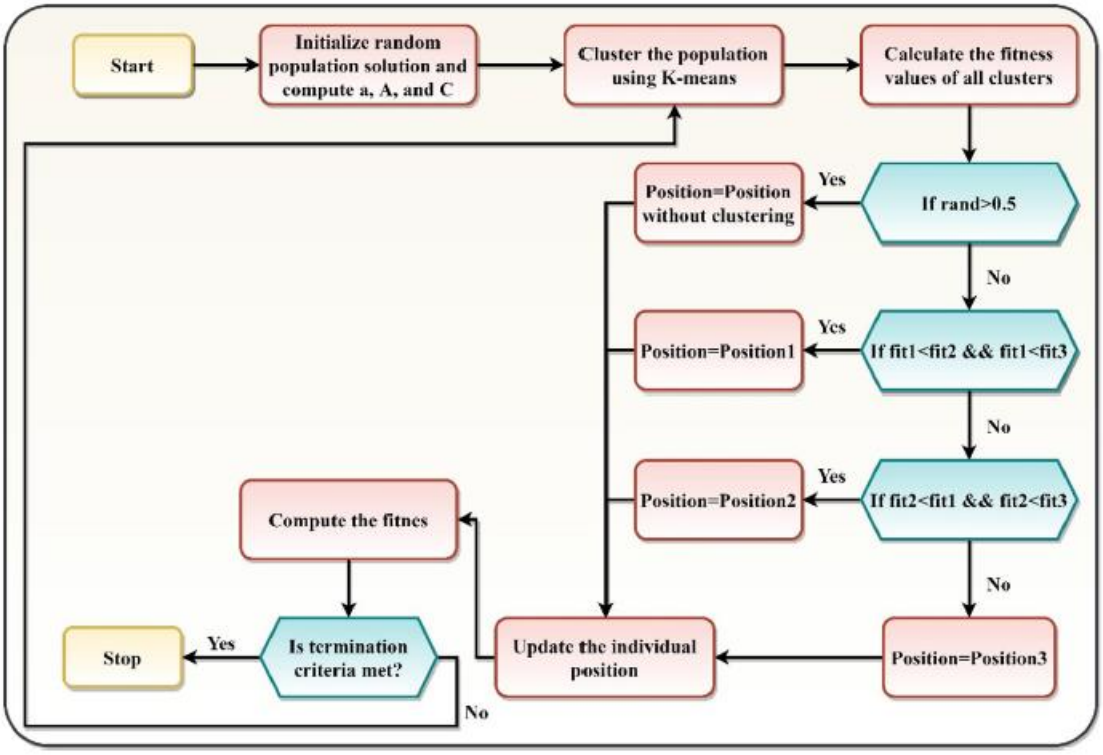


Figure 1. Flowchart of the proposed algorithm.

breakdown of every tunable parameter in KCGWO

1. Population Size (N or pop_size)
   • Definition: number of wolves (candidate solutions) exploring the space each iteration.
   • Effect of increasing N:
   – More animal scouts → better coverage of search space → higher chance to locate diverse basins of

attraction (exploration ↑)
– Cost: each iteration's fitness- and position-updates "N-times" → runtime ↑
• Effect of decreasing N:
– Faster per-iteration updates → less overall compute
– Risk of missing promising regions → exploration ↓, risk of premature convergence ↑

2. Max Iterations (MaxIter)
• Definition: total number of outer loops (t=1…MaxIter).
• Effect of increasing MaxIter:
– More "generations" of wolves → deeper exploitation over time → final solution quality ↑
– Runtime increases linearly
• Effect of decreasing MaxIter:
– Faster overall
– May terminate before fine-tuning centroids → exploitation ↓

3. Encircling Coefficient "a"
• Definition: linearly decreases from 2 → 0 as t goes from 1 → MaxIter.
• Role: controls magnitude of A, which in turn governs whether wolves diverge (explore) or converge (exploit).
• Exploration vs. Exploitation:
– Early (a close to 2): $A = 2 \cdot a \cdot r_1 - a$ can be >1 → wolves can overshoot the target or move outside → strong exploration
– Late (a close to 0): $A \in [-a, +a]$ shrinks → $|A|<1$ → wolves home in on α,β,δ → strong exploitation
• Tuning tip: slowing the decay of a (e.g. start at 2.5 or decay non-linearly) → exploration persists longer; speeding the decay → quicker local refinement (exploitation ↑ early but risk of local traps).

4. Random Vectors r_1 and r_2
• Definition: drawn fresh $\in [0,1]^D$ for each wolf & each leader (α,β,δ).
• Role: injects stochastic jitter into A and C.
• Exploration vs. Exploitation:
– Larger random fluctuations (r_1,r_2 near extremes) can boost exploration locally
– Not normally tuned (built-in randomness)

5. Coefficient Vectors A and C
• $A = 2 \cdot a \cdot r_1 - a$; $C = 2 \cdot r_2$
• Role:
– A dictates step-size & direction (sign flips allow pursuit or retreat)
– C scales the perceived distance to the guided positions
• Changing their generation (e.g. $C = c\_const \cdot r_2$) could bias wolves closer or farther from leaders

6. K-means Clustering Probability (p_km)
• Definition: fraction of iterations (default 0.5) where you run K-means on the wolf population to re-define α,β,δ from cluster centroids rather than individual best wolves.
• Effect of increasing p_km:
– More frequent group-wise guidance → wolves jump toward "collective" centroids → exploitation ↑ (stronger global pull)

– Potentially faster convergence but risk of over-commitment to early cluster structure
• Effect of decreasing p_km:
– Relies more on individual wolf fitness → retains diversity → exploration ↑

7. Number of Sub-clusters in Wolf-Clustering (K_wolf = 3)
   • Definition: the number of clusters you run in K-means on the N wolves. Paper fixes this at 3 to provide 3 guiding centroids.
   • Role: controls granularity of "group wisdom."
   • Changing K_wolf:
   – K_wolf>3 → more guiding centroids → could fragment direction signals (exploration ↑)
   – K_wolf<3 → fewer leaders → coarser guidance (exploitation ↑ but less nuance)

8. Weights on Alpha/Beta/Delta (w_1=3, w_2=2, w_3=1 in Eq.15)
   • Definition: in the position update
   $X(t+1) = (w_1X_1 + w_2X_2 + w_3X_3)/(w_1+w_2+w_3)$
   • Role: biases the new position more heavily toward the "best" leader (α) than β or δ.
   • Effects:
   – Increasing w_1 relative to w_2,w_3 → stronger pull to α → exploitation ↑ (faster convergence but risk of local trap)
   – More balanced weights (e.g. [1,1,1]) → more democratic movement → exploration ↑

9. Dimensionality D = n_clusters·n_features
   • Definition: length of each wolf's position vector (all centroids concatenated).
   • Effect:
   – Higher dimensionality → search space grows exponentially → need larger N or more iterations for same coverage
   – Lower dimensionality → easier to explore fully

   Summary:

- To boost **exploration** (broader search) ⇒
  increase pop_size, decrease p_km, slow the decay of a (or start a higher), reduce α-weight w_1, use more wolf-clusters K_wolf.
- To boost **exploitation** (tighter local search) ⇒
  increase MaxIter, increase p_km, speed up decay of a, increase w_1:w_2:w_3 ratio, lower pop_size (fewer, sharper scouts)

Hybrid Ant Clustering Algorithm (hACA)

Overview

The Hybrid Ant Clustering Algorithm (hACA) is an unsupervised clustering technique that extends the standard Ant Clustering Algorithm (ACA). It is inspired by the collective behavior of ants in nature, specifically their ability to sort items or brood. The hACA aims to enhance the efficiency and speed of the clustering process by integrating a **Genetic Algorithm (GA)** to guide ant movement, distinguishing it from the traditional ACA's typically localized random walks.

Core Components of Ant-Based Clustering

The foundational Ant Clustering Algorithm (ACA) relies on several key concepts:

1. **The Grid World:**

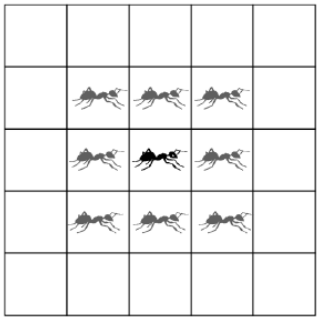


Fig. 1. A bi-dimensional grid with 5×5 cells

   - Data objects are placed within a bi-dimensional grid, typically of **m × m** cells. This grid serves as the environment where ants operate and objects are sorted.

2. **Artificial Ants:**
   - A population of N artificial ants moves within this grid.
   - Each ant can be unloaded (not carrying an object) or loaded (carrying a data object).
   - Ants interact indirectly by modifying the arrangement of objects on the grid.

