Pattern Analysis and Applications (2025) 28:107
https://doi.org/10.1007/s10044-025-01463-4

ORIGINAL ARTICLE

Efficient error minimization in kernel *k*-means clustering

Georgios Vardakas[1] · **Ioannis Papakostas**[1] · **Aristidis Likas**[1]

Received: 5 January 2025 / Accepted: 25 March 2025 / Published online: 22 May 2025
© The Author(s) 2025

Abstract
Kernel *k*-means extends the *k*-means algorithm to identify non-linearly separable clusters but is inherently sensitive to cluster initialization. To address this challenge, we first formulate the *kernel k-means* ++ method, which conveys the efficient center initialization strategy of *k*-means++ from Euclidean to kernel space. Building on this, we propose *global kernel k-means* ++ (GK*k*M++), a novel clustering algorithm designed to balance clustering error minimization with reduced computational cost. GK*k*M++ extends the well-established global kernel *k*-means algorithm by incorporating the stochastic initialization strategy of kernel *k*-means++. This approach significantly reduces computational complexity while preserving superior clustering error minimization capabilities akin to traditional global kernel *k*-means. The experimental results on synthetic, real, and graph datasets indicate that GK*k*M++ consistently outperforms both kernel *k*-means with random initialization and kernel *k*-means++, while achieving solutions comparable to those provided by the exhaustive and computational intensive global kernel *k*-means method.

Keywords Clustering · Graph partitioning · Kernel *k*-means · Global kernel *k*-means · Global optimization

1 Introduction

Clustering is a fundamental unsupervised learning problem, aiming at partitioning a dataset into homogeneous groups (called clusters) based on a similarity or distance measure [1–3]. Among the many clustering algorithms, *k*-means [4, 5] is particularly notable for its simplicity and speed. Specifically, the algorithm's objective is to identify groups by minimizing an objective called clustering error. Assuming a dataset $X = \{x_1, \ldots, x_N\}$, where $x_i \in \mathbb{R}^d$, partitioned into K disjoint clusters $\mathcal{C} = \{C_1, \ldots, C_K\}$, the clustering error is expressed as the sum of the squared Euclidean distances between each $x_i \in C_k$ to its respective cluster center μ_k:

$$E(\mathcal{C}) = \sum_{i=1}^{N} \sum_{k=1}^{K} 1_{C_k}(x_i) ||x_i - \mu_k||^2. \quad (1)$$

In this formulation, 1_{C_k} represents the indicator function for the set C_k, while the set of cluster centers $M = \{\mu_1, \ldots, \mu_K\}$ contains the mean of the data instances in each associated cluster. The symbol notations used in this paper are given in Table 1. In particular, the *k*-means algorithm suffers from two main limitations:

1. **Sensitivity to initial centers:** The solution relies critically on the initial placements of the cluster centers. It is possible that, due to poor initialization, the *k*-means may converge to poor local minima of the clustering error.
2. **Linear separability:** The resulting clusters are restricted to being linearly separable, limiting the algorithm's effectiveness in identifying complex cluster shapes.

One of the initial attempts to address the first limitation is to schedule multiple restarts from different initial center positions, each sampled uniformly at random from dataset X and keep the solution with the minimum clustering error. However, it has been shown that this procedure does

✉ Aristidis Likas
arly@cs.uoi.gr

Georgios Vardakas
g.vardakas@uoi.gr

Ioannis Papakostas
ipapakostas@cs.uoi.gr

[1] Department of Computer Science and Engineering, University of Ioannina, Ioannina 45110, Epirus, Greece

Table 1 Definitions of symbols used in this paper

Symbol	Definition
X	Dataset
N	Number of samples
K	Number of clusters
L	Number of candidates
x_i	i_{th} data instance
μ_i	i_{th} center in Euclidean space
m_i	i_{th} center in feature space
ϕ	Kernel function
K	Kernel matrix
K_{ij}	$(i,j)_{\text{th}}$ element of K
C_i	i_{th} cluster
$\mathcal{C}_i$	i_{th} clustering solution
E	Clustering error function
P	Probability vector
p_i	Probability of i_{th} data instance
1	Indicator function
d_i	Distance of i_{th} data instance from its nearest center
c_i	i_{th} center candidate
$c(x_i)$	Cluster of x_i
M	Set of cluster centers
$\mathcal{X}$	Data space
$\mathcal{F}$	Feature space
$\mathcal{U}$	Uniform distribution
A	Affinity matrix
A_{ij}	$(i,j)_{\text{th}}$ element of A
G	Graph
$\mathcal{E}$	Set of edges
$\mathcal{A}, \mathcal{B}, \mathcal{V}, \mathcal{V}_i$	Set of nodes
D	Node degree matrix

Table 2 Acronyms of algorithms used in this paper

Acronym	Definition	References
GkM	Global k-means	[7]
GkM++	Global k-means++	[9]
GKkM	Global kernel k-means	[10]
RKkM	Kernel k-means with random uniform initialization	[5, 11]
KkM++	Kernel k-means++	This work
GKkM++	Global kernel k-Means++	This work

not produce satisfactory results [6]. The global k-means (GkM) [7] algorithm has also been introduced as a solution to the center initialization problem. This deterministic approach proposes a global optimization strategy that eliminates the dependence on random initialization of cluster centers. Instead of initializing all cluster centers simultaneously, GkM adopts an incremental procedure, aiming to optimally add a new cluster center at each stage k, leveraging the clustering solution $k-1$. Despite its effectiveness, a shortcoming of this method is its increased computational complexity. Another commonly used approach for center initialization is k-means++ [8], which has become widely used due to its effectiveness. This method determines initial cluster center positions by selecting data instances based on a probability distribution to maximize their separation. The algorithm follows a two-step process to initialize each cluster center: first calculating a probability distribution that guides center selection and then using this distribution to sample a data instance that determines the position of the initial center. This distribution is iteratively updated to determine the positions of subsequent centers, and the iterations are repeated until all K cluster centers are initialized. The acronyms of the algorithms studied in this paper are presented in Table 2.

Both GkM and k-means++ constitute well-studied approaches. To enhance computational efficiency, several variations of the GkM algorithm have been proposed [9, 12–15]. Similarly, numerous variations and theoretical analyses have been developed for the k-means++ algorithm [16–21]. Empirical evidence and theoretical results indicate that both the GkM and k-means++ methods significantly outperform standard k-means [15, 22–24].

To address the second limitation of the k-means (i.e. linear cluster separability), the kernel version of the k-means algorithm has been proposed [10, 11, 25–27]. The kernel k-means idea is that the data instances are first mapped from the input space to a higher-dimensional feature space using a nonlinear transformation. In this richer representation, the transformed data are expected to become linearly separable, making the use of the k-means algorithm more effective. Therefore, the k-means clustering error is minimized in this feature space, allowing for the identification of nonlinearly separable clusters in the input space, thus overcoming the second limitation. Spectral clustering constitutes an additional nonlinear approach to clustering, exploiting the eigenvectors of an affinity matrix constructed from the data [28, 29]. Notably, a direct relationship between kernel-based methods and spectral clustering has been established in [11]. In addition, deep learning integrated into clustering methodologies illustrate another recent approach to solving the second limitation of linear separability [30–36]. Although these methods have shown great potential, they often involve large-scale training and hyperparameter tuning (e.g., choice of network architecture, activation function).

However, introducing kernel-trick into the k-means procedure reintroduces the first limitation: how to properly initialize the centers in feature space? Inspired by the GkM method, the global kernel k-means (GKkM) has been proposed to address the initialization issue [10, 37]. Specifically, GKkM operates in a similar incremental fashion to GkM by solving all intermediate subproblems with $k = 1, \ldots, K$, utilizing kernel k-means as a local search

procedure. The intuition behind both algorithms is that a near-optimal solution with k clusters can be achieved by starting with a near-optimal solution for $k-1$ clusters and initializing the k_{th} cluster N times, each time starting from a different data instance position. The solution with the lowest clustering error is selected for the k-clustering problem. GKkM presents very satisfactory optimization capabilities, provides solutions for all $k \in \{1, \dots K\}$, and is also deterministic, since it does not depend on cluster initialization. Nevertheless, its primary limitation is its significant computational complexity, inherited from the GkM since it requires KN executions of the kernel k-means to solve the K-clustering problem. As a result of its high computational demands, GKkM is only applicable to small datasets, making it impractical for larger-scale clustering tasks.

Recently, the global k-means++ (GkM++) clustering algorithm [9] has been proposed to leverage the superior clustering solutions of the GkM algorithm while mitigating its substantial computational requirements. This is accomplished by exploiting the effective center initialization strategy of k-means++. Specifically, for each k-cluster subproblem, GkM++ appropriately chooses $L \ll N$ data instances as candidate initializations for the new center (instead of N, which GkM requires). Candidate selection is made using the k-means++ probability vector. This selection method is fast and does not require additional distance computations. Notably, GkM++ retains the effectiveness of the incremental clustering strategy of GkM while reducing the computational demands since a small number of appropriately selected initial center positions are examined.

In this work, inspired by the capabilities of the GkM++ method [9], we present the *global kernel k-means* ++ (GKkM++) algorithm, a novel approach to obtain superior kernel-based clustering solutions similar to those of GKkM at a lower computational cost. This improvement is achieved by integrating the effective center selection probability distribution of kernel k-means++ with the global kernel optimization strategy, allowing GKkM++ to efficiently explore the solution space while maintaining excellent optimization capabilities.