3. **Data Objects:**
   - These are the individual items (data points) that need to be grouped into clusters.
   - Initially, these objects are randomly distributed across the grid cells.
4. **Local Density Perception (f(xi)):**
   - An ant **i** at a specific cell **c** (location **xi**) perceives the "density" of objects in its immediate neighborhood. This neighborhood is defined by an **s × s** region of cells around the ant.
   - The local density **f(xi)** for the **i**-th ant at cell **c** is calculated using **Equation (3)** from the paper:

$$f(x_i) = \sum_{n=1}^{s^2-1} N(j)M(j)$$

     Where:

- **xi** is the current location of the **i**-th ant.
- The sum is over the **s^2 - 1** neighboring cells (excluding the ant's own cell).
- **M(j) = 1** if an object is present in the **j**-th neighboring cell, and **0** otherwise.
- **N(j)** is described as representing a binary set of neighboring cells (likely related to similarity or just presence, often simplified to **N(j)=1** if **M(j)=1**). Essentially, **f(xi)** quantifies how "crowded" the ant's local area is with other objects

5. **Probabilistic Pick-up and Drop-off Rules:**
   - Ants pick up or drop objects based on probabilistic rules that depend on the local density **f(xi)**
   - **Pick-up Probability (Pp(xi)):** If an ant is unloaded and its current cell **xi** contains an object, it will pick up the object with a probability defined by **Equation (1) / (4)**:

$$P_p(x_i) = \left(\frac{k1}{k1 + f(x_i)}\right)^2$$

   Where **k1** is a threshold constant. A high **Pp(xi)** occurs when **f(xi)** is low (sparse area), encouraging ants to pick up isolated objects.
   - **Drop-off Probability (Pd(xi)):** If an ant is loaded and its current cell **xi** is empty (or a suitable drop site), it will drop the object with a probability defined by **Equation (2) / (5)**:

$$P_d(x_i) = \left(\frac{f(x_i)}{k2 + f(x_i)}\right)^2$$

   Where **k2** is a threshold constant. A high **Pd(xi)** occurs when **f(xi)** is high (dense area), encouraging ants to deposit objects in already clustered regions.

Genetic Algorithm (GA) for Enhanced Ant Movement

The "Hybrid" aspect of hACA comes from replacing the standard ACA's simple, localized ant movement (e.g., moving to one of 8 adjacent cells) with a movement strategy determined by a Genetic Algorithm. This allows for more intelligent, potentially long-range jumps, aiming for faster clustering.

1. **Chromosome Representation:**

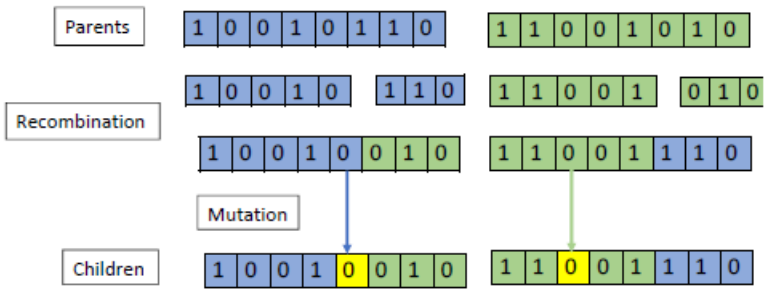


Fig. 2. Genetic algorithm applied in hACA

- An ant's location **(a, b)** (row **a**, column **b**) in the grid is represented as a binary string.
- The length of the binary string is determined by the grid dimensions (e.g., if the grid is **m × m**, **ceil(log2(m))** bits are needed for each coordinate).
- Figure 2 in the paper illustrates this with parent chromosomes like **10010110** and **11001010**.

2. **Genetic Operators:**

    - **Recombination (Crossover):** Standard crossover operations (e.g., single-point crossover, as depicted in Figure 2) are applied to parent chromosomes (e.g., current ant's location and another selected location) to generate offspring chromosomes, representing new potential locations.

        - *Figure 2 Example:*

            - Parent 1: **10010 | 110**
            - Parent 2: **11001 | 010**
            - Child 1 (after crossover): **10010010**

    - **Mutation:** Bits in the offspring chromosomes are flipped with a small probability, introducing diversity and preventing premature convergence.

        - *Figure 2 Example:*

            - Child **100100010** becomes **100101010** after mutation.

3. **Movement Decision:**

    - The GA produces one or more candidate new locations.
    - The ant then moves to one of these GA-determined locations. The selection of the "best" GA-produced location is typically guided by a fitness function (e.g., preferring locations that are good candidates for picking up an isolated object or dropping a carried object into a dense cluster).

The Hybrid Ant Clustering Algorithm(hACA) process

The hACA combines the ACA's probabilistic object manipulation with the GA-guided movement. The overall flow is described in Algorithm 1 of the paper:

1. **Initialization:**

    - Place all **N** artificial ants randomly on the grid.
    - Place all data objects randomly in the grid cells.
    - Set iteration counter **t = 0**.

2. **Iterative Clustering Loop (while t <= MaxIter):**

    - **For each ant i from 1 to N:**
      a. **Compute Local Density:** Calculate **f(xi)** at the ant's current location **xi** using **Equation (3)**.
      b. **Pick-up / Drop-off Action:**
      * **If** the ant is unloaded AND its current cell **xi** is occupied by an object **xj**:
      * Calculate **Pp(xi)** using **Equation (4)**.
      * Pick up object **xj** with probability **Pp(xi)**. If picked, the cell **xi** becomes empty.
      * **Else if** the ant is carrying an object **xj** AND its current cell **xi** is empty (or a suitable drop site):

* Calculate **Pd(xi)** using **Equation (5)**.
* Drop object **xj** with probability **Pd(xi)**. If dropped, cell **xi** becomes occupied by **xj**.

c. **Determine Next Location via GA:**

* Perform recombination (crossover) and mutation on binary representations of locations (as per Figure 2) to generate candidate next locations for the current ant.
* The ant moves to the selected GA-determined location.

- Increment iteration counter **t = t + 1**.

3. **Termination:**

- When **t** exceeds **MaxIter**, the algorithm stops.
- The final locations of the objects on the grid represent the formed clusters.

breakdown of every tunable parameter in hACA

1. Grid Dimension (m)
   • Role
   – Size of the toroidal world: there are m×m cells.
   • Increase m
   – Lowers overall density (same #objects in a bigger world).
   – Ants spend longer wandering before bumping into other items → more exploration.
   – Risk: clusters stay small and scattered.
   • Decrease m
   – Raises density, forces frequent encounters → more exploitation (fast cluster merging).
   – Risk: overcrowding → objects "stack up" or get repeatedly shuffled.

2. Number of Ants (N)
   • Role
   – How many mobile agents simultaneously rearrange objects.
   • Increase N
   – More parallel picks/drops → faster sampling of the grid.
   – Generally speeds up both exploration (many ants probing) and exploitation (many ants reinforcing clusters).
   – Too many ants can create "jitter," undoing formed clusters.
   • Decrease N
   – Slower overall dynamics → may get stuck in local patterns → more exploitation (since fewer attempts to break clusters).

3. Neighborhood Window Size (s)
   • Role
   – Each ant "sees" an s×s patch around itself to compute local density f(x).
   • Increase s
   – Density estimate uses more cells → smoother, more reliable measure.
   – Ants drop quickly in moderately large gatherings → strong exploitation (large, stable clusters).
   – Too large s blurs fine structure; distant clusters look "dense."

• Decrease s

– Ants respond to very local context only → noisy decisions → more exploration (micro-clusters form & break often).

4. Pick-up Threshold (k1)
   • Role
   – Governs pick-up probability:
   Pp = (k1 / (k1 + f))^2
   • Increase k1
   – Raise numerator → Pp larger for any fixed f → ants pick up objects even in semi-populated spots → exploration (breaking clusters).
   • Decrease k1
   – Lower pick-up chance → objects stay put unless truly isolated → exploitation (clusters remain intact).

5. Drop-off Threshold (k2)
   • Role
   – Governs drop-off probability:
   Pd = (f / (k2 + f))^2
   • Increase k2
   – Raises denominator → Pd smaller for a given f → ants carry objects longer in search of really dense zones → exploration (longer relocations).
   • Decrease k2
   – Boosts drop-off chance in moderate densities → exploitation (objects quickly reinforce existing blobs).

6. Mutation Rate (pμ) in GA
   • Role
   – Probability each bit in the offspring chromosome flips.
   • Increase pμ
   – Children coordinates become more random → jumps scatter widely → exploration.
   • Decrease pμ
   – Children stay close to parents' bits → small mutations only → exploitation (fine-tuning local clusters).

7. GA "Population" / Candidates per Ant
   • Role
   – How many crossover/mutation offspring you generate (including Parent-2 variants) before selecting one.
   • Increase #candidates
   – Ant evaluates more possible moves → more chance to find a high-density target → exploitation (smarter local jumps).
   • Decrease #candidates
   – Move proposals are fewer and more random → exploration.

8. Parent-2 Selection Strategy
   • Role
   – Determines the "diversity" gene in the crossover.
   • Random Grid Cell
   – Child may lie anywhere → exploration.

- Random Object of Same Type
  – Biased toward existing cluster of that type → exploitation.
- Nearest Same-Type Object
  – Very strong cluster bias → heavy exploitation (risk of getting stuck in local clusters).

9. Max Iterations (MaxIter)
   - Role
     – Total steps before stopping.
   - Increase MaxIter
     – More time to refine clusters → more exploitation (clusters solidify).
   - Decrease MaxIter
     – Stops early → clustering remains coarse → more exploration (insufficient convergence).

Tuning Tips

1. If you see too many tiny clusters: shrink **m**, increase **s**, decrease **k1**, decrease **pμ**, choose Parent-2 from same-type objects.
2. If clustering never stabilizes (ants keep redistributing endlessly): lower **k1** or increase **k2**, increase **MaxIter**, increase **s**.
3. Balance exploration vs. exploitation: start with moderate **k1≈0.1**, **k2≈0.3**, **s=3**, **m≈5√n**; then tweak one parameter at a time, observing its effect.