The proposed GKkM++ algorithm incrementally addresses all intermediate subproblems for $k = 1, \dots K-1$ in the kernel space, ultimately yielding the solution for K clusters. The core concept behind this method is that the optimal kernel k-means solution for K clusters can be derived through a sequence of local kernel k-means executions, appropriately initialized. In particular, during each local optimization with k clusters, the $k-1$ clusters are consistently initialized on the solution obtained from the previous problem with $k-1$ clusters. The newly added k_{th} cluster contains a single instance that is initialized at several starting positions determined by sampling from the k-means++ probability distribution in the kernel space. Complementary to the GKkM++ method, we also formulate the *kernel k-means* ++ (KkM++) to evaluate and compare the capabilities of both initialization strategies in the kernel space. Therefore, the key contributions of the paper are[1]:

- We propose GKkM++, a new algorithm for clustering in the feature space. The proposed algorithm constitutes an effective balance of the best properties of GKkM and (kernel) k-means++. It provides solutions with low clustering error (comparable to GKkM) at lower computational cost.
- We formulate the KkM++ method in order to evaluate and compare the capabilities of both initialization strategies in the kernel space.

The rest of this paper is structured as follows. In Sect. 2, we first provide a brief overview of the k-means and kernel k-means, followed by a discussion of some of the most popular variations for center initialization. Then, in Sect. 3, we formulate the kernel k-means++ method and introduce the global kernel k-means++. In Sect. 4, we present the results of an extensive comparative experimental study using both synthetic and real datasets. Additionally, we evaluate the proposed method in the context of the graph partitioning task (community detection). Finally, in Sect. 5, we provide conclusions and directions for future work.

2 Related work

This section presents related work, divided into two parts: k-means clustering in Euclidean space to introduce the clustering problem and kernel k-means for clustering in feature space, which is the focus of this paper.

2.1 *k*-means for clustering Euclidean space

k-means is an iterative algorithm that adjusts the positions of K cluster centers. In its simplest form (Lloyd's algorithm [5]), the K centers are randomly initialized through uniform sampling from the set of data instances $X = \{x_1, \dots, x_N\}$, where $x_i \in \mathbb{R}^d$. The algorithm follows a two-step iterative procedure until convergence: the cluster assignment step and the center update step. During the assignment step, each data instance x_i is assigned to the cluster C_j whose center μ_j is the nearest, i.e.

[1] Our implementations of the GKkM++ and KkM++ clustering algorithms are available in the following GitHub repository: https://github.com/gvardakas/global-kernel-k-means-pp.

$j = \arg\min_k ||x_i - \mu_k||^2$. At the center update step, each cluster center is set equal to the mean of the data instances assigned to its cluster $\mu_j = \frac{1}{|C_j|}\sum_{x_i \in C_j} x_i$. This simple iterative procedure converges to local minima of the clustering error but is highly sensitive to the initial placement of cluster centers. In contrast, the resulting clusters are limited to be linearly separable, preventing the algorithm potential from identifying complex cluster structures.

The *k-means*++ algorithm [8] is widely known as an effective method for the selection of the initial cluster positions with several variants [16–18, 38] and theoretical results [8, 19, 20]. Specifically, the *k*-means++ starts by determining the first initial center μ_1 through uniform random selection from the dataset. At each iteration k, assuming that a set of $k-1$ centers $\{\mu_1, \ldots, \mu_{k-1}\}$ have been determined, it computes the distances $d_i = \min_j ||x_i - \mu_j||^2$, for each data instance x_i from its nearest center to form the probability vector $P = (p_1, \ldots, p_N)$, where each component p_i is given by $p_i = Pr(\mu = x_i) = d_i / \sum_{j=1}^{N} d_j$. Subsequently, the next center position μ is determined by sampling a data instance using the probability vector P. The process of sampling and updating the probability vector continues until the K initial center positions have been specified. Finally, the standard *k*-means algorithm is applied to provide the clustering result.

The *global k-means* (GkM) clustering algorithm [7] is an incremental deterministic global optimization method that utilizes the *k*-means algorithm as a local search procedure. Instead of randomly choosing initial values for cluster centers, GkM incrementally adds a new cluster center at each stage in an attempt to place it optimally. To achieve this, the algorithm starts with one cluster $(k = 1)$ and solves the problem for the k clusters by building on the solution obtained for the $(k-1)$-cluster subproblem, thus sequentially addressing each intermediate subproblem with k clusters ($k \in \{1, \ldots, K\}$). Its main drawback lies in its significant computational complexity, requiring KN executions of *k*-means for the K-cluster problem.