Simulated Annealing(SA)

Overview

Simulated Annealing (SA) is a probabilistic metaheuristic optimization algorithm inspired by the physical annealing process in metallurgy. It is particularly effective for combinatorial and continuous optimization problems, especially when the objective function has many local optima. SA explores the solution space by occasionally accepting worse solutions based on a temperature-controlled probability, which helps escape local minima and potentially reach a global optimum.

Core Components of Simulated Annealing

The Physical Analogy: Annealing in Metallurgy

- **A metal is heated to a high temperature and then gradually cooled.**
- **This slow cooling allows atoms to move freely initially and then settle into a low-energy crystalline state.**
- **Simulated Annealing mimics this by:**
    - **Starting with a high "temperature".**
    - **Slowly decreasing it according to a cooling schedule.**
    - **Allowing probabilistic acceptance of worse solutions to escape local minima.**

Solution Representation

- **A candidate solution is represented depending on the problem domain:**
    - **Binary strings, permutations, real-valued vectors, etc.**
- **The objective function evaluates the cost or energy of each candidate.**

Iteration Process

1. **Initialization:**
    - **Start with an initial solution S0S_0.**
    - **Set the initial temperature T0T_0.**
    - **Define a cooling schedule and stopping condition.**

2. **Neighbor Generation:**
    - **Generate a neighboring solution S'S' from the current solution SS.**
    - **The method of perturbation depends on the solution representation (e.g., swap two elements, add small noise).**
3. **Acceptance Probability:**
    - **Compute the difference in energy (objective cost):**
      ΔE=E(S')−E(S)\Delta E = E(S') - E(S)
    - **Accept S'S' with probability:**

P(accept)={1if ΔE≤0 (better solution)exp⁡(−ΔET)if ΔE>0 (worse solution)P(\text{accept}) = \begin{cases} 1 & \text{if } \Delta E \leq 0 \ (\text{better solution}) \\ \exp\left(-\frac{\Delta E}{T}\right) & \text{if } \Delta E > 0 \ (\text{worse solution}) \end{cases}

4. **Cooling:**
    - **Update the temperature using a cooling schedule, e.g.:**
        - **Geometric: Tk+1=αTkT_{k+1} = \alpha T_k, with α∈(0,1)\alpha \in (0,1)**
        - **Logarithmic or Linear: Less aggressive decay methods**
5. **Repeat:**
    - **Continue until a stopping condition is met (e.g., minimum temperature or maximum iterations).**

Key Parameters and Their Roles

Parameter	Role
Initial Temperature (T_0)	Sets the level of randomness in early iterations. High T_0 allows worse moves to be accepted, encouraging exploration.
Cooling Schedule	Determines how T decreases over time. Controls the exploration–exploitation balance.
Alpha (α)	In geometric cooling, Tk+1=αTkT_{k+1} = \alpha T_k. Higher α slows cooling (more exploration). Lower α encourages faster convergence.
Neighbor Function	Generates new candidate solutions. Affects search locality and solution space coverage.

Parameter	Role
Stopping Criteria	Limits total computation. Options: min temperature, max iterations, or no improvement over time.
Objective Function (Cost Function)	Problem-specific function to minimize or maximize. Defines "energy landscape" of the problem.

Tuning Tips

Issue	Suggested Fix
Stuck in local minima	Increase T_0 or slow down cooling (increase α). Use a more diverse neighbor function.
Too slow convergence	Lower T_0 or increase cooling rate (lower α). Use aggressive stopping criteria.
Unstable results (too random)	Reduce T_0, lower mutation strength in neighbors, or slow neighbor changes.
Good performance but noisy results	Increase iterations at lower temperatures to refine solutions.

Exploration vs. Exploitation in SA

High Temperature (Early Phase)	Low Temperature (Later Phase)
Accepts worse solutions → Exploration	Rejects worse solutions → Exploitation
Broad search of solution space	Fine-tuning around local optima
Helps escape local minima	Encourages convergence

Application Areas

- Traveling Salesman Problem (TSP)
- Job Scheduling
- VLSI design
- Neural Network Training

- **Combinatorial Optimization**

- **Hyperparameter tuning**

Literature review for Artificial Immune System Approach

Title	**Integration of Artificial Immune System and K-Means Algorithm for Customer Clustering**
Year	2014
Authors	R. J. Kuo, N. J. Chiang & Z.-Y. Chen
Keywords	Artificial Immune System (AIS),Clonal Selection (CS),K-means Algorithm,Customer Clustering , AISK Algorithm
Brief	AISK is a hybrid clustering algorithm that encodes K-means centroids as "antibodies," then uses AIS-style clonal selection and hypermutation to explore the search space while K-means updates refine solutions locally. In each generation it clones the best antibodies proportional to their affinity, mutates them (high-affinity clones mutate less), runs a K-means update, and carries forward top solutions (memory) plus a few random newcomers (remainder) to balance exploitation and exploration.

AISK: Artificial Immune System K-means Clustering Algorithm

The AISK algorithm is a novel clustering technique that integrates the Artificial Immune System (AIS) with the K-means algorithm. This hybrid approach aims to leverage the strengths of both AIS and K-means to improve the clustering process.

Artificial Immune System (AIS)

The AIS algorithm is a computational intelligence technique inspired by the biological immune system. In AIS, the population of antibodies is used to represent candidate solutions to the clustering problem.

Clonal Selection Principle

The clonal selection principle is a key component of AIS. It states that when an antibody recognizes an antigen, it proliferates, and its clones undergo hypermutation to improve their affinity for the antigen.

The clonal selection principle can be mathematically formulated as:

- Number of clones: $C = n.f$ where n is the number of antibodies to be cloned and f is a positive integer.
- Clone count for each antibody: $C_i = C \times \frac{(Ab)_i}{\sum_{i=1}^{n}(Ab)_i}$ where $(Ab)_i$ is the affinity of the i-th antibody.

K-means Clustering

K-means is a widely used unsupervised clustering algorithm that partitions the data into K clusters based on their similarities. The K-means algorithm minimizes the sum of squared distances between the data points and their assigned cluster centroids.

The K-means objective function is given by:

$$D_i = \sum_{i=1}^{k} \sum_{x_j \epsilon c_i} \|x_j - \mu_i\|^2$$

where K is the number of clusters, Ci is the i-th cluster, Xj is a data point, and μi is the centroid of the i-th cluster.

AISK Algorithm

The AISK algorithm combines the AIS and K-means clustering techniques. The algorithm starts by initializing a population of antibodies randomly in the search space. Then, it applies the following steps:

1. Evaluate fitness: Evaluate the fitness of each antibody using the Affinity objective function.

$$(AB)_i = \frac{1}{1 + D_i}$$

2. Clonal selection: Select the top n antibodies with the highest fitness and clone them proportionally to their fitness using the following procedure :

   I. Reproduce the *Pn* in the antibody population, and generate the new antibody population (*C*) followed by clonal selection through
   Equation (6).

$$C = f.n$$

   II. When the clonal reproduction has been performed, the scale of antibody in the population (pn) will be proportional to the affinity of the antibody which is shown as Equation (7):

$$C \times \frac{(Ab)_i}{\sum_{i=1}^{n}(Ab)_i}$$

3. Affinity maturation: Apply hypermutation to the clones to introduce diversity.

$$Mutation\ rate\ =\ \alpha_i = e^{-\rho.(Ab^*)_i}$$

   Where $(Ab^*)_i = \frac{(Ab)_i}{(Ab)_{max}}$ is the normalized affinity value,ρ is constant

   Affinity maturation procedure:

   I. For each antibody, generate a random variable *r* that has a uniform distribution U(0,1)
   II. If *r* is smaller than the mutation rate, then mutation is executed for the antibody
   III. Adopt the "Gaussian mutation" approach as Equation (9) to change each element of the attribute in the antibody:

$$m` = m + \alpha_i.N(0,1)$$

   where *m* = (*m*1,*m*2, · · ·,*mL*) is the attribute list of antibodies once the clonal selection is performed ; $m`$ is the attribute list of antibodies after mutation; Finally, *N*(0, 1) is the vector of independent Gaussian random variables

4. K-means refinement: Apply K-means clustering to the mutated clones to refine the cluster assignments and centroids.

5. Update memory and remainder cells: Update the memory cells with the best antibodies and replace the remainder cells with new random antibodies

Advantages

The AISK algorithm has several advantages, including:

- Improved exploration: The algorithm balances exploration and exploitation by using AIS to search for promising regions in the search space.
- Robust clustering: The algorithm is robust to noise and outliers in the data.
- Flexibility: The algorithm can be applied to various clustering problems, including customer segmentation.

Example Walkthrough

Suppose we have a dataset of customers with RFM attributes, and we want to cluster them into 3 segments using AISK.

1. Initialize a population of 100 antibodies, each representing 3 cluster centroids.
2. Evaluate the fitness of each antibody using the K-means objective function.
3. Select the top 20 antibodies with the highest fitness and clone them proportionally to their fitness.
4. Apply hypermutation to the clones to introduce diversity.
5. Apply K-means clustering to the mutated clones to refine the cluster assignments and centroids.
6. Update the memory cells with the best antibodies and replace the remainder cells with new random antibodies.
7. Repeat steps 2-6 until the termination criterion is met.

The final clustering solution is obtained from the best antibody in the memory cells, which represents the 3 customer segments.

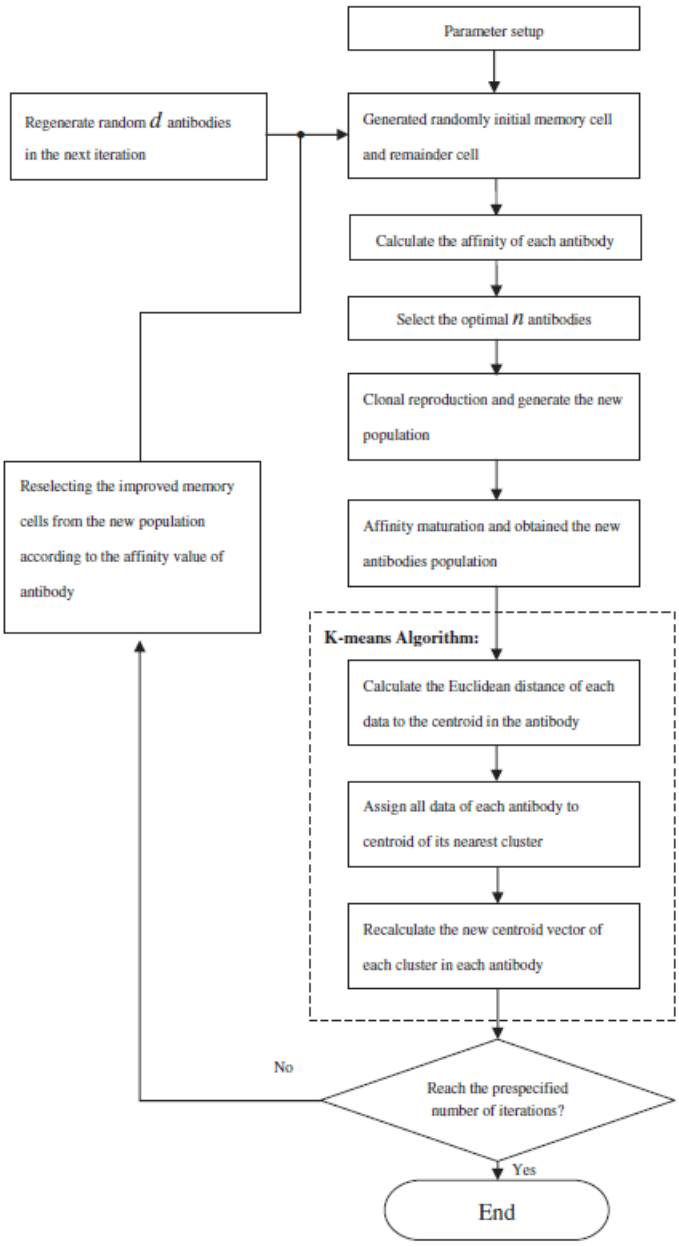


FIGURE 2 The flowchart of the proposed AISK algorithm.

breakdown of every tunable parameter in AISK

1. memory_size (M)
   • Number of best antibodies carried forward each generation ("memory cells").
   • If you increase M → you preserve more good solutions → stronger **exploitation** (you "trust" your current best and refine them).
   • If you decrease M → you keep fewer old solutions → opens room to new ones → more **exploration**.

2. remainder_size (Pr)
   • Number of fresh/random antibodies inserted each generation.
   • Increase Pr → more random newcomers → more **exploration** (search new areas).
   • Decrease Pr → fewer random antibodies → less exploration, more focus on existing ones → more **exploitation**.

3. selection_size (n)
   • How many top antibodies you pick each generation to clone.
   • Increase n → you select more parents → broader exploitation of good regions (more different centers get cloned).
   • Decrease n → you only pick the very top → you concentrate effort on the absolute best → narrower exploitation vs. some loss of diversity.

4. clone_factor (f)
   • Multiplier that determines total clones: $C = n \cdot f$.
   • Increase f → you generate more clones per selected antibody → deeper **exploitation** around those good solutions.
   • Decrease f → fewer clones → lighter local search, leaving more room for exploration by the remainder cells.

5. ρ (rho) – mutation-rate parameter
   • Controls how aggressively clones mutate:
   $\alpha_i = \exp(-\rho \cdot \text{normalized\_affinity}_i)$
   • If ρ is large → high-affinity parents get extremely low mutation (hardly explore), low-affinity ones mutate almost fully → strong **exploitation** of best and strong **exploration** of worst.
   • If ρ is small → all clones mutate at moderate rates → more uniform **exploration** overall.

6. max_iter
   • Maximum number of generations to run.
   • Increase max_iter → more time to both explore (early) and exploit (late).
   • Decrease max_iter → shorter run, risk of stuck in local minima (less both exploration & exploitation).

Summary:

• Exploration drivers:

– Pr (remainder_size): fresh random antibodies

– small ρ: higher mutation across the board

– small M: fewer memories → more space for new solutions

– smaller f: fewer clones → more room for random remainder

• Exploitation drivers:

– M (memory_size): keep and refine more best solutions

– n (selection_size): clone more of the top individuals

– f (clone_factor): heavier cloning of good solutions

– large ρ: low-aff clones mutate heavily, best clones hardly move

Experiments and evaluation

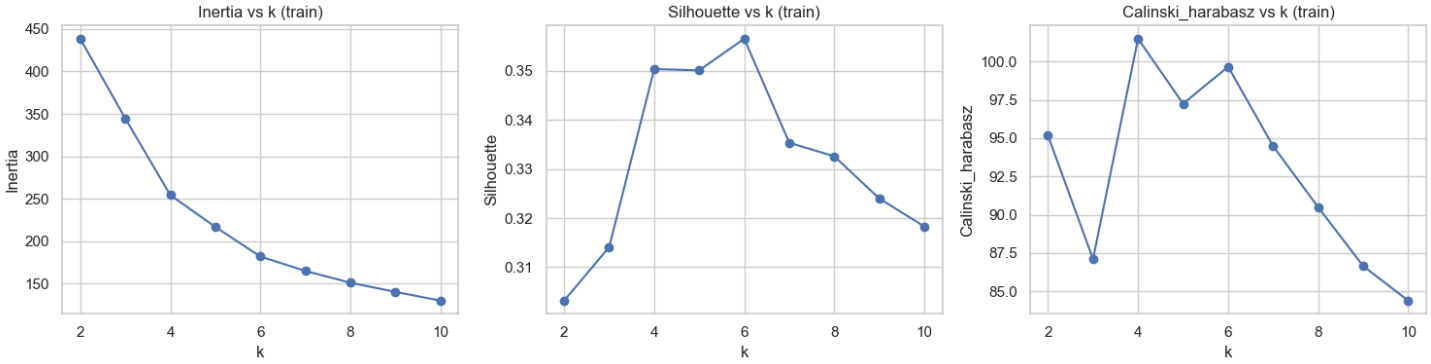
Metric	Goal	Explanation
Inertia	**Minimize**	Measures within-cluster sum-of-squares (compactness). Lower is better.
Silhouette	**Maximize**	Measures how similar an object is to its own cluster vs. others. Higher is better.
Calinski-Harabasz	**Maximize**	Ratio of between-cluster dispersion to within-cluster dispersion. Higher is better.

Standard K-means benchmarks

We ran the experiment 30 times and took the average.

Standard K-means benchmarks on mall customer segmentation data

k	dataset	inertia	silhouette	calinski_harabasz
2	train	438.522412	0.303198	95.160478
3	train	344.564859	0.314088	87.108421
4	train	254.287142	0.350318	101.484488
5	train	216.809241	0.350059	97.241952
6	train	181.951436	0.356486	99.654879
7	train	164.919302	0.335315	94.473259
8	train	151.037433	0.332566	90.485458
9	train	140.264285	0.323972	86.646473
10	train	129.931941	0.318277	84.390833



We normalize the weights and unify the objective direction by converting all metrics to maximization (using the negative of inertia). Then, we scale all metrics using MinMaxScaler to enable comparison. The final score is a weighted composition of the scaled metrics: 0.4 each for inertia and Calinski-Harabasz, and 0.2 for silhouette.

k	dataset	inertia	silhouette	calinski_harabasz	composite_score
2	train	0.000000	0.000000	0.630038	0.252015
3	train	0.304473	0.204364	0.158982	0.226255
4	train	0.597022	0.884255	1.000000	0.815660
5	train	0.718471	0.879393	0.751806	0.763989
6	train	0.831429	1.000000	0.892966	0.889758
7	train	0.886622	0.602716	0.589834	0.711126
8	train	0.931607	0.551120	0.356543	0.625484
9	train	0.966518	0.389858	0.131958	0.517362
10	train	1.000000	0.282984	0.000000	0.456597

We can also use the **Pareto frontier** (also called **Pareto front** or **Pareto optimal set**) is a concept from **multi-objective optimization**.