In order to reduce the execution time of GkM, several methods have been proposed [7, 9, 12–15, 39]. Among these, the *global k-means* ++ (GkM++) clustering algorithm [9] is a recent approach that seeks to combine the strengths of both GkM and *k*-means++ methods. The key idea is to harness the high-quality clustering results of the GkM algorithm while reducing its significant computational burden by adopting the effective center selection mechanism of *k*-means++. More specifically, GkM++ inherits the incremental clustering strategy from GkM and adds one cluster center at a time. In order to solve the problem with the k-cluster problem, the $k-1$ cluster centers are initialized from the solution of the $(k-1)$-cluster problem. A number L of initial locations is considered for the k_{th} cluster center. Those locations are determined through sampling from the probability vector of *k*-means++. Thus, to solve the K-clustering problem, a total of LK *k*-means runs are required instead of NK, where $L \ll N$. This approach is both fast and computationally efficient, requiring minimal effort for distance computations.

2.2 Kernel *k*-means for clustering in feature space

The kernel *k*-means algorithm [25] extends the typical *k*-means algorithm to another space of higher dimension, called feature space $\mathcal{F}$. This is achieved by mapping original data instances using a non-linear function $\phi : \mathcal{X} \to \mathcal{F}$. In the transformed space, the algorithm minimizes the clustering error as presented in the following equation (weighted version):

$$E(\mathcal{C}) = \sum_{i=1}^{N}\sum_{k=1}^{K} 1_{C_k}(x_i) w_i ||\phi(x_i) - m_k||^2, \tag{2}$$

$$\text{where } m_k = \frac{\sum_{i=1}^{N} 1_{C_k}(x_i) w_i \phi(x_i)}{\sum_{i=1}^{N} 1_{C_k}(x_i) w_i}. \tag{3}$$

In the above equation, the kernel matrix is denoted by $\mathrm{K} \in \mathbb{R}^{N \times N}$, where each element $\mathrm{K}_{ij} = \phi(x_i)^T \phi(x_j)$ represents the inner product similarity in the feature space between instance i and j. In order for the kernel matrix to be valid, it should be positive semidefinite [26, 40].

By utilizing the kernel trick [25, 40], the squared distances in Eq. (2) can be computed without explicitly knowing the transformation ϕ, as shown in the following equation:

$$\begin{aligned} ||\phi(x_i) - m_k||^2 = \mathrm{K}_{ii} &- \frac{2\sum_{j=1}^{N} 1_{C_k}(x_j) w_j \mathrm{K}_{ij}}{\sum_{j=1}^{N} 1_{C_k}(x_j) w_j} \\ &+ \frac{\sum_{j=1}^{N}\sum_{l=1}^{N} 1_{C_k}(x_j) 1_{C_k}(x_l) w_j w_l \mathrm{K}_{jl}}{\sum_{j=1}^{N}\sum_{l=1}^{N} 1_{C_k}(x_j) 1_{C_k}(x_l) w_j w_l}. \end{aligned} \tag{4}$$

The kernel *k*-means does not explicitly use the transformation $\phi(x)$; therefore, the centers m_k of the clusters in the feature space cannot be computed directly. However, the values of the kernel matrix provide the information required to compute the feature space distance between instances and cluster centers. Kernel *k*-means can achieve non-linear cluster separation, thereby addressing the second *k*-means limitation. Unless specified otherwise, we assume the unweighted version of the algorithm, i.e. $w_i = 1, i = 1, \ldots, N$.

Require: **K**: Kernel matrix
Require: K: Number of clusters
Require: $\{C_1, C_2, \ldots, C_K\}$: Cluster Initialization ▷ Typically random initialization
Initialize *converged* ← False
while not *converged* **do**
  for $k = 1, \ldots, K$ **do**
    for all $x_i \in X$ **do**
      Compute $\|\phi(x_i) - m_k\|^2$ using eq. 4
    end for
  end for
  for all $x_i \in X$ **do**
    Assign x_i to the cluster with the nearest center $c^\star(x_i) \leftarrow \arg\min_k ||\phi(x_i) - m_k||^2$
  end for
  for $k = 1, \ldots, K$ **do**
    Update cluster $C_k \leftarrow \{x_i | c^\star(x_i) = k\}$
  end for
  if no change in cluster labels **then**
    converged ← True
  end if
end while
return final partition $\mathcal{C}^\star = \{C_1, C_2, \ldots, C_K\}$ and clustering error $E(\mathcal{C}^\star)$ using eq. 2

Algorithm 1 Kernel k-Means

It is known that given a positive semidefinite kernel matrix, the kernel k-means monotonically converges to a local minimum of clustering error in the feature space. In this case, the algorithm's computational complexity is $\mathcal{O}(N^2\tau)$, where τ denotes the number of iterations to convergence [41]. It is important to point out that applying kernel k-means does not require the data vectors themselves, but only the values of the kernel matrix K. The detailed kernel k-means algorithm is described in Algorithm 1. It should be emphasized that the kernel k-means requires an initial partition to be specified. At each iteration, the partition is updated in order to reduce the clustering error. It should be noted that in most software packages, the standard method for initializing kernel k-means is either random initialization [5] or random instance labeling, also known as Forgy's method [42]. It is widely known that both strategies very often lead to convergence at poor suboptimal solutions.