A solution is **Pareto optimal** if **no other solution is better in all objectives at once, the Pareto front includes all the solutions where:**

- **We can't improve one metric without making at least one other worse.**

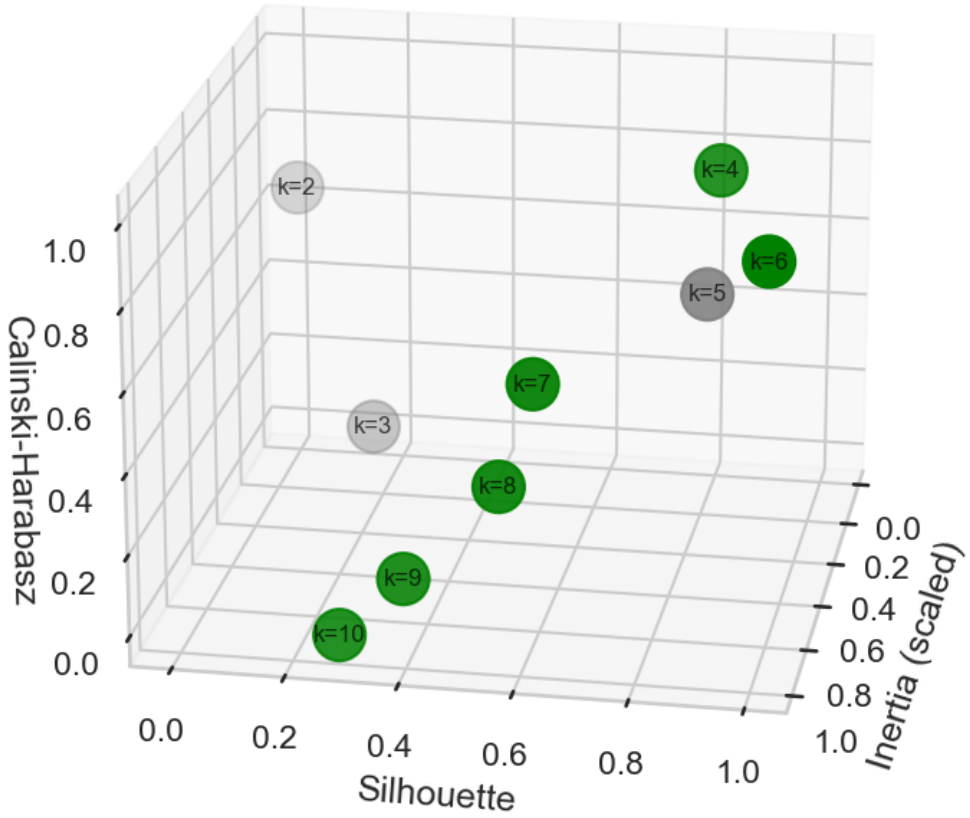
The Pareto frontier includes all points that aren't "dominated" by any others.

A point A dominates point B if A is equal or better in all metrics, and strictly better in at least one.

Pareto optimal set (k=6, is the best according to our weights, or tradeoff)

k	inertia	silhouette	calinski_harabasz	composite_score
4	0.597022	0.884255	1.000000	0.815660
6	0.831429	1.000000	0.892966	0.889758
7	0.886622	0.602716	0.589834	0.711126
8	0.931607	0.551120	0.356543	0.625484
9	0.966518	0.389858	0.131958	0.517362
10	1.000000	0.282984	0.000000	0.456597

Pareto Front in 3D Metric Space with 'k' Labels

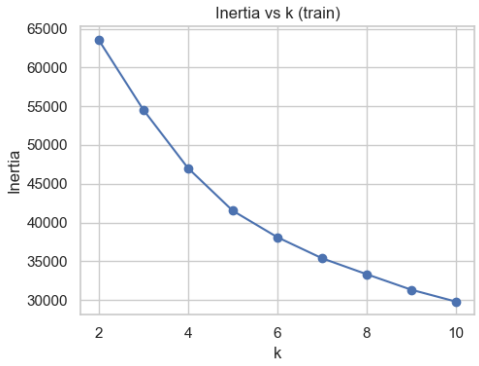


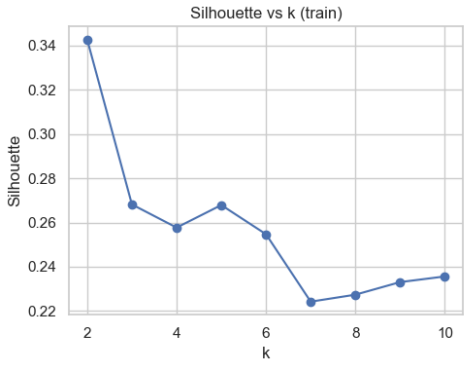
Standard K-means benchmarks on customer segmentation data

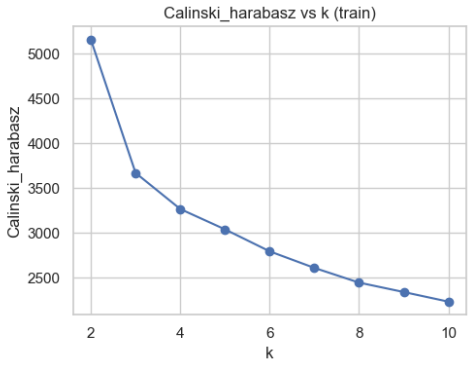
k	dataset	inertia	silhouette	calinski_harabasz	inertia/samples
2	train	63574.315811	0.342596	5154.901068	7.879811
3	train	54558.984442	0.268711	3669.310188	6.762393
4	train	47037.091681	0.258089	3266.888090	5.830081
5	train	41586.576834	0.265683	3035.200079	5.154509
6	train	38123.906255	0.253794	2794.786853	4.725323
7	train	35397.839814	0.224656	2611.492722	4.387437
8	train	33339.768498	0.227608	2447.389531	4.132346
9	train	31347.836288	0.233127	2341.280397	3.885453
10	train	29841.836769	0.235297	2231.114762	3.698790

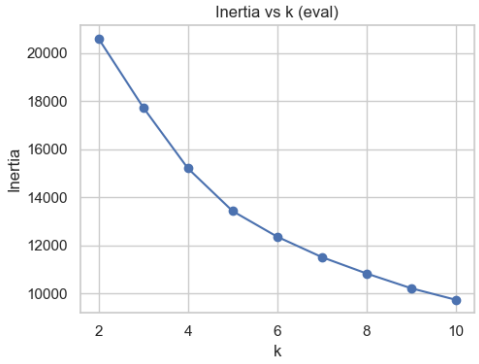
k	dataset	inertia	silhouette	calinski_harabasz	inertia/samples
2	eval	20607.099081	0.351026	1758.619203	7.844347
3	eval	17737.647556	0.267842	1234.187945	6.752055
4	eval	15204.889673	0.256681	1105.060367	5.787929
5	eval	13444.149269	0.265602	1023.970787	5.117681
6	eval	12357.282591	0.254210	937.539890	4.703952
7	eval	11502.859458	0.224579	872.190488	4.378706
8	eval	10829.848881	0.227509	817.099664	4.122516
9	eval	10211.720865	0.231669	778.261117	3.887218
10	eval	9740.185195	0.233701	739.512030	3.707722

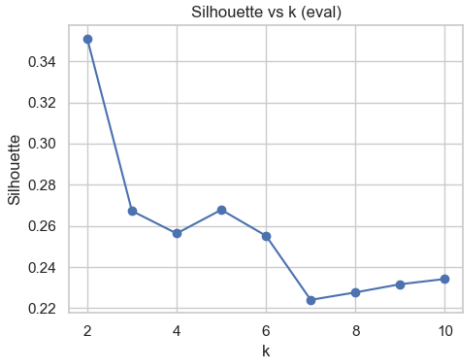
For easy comparison between the training and testing (eval) we would divide each metric by the total number of samples in the set (either training set or testing set) so the measure be homogeneous.

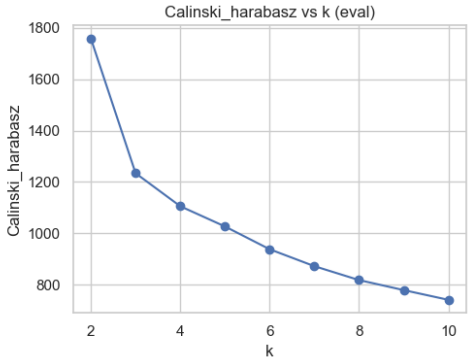












k	dataset	inertia/samples	calinski_harabasz/samples	silhouette/samples
2	train	7.879811	0.638932	0.000042
3	train	6.762393	0.454798	0.000033
4	train	5.830081	0.404919	0.000032
5	train	5.154509	0.376202	0.000033
6	train	4.725323	0.346404	0.000031
7	train	4.387437	0.323685	0.000028
8	train	4.132346	0.303345	0.000028
9	train	3.885453	0.290193	0.000029
10	train	3.698790	0.276539	0.000029

k	dataset	inertia/samples	calinski_harabasz/samples	silhouette/samples
2	eval	7.844347	0.669440	0.000134
3	eval	6.752055	0.469809	0.000102
4	eval	5.787929	0.420655	0.000098
5	eval	5.117681	0.389787	0.000101
6	eval	4.703952	0.356886	0.000097
7	eval	4.378706	0.332010	0.000085
8	eval	4.122516	0.311039	0.000087
9	eval	3.887218	0.296255	0.000088
10	eval	3.707722	0.281504	0.000089

Using the composite score (weighted summation of metrics) we see that k=2 is the optimal here.

k	dataset	inertia	silhouette	calinski_harabasz	inertia/samples	calinski_harabasz/samples	silhouette/samples	composite_score
2	train	0.000000	1.000000	1.000000	7.879811	0.638932	0.000042	0.600000
3	train	0.267260	0.373538	0.491895	6.762393	0.454798	0.000033	0.378369
4	train	0.490246	0.283471	0.354258	5.830081	0.404919	0.000032	0.394496
5	train	0.651827	0.347866	0.275015	5.154509	0.376202	0.000033	0.440310
6	train	0.754478	0.247055	0.192788	4.725323	0.346404	0.000031	0.428318
7	train	0.835292	0.000000	0.130098	4.387437	0.323685	0.000028	0.386156
8	train	0.896304	0.025032	0.073971	4.132346	0.303345	0.000028	0.393116
9	train	0.955355	0.071821	0.037679	3.885453	0.290193	0.000029	0.411578
10	train	1.000000	0.090222	0.000000	3.698790	0.276539	0.000029	0.418044

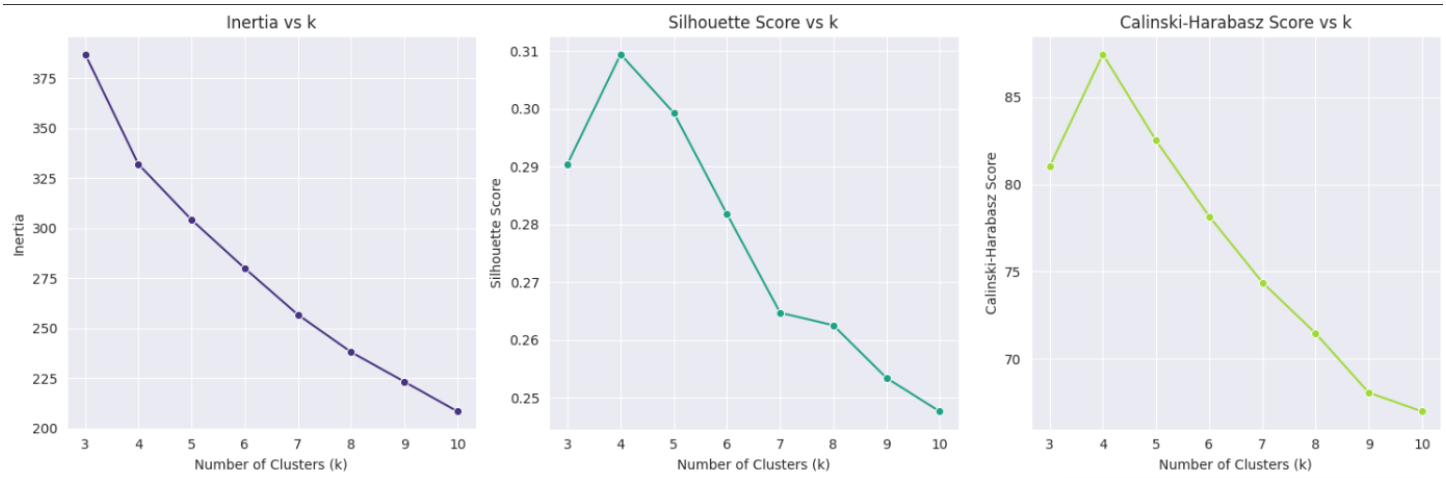
k	dataset	inertia	silhouette	calinski_harabasz	inertia/samples	calinski_harabasz/samples	silhouette/samples	composite_score
2	eval	0.000000	1.000000	1.000000	7.844347	0.669440	0.000134	0.600000
3	eval	0.264054	0.342143	0.485401	6.752055	0.469809	0.000102	0.368211
4	eval	0.497125	0.253880	0.358695	5.787929	0.420655	0.000098	0.393104
5	eval	0.659152	0.324433	0.279125	5.117681	0.389787	0.000101	0.440198
6	eval	0.759168	0.234337	0.194315	4.703952	0.356886	0.000097	0.428261
7	eval	0.837794	0.000000	0.130191	4.378706	0.332010	0.000085	0.387194
8	eval	0.899726	0.023178	0.076133	4.122516	0.311039	0.000087	0.394979
9	eval	0.956608	0.056077	0.038023	3.887218	0.296255	0.000088	0.409068
10	eval	1.000000	0.072141	0.000000	3.707722	0.281504	0.000089	0.414428

Differential Evolution approaches experiments and evaluation

Swarm Intelligence approaches experiments and evaluation

KCGWO benchmarks on mall customer segmentation data

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score
0	3	387.003152	0.290479	81.038290	1.241073
1	4	332.057325	0.309436	87.466384	1.171317
2	5	304.251032	0.299299	82.517473	1.181248
3	6	280.020092	0.281799	78.163502	1.277453
4	7	256.827495	0.264711	74.375683	1.262615
5	8	238.143309	0.262550	71.456197	1.300009
6	9	223.346620	0.253436	68.052052	1.343046
7	10	208.471104	0.247647	66.970844	1.324155

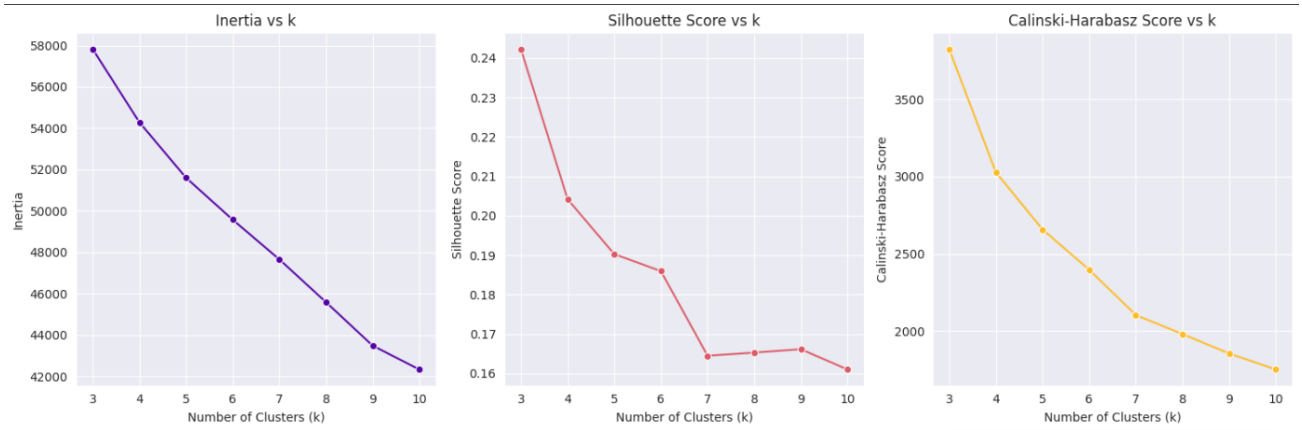


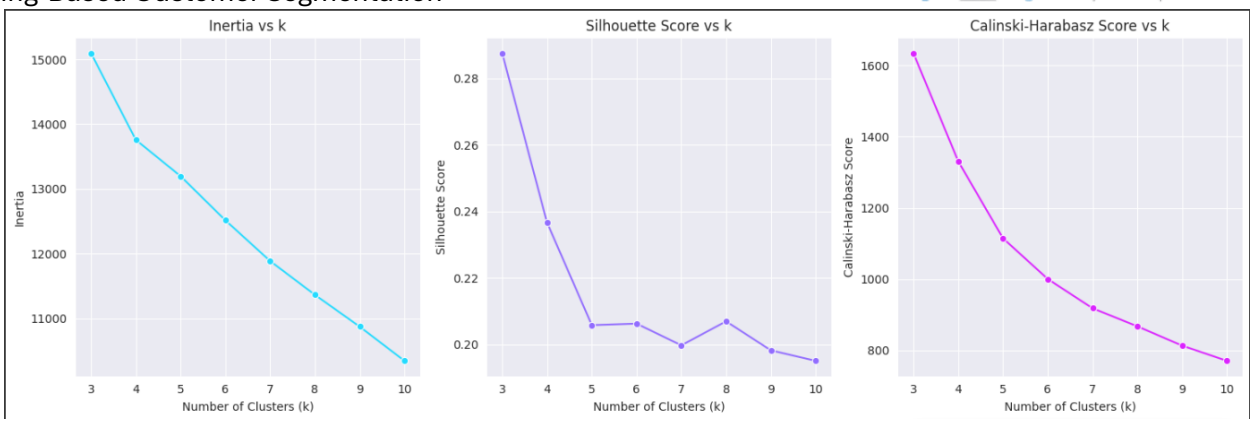
	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	composite_score
0	3	-0.000000	0.693193	0.686366	1.241073	0.413185
1	4	0.307765	1.000000	1.000000	1.171317	0.723106
2	5	0.463514	0.835947	0.758537	1.181248	0.656010
3	6	0.599237	0.552723	0.546102	1.277453	0.568680
4	7	0.729144	0.276170	0.361290	1.262615	0.491408
5	8	0.833799	0.241190	0.218845	1.300009	0.469296
6	9	0.916679	0.093687	0.052753	1.343046	0.406510
7	10	1.000000	0.000000	0.000000	1.324155	0.400000

KCGWO benchmarks on customer segmentation data

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset
0	3	57824.929808	0.242254	3824.966685	1.638805	train
1	4	54287.290249	0.204188	3028.331487	1.757613	train
2	5	51602.599960	0.190282	2658.239815	1.776527	train
3	6	49573.165006	0.185957	2397.183259	1.811206	train
4	7	47651.844136	0.164511	2103.780252	1.966742	train
5	8	45582.006660	0.165298	1980.878379	1.933725	train
6	9	43478.505170	0.166157	1854.840754	1.883844	train
7	10	42323.904588	0.161015	1751.273606	1.898132	train