Require: **K**: Kernel matrix
Require: K: Number of clusters
Initialize $\mathcal{C}_1^\star \leftarrow X$
for $k = 2, \ldots, K$ **do**
  for all $x_n \in X$ **do**
    $C'_k \leftarrow \{x_n\}$ ▷ Initialization of the new cluster C'_k using data instance x_n
    $\mathcal{C}'_{k-1} \leftarrow \mathcal{C}^\star_{k-1} / \{x_n\}$ ▷ Exclude x_n from the $k-1$ solution
    $\mathcal{C}'_k \leftarrow \mathcal{C}'_{k-1} \cup C'_k$ ▷ Initialization of partition $\mathcal{C}'_k$
    $(\mathcal{C}^n_k, E(\mathcal{C}^n_k)) \leftarrow$ Run `kernel k-means`$(\mathbf{K}, k, \mathcal{C}'_k)$ (Alg. 1) ▷ Solution with k clusters
  end for
  $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k)) \leftarrow$ Solution with the minimum error $E(\mathcal{C}^\star_k)$ among the N partitions $\mathcal{C}^n_k$
end for
return solutions $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k))$ for every $k \in \{1, \ldots, K\}$

Algorithm 2 Global Kernel k-Means

The *global kernel k-means* (GKkM) [10] is an extension of the GkM algorithm for feature space clustering error minimization (Eq. 2). GKkM employs kernel k-means as a local minimization procedure and operates incrementally, solving all subproblems for $k = 1, \ldots, K$ successively. The underlying idea is that a near-optimal solution for k clusters can be attained by first obtaining a near-optimal solution for $k-1$ clusters and then initializing the k_{th} cluster N times, with each initialization starting from a different data instance. Among the N solutions, the one with the smallest clustering error is selected for the k-clustering problem. GKkM presents very satisfactory optimization capabilities, provides all solutions for all $k \in \{1, \ldots K\}$, and is also deterministic since it does not depend on initialization. However, its primary limitation is the high computational complexity inherited from GkM. If the final number of clusters is K, then KN kernel k-means executions are required, resulting in an overall complexity of $\mathcal{O}(N^3K\tau)$, assuming the kernel matrix has been precomputed [10].

The details of GKkM are outlined in Algorithm 2. Specifically, the algorithm requires the number of clusters K and a kernel matrix K as input. It should be noted that C_k refers to the k_{th} cluster in the partition, while $\mathcal{C}_k$ denotes the entire clustering solution for k clusters. At first, the method addresses the kernel 1-means subproblem by setting the clustering solution $\mathcal{C}_1$ to represent the entire dataset X (Step 1 in Algorithm 2). Next, to tackle the kernel 2-means

subproblem, kernel k-means is executed N times. In each of these N executions, the second cluster, C_2', is initialized using a data instance x_n, which is removed from the previously obtained clustering solution $\mathcal{C}_1'$ (Steps 4 and 5 in Algorithm 2). The initialization for two clusters is constructed by combining the solution $\mathcal{C}_1'$ with the single-point cluster C_2' (Step 6 in Algorithm 2). The Kernel k-means then further optimizes the clustering objective using this initialization (Step 7 in Algorithm 2). After all data instances have been considered as possible initializations for the second cluster (i.e., kernel k-means has been executed N times), the lowest error solution is selected as the final solution for the kernel 2-means problem (Step 9 in Algorithm 2). This process is repeated incrementally for every $k = 2, \ldots, K$, each time exploiting the solution with $(k-1)$ clusters. Finally, the algorithm provides solutions for all $k = 1, \ldots, K$.

3 Method

This section presents the methodology developed to minimize the feature space clustering error (Eq. 2) while mitigating the computational requirements. Specifically, in Sect. 3.1, we introduce the formulation of the kernel k-means++ method. Furthermore, in Sect. 3.2, we present the global kernel k-means++ algorithm, which offers enhanced optimization capabilities and reduced computational requirements compared to its predecessor, the GKkM method. Finally, in Sect. 3.3, we provide results on computational complexity.

3.1 Kernel *k*-means++

As previously mentioned, k-means++ [8] is a successful algorithm for selecting initial center positions in Euclidean space, offering provable guarantees. The intuition behind the k-means++ initialization algorithm is to select a set of well-dispersed initial centers throughout the dataset. This makes the algorithm less likely to converge to poor local minima, leading to improved clustering performance. Thus, we aimed to evaluate its performance in feature space clustering when used as an initialization method for kernel k-means. Therefore, we study the *kernel k-means* ++ (KkM++) algorithm that can be derived from k-means++ using the feature space distance formulation described in Algorithm 3.