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset
0	3	15094.553368	0.287540	1633.965714	1.370564	eval
1	4	13756.979813	0.236639	1330.791660	1.572952	eval
2	5	13195.816784	0.205779	1114.762817	1.668627	eval
3	6	12514.783061	0.206234	1000.487217	1.693436	eval
4	7	11885.221561	0.199709	918.449677	1.681487	eval
5	8	11363.305790	0.206920	867.764069	1.662324	eval
6	9	10872.093034	0.198196	813.666924	1.673849	eval
7	10	10343.942971	0.195044	770.919232	1.684253	eval





	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset	composite_score
0	3	0.000000	1.000000	1.000000	1.638805	train	0.600000
1	4	0.228220	0.531432	0.615837	1.757613	train	0.443909
2	5	0.401414	0.360261	0.437368	1.776527	train	0.407565
3	6	0.532337	0.307027	0.311478	1.811206	train	0.398931
4	7	0.656285	0.043035	0.169990	1.966742	train	0.339117
5	8	0.789814	0.052721	0.110723	1.933725	train	0.370759
6	9	0.925515	0.063292	0.049943	1.883844	train	0.402842
7	10	1.000000	0.000000	0.000000	1.898132	train	0.400000

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset	composite_score
0	3	0.000000	1.000000	1.000000	1.370564	eval	0.600000
1	4	0.281558	0.449691	0.648716	1.572952	eval	0.462048
2	5	0.399683	0.116053	0.398407	1.668627	eval	0.342446
3	6	0.543040	0.120977	0.265997	1.693436	eval	0.347810
4	7	0.675562	0.050432	0.170941	1.681487	eval	0.348688
5	8	0.785425	0.128395	0.112213	1.662324	eval	0.384734
6	9	0.888825	0.034070	0.049531	1.673849	eval	0.382156
7	10	1.000000	0.000000	0.000000	1.684253	eval	0.400000

So here we choose k=3 according to KCGWO Algorithm.

EA approaches experiments and evaluation

AE-IEMOKC benchmarks on mall customer segmentation

- Best k by silhouette: 2.0 (score: 0.195)
- Best k by inertia: 10.0 (score: 258.327)
- Best k by Calinski-Harabasz: 3.0 (score: 44.336)

k	dataset	inertia	silhouette	calinski_harabasz
2	train	733.623430	0.195195	44.066356
3	train	587.578052	0.167553	44.336302
4	train	504.126359	0.145271	38.951193
5	train	438.096594	0.154706	41.801746
6	train	375.305730	0.145952	40.146167
7	train	341.884133	0.148493	41.491078
8	train	310.988937	0.149294	40.988635
9	train	274.133776	0.141900	39.295007
10	train	258.326911	0.144725	39.793780

k	dataset	inertia	silhouette	calinski_harabasz	composite_score
2	train	0.000000	1.000000	0.949872	0.579949
3	train	0.307272	0.481339	1.000000	0.619177
4	train	0.482850	0.063243	0.000000	0.205789
5	train	0.621774	0.240287	0.529340	0.508503
6	train	0.753882	0.076033	0.221904	0.405521
7	train	0.824200	0.123702	0.471650	0.543080
8	train	0.889202	0.138739	0.378348	0.534767
9	train	0.966743	0.000000	0.063845	0.412235
10	train	1.000000	0.053002	0.156466	0.473187

AE-IEMOKC benchmarks on customer segmentation data

- Best k by silhouette: 2.0 (score: 0.267)
- Best k by inertia: 10.0 (score: 52275.304)
- Best k by Calinski-Harabasz: 2.0 (score: 3033.148)

k	dataset	inertia	silhouette	calinski_harabasz
2	eval	34306.574040	0.277445	1069.050101
3	eval	34181.327633	0.220056	692.410479
4	eval	27358.683832	0.196588	749.258097
5	eval	26007.500825	0.147246	625.084087
6	eval	22927.049702	0.098949	441.909888
7	eval	20892.948707	0.155486	558.207023
8	eval	19261.137477	0.166035	538.932580
9	eval	22077.895065	0.121950	377.995776
10	eval	17039.824539	0.115996	427.842692

k	dataset	inertia	silhouette	calinski_harabasz
2	train	104847.221535	0.266506	3033.147775
3	train	105349.685452	0.214446	2016.829679
4	train	83737.011536	0.199341	2236.967873
5	train	79923.349256	0.148978	1851.706367
6	train	70194.794254	0.102686	1320.490194
7	train	63885.141118	0.150674	1649.485368
8	train	59111.062015	0.170817	1630.176318
9	train	67158.703832	0.130223	1150.017670
10	train	52275.304029	0.119918	1314.680070

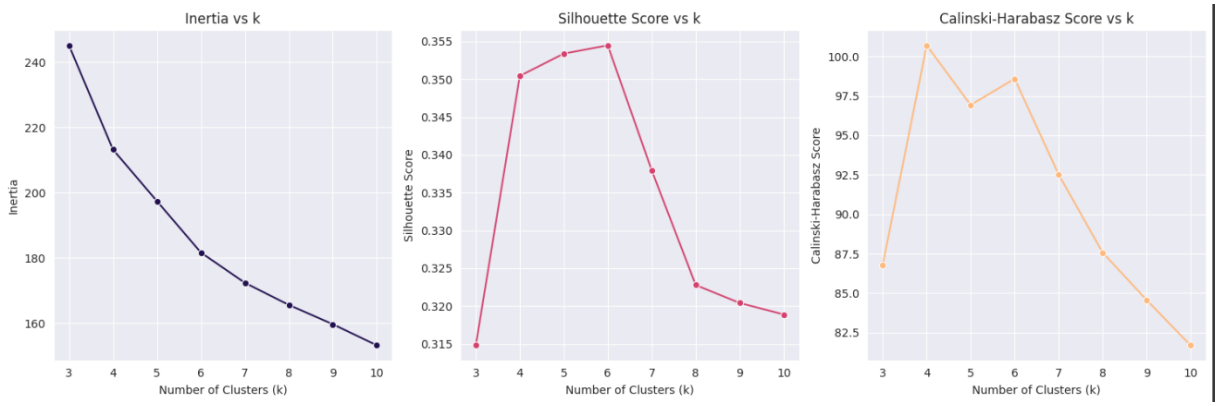
k	dataset	inertia	silhouette	calinski_harabasz	composite_score
2	train	0.009467	1.000000	1.000000	0.603787
3	train	0.000000	0.682214	0.460304	0.320564
4	train	0.407215	0.590008	0.577204	0.511769
5	train	0.479070	0.282577	0.372618	0.397191
6	train	0.662370	0.000000	0.090526	0.301159
7	train	0.781253	0.292927	0.265233	0.477180
8	train	0.871204	0.415888	0.254979	0.533651
9	train	0.719575	0.168088	0.000000	0.321447
10	train	1.000000	0.105187	0.087441	0.456014

k	dataset	inertia	silhouette	calinski_harabasz	composite_score
2	eval	0.000000	1.000000	1.000000	0.600000
3	eval	0.007254	0.678485	0.454978	0.320590
4	eval	0.402386	0.547010	0.537240	0.485252
5	eval	0.480639	0.270581	0.357553	0.389393
6	eval	0.659043	0.000000	0.092488	0.300612
7	eval	0.776847	0.316745	0.260777	0.478399
8	eval	0.871353	0.375843	0.232886	0.516864
9	eval	0.708221	0.128862	0.000000	0.309061
10	eval	1.000000	0.095508	0.072132	0.447954

Artificial Immune System (AIS) approach experiments and evaluation

AISK benchmarks on mall customer segmentation data

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score
0	3	245.085535	0.314820	86.787147	1.161853
1	4	213.186147	0.350440	100.701404	1.057712
2	5	197.381230	0.353383	96.925196	1.034261
3	6	181.628389	0.354475	98.584646	1.013196
4	7	172.359766	0.337946	92.524186	1.043569
5	8	165.581477	0.322786	87.576098	1.095477
6	9	159.669089	0.320417	84.567830	1.107333
7	10	153.217756	0.318898	81.692155	1.101232



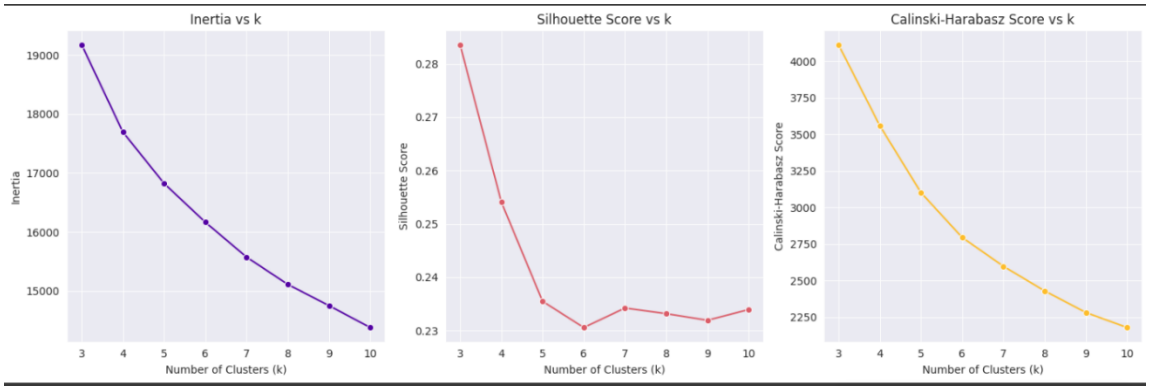
	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	composite_score
0	3	-0.000000	0.000000	0.268027	1.161853	0.107211
1	4	0.347232	0.898231	1.000000	1.057712	0.718539
2	5	0.519271	0.972451	0.801349	1.034261	0.722738
3	6	0.690744	1.000000	0.888646	1.013196	0.831756
4	7	0.791635	0.583168	0.569830	1.043569	0.661219
5	8	0.865418	0.200881	0.309531	1.095477	0.510156
6	9	0.929776	0.141146	0.151278	1.107333	0.460651
7	10	1.000000	0.102833	0.000000	1.101232	0.420567