Require: $\mathbf{K}$: Kernel matrix
Require: K: Number of clusters
1: Initialize the set of cluster centers $M \leftarrow \{\}$
2: Initialize the clusters $\mathcal{C} = \{C_1, C_2, \ldots, C_K\}$ where $C_i \leftarrow \{\}$, $\forall i = 1, 2, \ldots, K$
3: Choose a data instance $\mu_1 \sim \mathcal{U}(X)$
4: $M \leftarrow M \cup \{\mu_1\}$
5: **for** $k = 2, \ldots, K$ **do**
6: **for all** $x_i \in X$ **do**
7: Compute distance $d_i \leftarrow \min_j d\left(\phi(x_i), \phi(\mu_j)\right)$ using eq. 5
8: **end for**
9: Choose a data instance $\mu_k \leftarrow x_i$ at random from X with probability $p_i \leftarrow \frac{d_i}{\sum_{j=1}^{n} d_j}$
10: $M \leftarrow M \cup \{\mu_k\}$
11: **end for**
12: **for all** $x_i \in X$ **do**
13: Assign x_i to the cluster with the nearest center $c^\star(x_i) \leftarrow \arg\min_k d\left(\phi(x_i), \phi(\mu_k)\right)$ using eq. 5
14: **end for**
15: **for** $k = 1, \ldots, K$ **do**
16: Compute initial cluster $C_k \leftarrow \{x_i | c^\star(x_i) = k\}$
17: **end for**
18: **return** clustering initialization $\mathcal{C}^\star = \{C_1, C_2, \ldots, C_K\}$

Algorithm 3 Kernel k-Means++ Initialization

In the following formulation, μ and m represent a cluster center in Euclidean and feature spaces, respectively. Specifically, KkM++ requires the number of clusters K and a kernel matrix K as input. The algorithm begins by initializing the first cluster center μ_1 by randomly selecting a data instance x_i from X (Step 3 in Algorithm 3). For each data instance x_i, it computes the distance $d_i = \min_k ||\phi(x_i) - m_k||^2$, where m_k represents the closest cluster center in feature space, defined as $m_k = \phi(\mu_k)$ (Steps 6–8 in Algorithm 3). The probability vector $P = (p_1, \ldots, p_N)$ is then defined, where each component p_i is given by $p_i = Pr(m = \phi(x_i)) = d_i / \sum_{j=1}^{N} d_j$ (Step 9 in Algorithm 3). Since each cluster center $m_k = \phi(\mu_k) = \phi(x_j)$ is initialized by a sampled data instance x_j from the dataset X, the distance is computed using kernel matrix values:

$$\begin{aligned} d\left(\phi(x_i), m_k\right) &= d\left(\phi(x_i), \phi(\mu_k)\right) = ||\phi(x_i) - \phi(x_j)||^2 \\ &= \mathrm{K}_{ii} - 2\mathrm{K}_{ij} + \mathrm{K}_{jj}. \end{aligned} \tag{5}$$

Subsequently, the next center m is sampled from the dataset based on the multinomial distribution defined by the probability vector P, and the process is repeated until all K centers are initialized (Steps 5–11 in Algorithm 3). Note that even though we do not have direct access to the data instances $\phi(x)$ or cluster centers m in the feature space, the described process allows us to compute the distribution of distances between each data instance and center in the feature space, thus enabling us to sample the next cluster center directly from X.

3.2 Global kernel *k*-means++

GKkM (Algorithm 2) constitutes a deterministic method that aims to tackle the initialization issue of kernel k-means but at a high computational cost. It operates incrementally and provides feature space partitions into K clusters by sequentially addressing every k-cluster subproblem $k = 1, \ldots, K$. To address the problem with k clusters, N kernel k-means executions are implemented, where each of the N data instances is considered as the initial position for the new cluster center in the feature space. This exhaustive consideration of initial positions for the new cluster center results in an effective placement; thus, superior clustering solutions are obtained. However, exhaustive search limits the algorithm's applicability to relatively small datasets.

achieves this by using an efficient candidate selection strategy that prioritizes sampling from regions without existing cluster centers in the feature space. Specifically, the method employs the KkM++ instance selection probability distribution to guide the selection process. The main objective of this approach is to obtain enhanced optimization capabilities at low computational complexity. The GKkM++ algorithm is presented in detail in Algorithm 4.