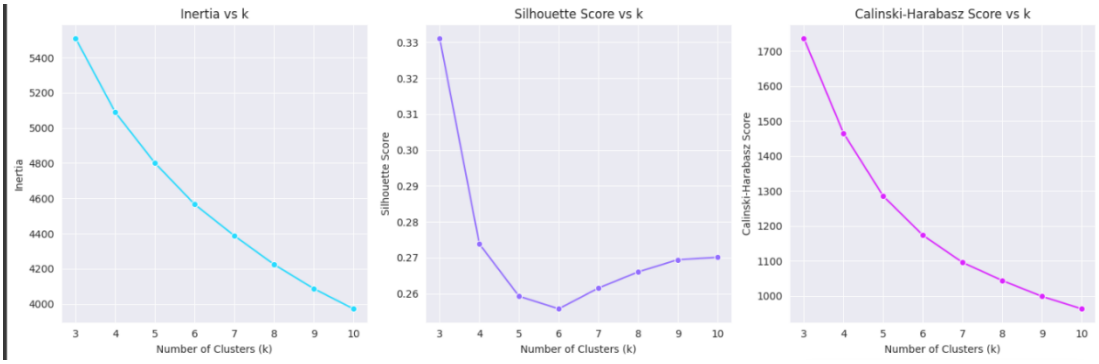
So here we choose k=6 according to AISK Algorithm.

AISK benchmarks on customer segmentation data

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset
0	3	19174.498251	0.283585	4112.678543	1.393860	train
1	4	17693.362868	0.254101	3559.163826	1.475639	train
2	5	16821.665255	0.235481	3100.506325	1.521543	train
3	6	16165.088883	0.230551	2793.566082	1.542022	train
4	7	15573.526616	0.234214	2596.644285	1.503162	train
5	8	15110.743149	0.233156	2428.166103	1.512900	train
6	9	14750.619708	0.231917	2280.392817	1.520862	train
7	10	14378.871829	0.233928	2177.345887	1.502361	train

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset
0	3	5509.986732	0.331131	1737.122786	1.191224	eval
1	4	5089.083502	0.273831	1465.485851	1.378403	eval
2	5	4800.819457	0.259251	1285.384410	1.384585	eval
3	6	4566.206283	0.255756	1173.208023	1.388889	eval
4	7	4387.870720	0.261534	1095.317237	1.374500	eval
5	8	4224.661968	0.266033	1043.596396	1.363156	eval
6	9	4087.334973	0.269448	998.212586	1.356946	eval
7	10	3972.352719	0.270095	962.578255	1.332507	eval





	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset	composite_score
0	3	0.000000	1.000000	1.000000	1.393860	train	0.600000
1	4	0.308851	0.444053	0.713995	1.475639	train	0.497949
2	5	0.490621	0.092967	0.477003	1.521543	train	0.405643
3	6	0.627532	0.000000	0.318405	1.542022	train	0.378375
4	7	0.750887	0.069069	0.216654	1.503162	train	0.400830
5	8	0.847388	0.049125	0.129601	1.512900	train	0.400620
6	9	0.922482	0.025756	0.053245	1.520862	train	0.395442
7	10	1.000000	0.063675	0.000000	1.502361	train	0.412735

	k	inertia	silhouette	Calinski-Harabasz-score	davies-bouldin-score	dataset	composite_score
0	3	0.000000	1.000000	1.000000	1.191224	eval	0.600000
1	4	0.273734	0.239794	0.649295	1.378403	eval	0.417170
2	5	0.461207	0.046369	0.416769	1.384585	eval	0.360464
3	6	0.613787	0.000000	0.271940	1.388889	eval	0.354291
4	7	0.729768	0.076655	0.171377	1.374500	eval	0.375789
5	8	0.835911	0.136347	0.104601	1.363156	eval	0.403474
6	9	0.925221	0.181647	0.046007	1.356946	eval	0.424821
7	10	1.000000	0.190239	0.000000	1.332507	eval	0.438048

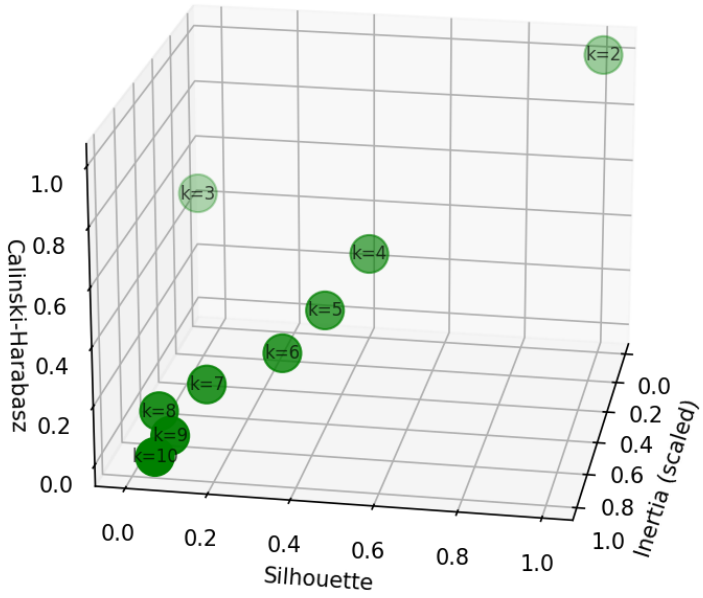
So here we choose k=3 according to AISK Algorithm.

Hybrid approaches experiments and evaluation

Dataset (utomobile-customer-segmentation)

	k	interia	silhouette	calinski_harabasz	composite_score
0	2	2.220446e-16	1.000000	1.000000	0.600000
1	3	2.352179e-01	0.000000	0.535089	0.308123
2	4	4.704054e-01	0.483569	0.470374	0.473026
3	5	6.049749e-01	0.397195	0.331533	0.454042
4	6	7.103173e-01	0.311191	0.228165	0.437631
5	7	7.978363e-01	0.141323	0.149504	0.407201
6	8	8.723214e-01	0.038925	0.086737	0.391409
7	9	9.432770e-01	0.081999	0.044159	0.411374
8	10	1.000000e+00	0.057115	0.000000	0.411423

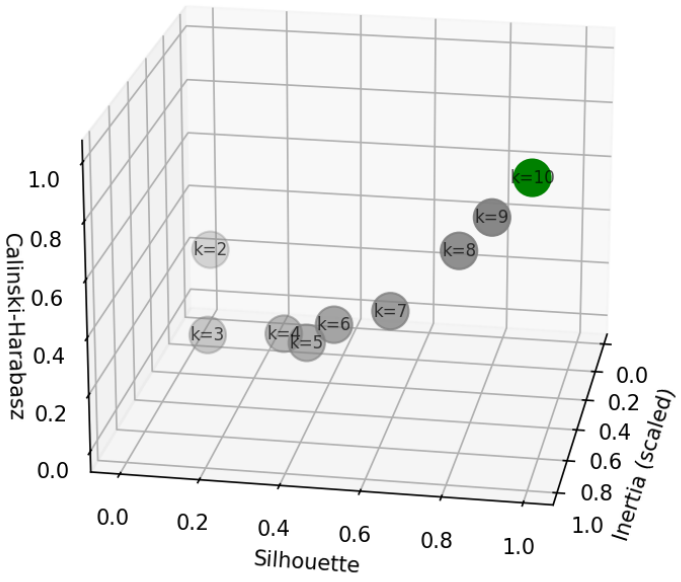
Pareto Front in 3D Metric Space with 'k' Labels



Dataset (mall-customer-segmentation)

	K	inertia	silhouette	calinski_harabasz	composite_score
0	2	0.000000	0.000000	0.193919	0.077568
1	3	0.255367	0.045650	0.000000	0.111277
2	4	0.462005	0.292376	0.133240	0.296573
3	5	0.601275	0.379249	0.179558	0.388183
4	6	0.717650	0.471256	0.312856	0.506453
5	7	0.805506	0.628694	0.419630	0.615793
6	8	0.891721	0.810854	0.687208	0.793742
7	9	0.950610	0.898521	0.836040	0.894364
8	10	1.000000	1.000000	1.000000	1.000000

Pareto Front in 3D Metric Space with 'k' Labels



Conclusion

Approach	Best composite score (dataset1)	Best K (dataset1)	Best composite score (dataset2)	Best K (dataset2) Train/Validation
Standard K-means	0.889	6	0.600 /0.600	2/2
KCGWO	0.723	4	0.600 /0.600	3/3
AE-IEMOKC	0.619	3	0.603 /0.600	2/2
AISK	0.831	6	0.600 /0.600	3/3
SA-PSO-GK++ (Hybrid)	0.999	10	0.600/-	2/-
SA	0.600	3	0.602/0.600	3/3
ACDE	0.999	4	0.994/-	2/-
DE	0.747	3	0.909/-	3/-