The algorithm requires as input the kernel matrix K, the final cluster number K, the number of candidates L, and the strategy S used to sample candidates: either *batch* or *sequential*. Note that $D = (d_1, \ldots, d_N)$ denotes the vector of distances d_i, with d_i being the distance of $\phi(x_i)$ from its nearest center m_j in the feature space. The method follows the steps outlined below. First, it computes the solution to the kernel 1-means subproblem by setting the first cluster to contain all data instances. It also initializes the distances d_i (steps 1–2 in Algorithm 4). Then, in order to solve the 2-cluster subproblem, the kernel 1-means solution (obtained in step 1) is utilized as the first initial cluster. Then the set of L candidate instances $\{c_1, \ldots, c_L\}$ is formed through sampling using the batch or sequential strategy (steps 4 in Algorithm 4) in order to determine candidate initialization for the second cluster. Since we do not assume direct access to $\phi(c_\ell)$, the GKkM++ method, similar to GKkM, initializes the candidate c_ℓ as a new cluster containing a single

Require: $\mathbf{K}$: Kernel matrix
Require: K: Number of clusters
Require: L: Number of candidates
Require: $S \in \{$`'Batch'`, `'Sequential'`$\}$: Sampling Method
1: Initialize $\mathcal{C}_1^\star \leftarrow X$
2: $D \leftarrow (||\phi(x_1) - m_1||^2, \ldots, ||\phi(x_N) - m_1||^2)$ using eq. 4 ▷ Distance vector
3: **for** $k = 2, \ldots, K$ **do**
4: $\{c_1, \ldots, c_L\} \leftarrow$ `Candidate Selection`$(S, \mathbf{K}, L, D)$
5: **for all** $c_\ell \in \{c_1, \ldots, c_L\}$ **do** ▷ L kernel k-means executions
6: $C'_k \leftarrow \{c_\ell\}$ ▷ Initialization of the new cluster C'_k using data instance c_ℓ
7: $\mathcal{C}'_{k-1} \leftarrow \mathcal{C}^\star_{k-1} / \{c_\ell\}$ ▷ Exclude c_ℓ from the $k-1$ solution
8: $\mathcal{C}'_k \leftarrow \mathcal{C}'_{k-1} \cup C'_k$ ▷ Initialization of clustering $\mathcal{C}'_k$
9: $(\mathcal{C}^\ell_k, E(\mathcal{C}^\ell_k)) \leftarrow$ Run `kernel k-means`$(\mathbf{K}, k, \mathcal{C}'_k)$ (Alg. 1) ▷ Solution with k clusters
10: **end for**
11: $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k)) \leftarrow$ Solution with the minimum error $E(\mathcal{C}^\star_k)$ among the L partitions $\mathcal{C}^\ell_k$
12: **for all** $x_i \in X$ **do** ▷ Update distance vector
13: $D[i] \leftarrow \min_j(||\phi(x_i) - m_j||^2)$ using eq. 4 ▷ Pre-computed distances in step 9
14: **end for**
15: **end for**
16: **return** solutions $(\mathcal{C}^\star_k, E(\mathcal{C}^\star_k))$ for every $k \in \{1, \ldots, K\}$

Algorithm 4 Global Kernel k-Means++

To preserve the optimization capabilities of the greedy sequential strategy of GKkM, and inspired by the GkM++ method, we introduce the *global kernel k-means* ++ (GKkM++) method for feature space clustering. The key difference is that, instead of evaluating every data instance $x_n \in X$ as a potential candidate for the new center, the proposed method considers only L appropriately selected candidates. Therefore, only L kernel k-means runs are required for each k-cluster subproblem, with $L \ll N$. GKkM++ element, bypassing this problem (steps 6–8 in Algorithm 4). Next, L executions of kernel k-means until convergence take place, one for each initial cluster $\{c_\ell\}$ (steps 5–10 in Algorithm 4). From the L solutions found, we select the one with the minimum clustering error value (step 11 in Algorithm 4). Finally, we update the distances d_i (steps 12–14 in Algorithm 4). In the same spirit, for each $k = 2, \ldots, K$, the method progressively solves the k-cluster subproblem by exploiting the solution of the $(k-1)$-cluster subproblem.

To compute the solution with k clusters, the $k-1$ clusters are initialized using the already found partition of the $(k-1)$-cluster subproblem. Then, the L candidate instances are sampled (using either the batch or sequential strategy discussed next) to select the element that will be used to initialize the k_{th} cluster. The final output of the algorithm is a clustering solution for all $k = 1, \ldots, K$. It is important to note that steps 5–10 of Algorithm 4 could be executed in parallel, providing significant speedup (up to L times faster). For further speed up kernel k-means execution, MapReduce schemes [43], and coreset [44] could also be applied.

The initial center candidates are sampled using the probability distribution P, which expresses the feature space distance of the instances from the closest cluster center. As presented in Algorithm 5, two sampling strategies have been considered. *Batch sampling* (steps 2–3 in Algorithm 5) is simpler since it exploits only the pre-computed distances d_i, $i = 1, \ldots, N$. Using these distances, the selection probability p_i for data instance x_i (step 2 in Algorithm 5) is computed and the probability vector $P = (p_1, \ldots, p_N)$ is formed. Probability p_i is inversely proportional to distance d_i, so data instances far (in feature space) from the current cluster centers are more likely to be selected. Then, using the computed kernel k-means++ probability vector P, a set of L candidates $\{c_1, \ldots, c_L\}$ are selected by sampling without replacement (step 3 in Algorithm 5).

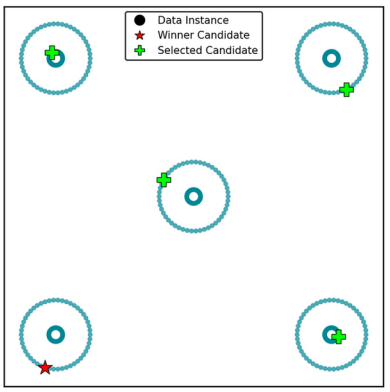
Require: $\mathbf{K}$: Kernel matrix
Require: L: Number of candidates
Require: $D = (d_1, \ldots, d_N)$: Vector of minimum instance to cluster center distances in feature space
1: **if** $S =$ '`Batch`' **then** ▷ Batch Sampling Strategy
2: Compute the probability vector $P \leftarrow (p_1, \ldots p_N)$, where $p_i = d_i / \sum_{j=1}^{N} d_j$
3: $\{c_1, \ldots, c_L\} \leftarrow$ Sample without replacement L candidates
4: **else** ▷ Sequential Sampling Strategy
5: **for** $\ell = 1, \ldots, L$ **do**
6: Compute the probability vector $P = (p_1, \ldots p_N)$, where $p_i = d_i / \sum_{j=1}^{N} d_j$
7: $c_\ell \leftarrow$ Sample without replacement one candidate instance c_ℓ from X using P
8: $\Phi_c^{(\ell)} \leftarrow \{c_1, \ldots, c_{\ell-1}\} \cup \{c_\ell\}$
9: **for all** $x_i \in X$ **do**
10: $d_i \leftarrow \min_{r_j \in M \cup \Phi_c^{(\ell)}} ||\phi(x_i) - r_j||$ using eq. 4
11: **end for**
12: **end for**
13: **end if**
14: **return** Set of initial center candidates $\{c_1, \cdots, c_L\}$

Algorithm 5 Candidate Selection

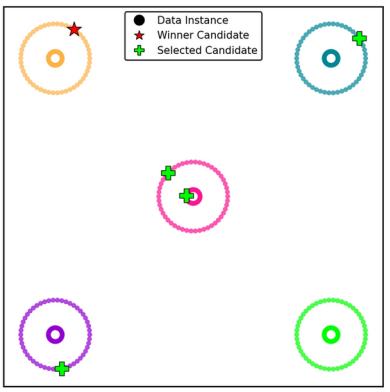
Alternatively, a sequential sampling approach can be used in which each candidate instance is sequentially sampled. In this case, the distribution P is updated so that instances are selected to be distant not only from the current cluster centers but also from the already selected instances (step 7 in Algorithm 5). Therefore, after each sampling, the distance d_i of each data instance $\phi(x_i)$ is updated accordingly (steps 9–11 in Algorithm 5) to take into account the newly selected instance. After L sampling steps, the set $\{c_1, \ldots, c_L\}$ of candidates has been determined (steps 5–12 in Algorithm 5). While the sequential sampling method requires recomputing the minimum distance values, it results in a more effective distribution of the L candidates within the feature space. It is important to note that at the beginning of the sequential sampling strategy, the relation $d_i = \min_{r_j \in M} ||\phi(x_i) - r_j||$ holds. Consequently, the computation in step 10 only needs to consider the set $\Phi_c^{(\ell)}$, which reduces the distance computation to $d_i = \min_\ell \{d_i, |\phi(x_i) - \phi(c_\ell)|^2\}$.

Figure 1 demonstrates a practical example of the proposed GKkM++ algorithm using a synthetic two-dimensional dataset with 10 clusters arranged in five pairs of concentric rings. The dots represent the data instances, while the color indicates the respective cluster assignments determined by the algorithm. Green crosses mark the candidate positions for initializing the next cluster. Additionally, the red star denotes the winner candidate corresponding to the best initialization. In Fig. 1a–d, the method addresses a kernel k-means subproblem with different number of clusters k. Based on each subproblem solution, the next center candidates are sampled from the KkM++ distribution. Specifically, in Fig. 1a, the kernel 1-means subproblem is initially solved by considering the entire dataset as a single cluster (denoted in blue), and the method successfully samples the next candidate initial center positions across multiple distinct clusters in the dataset. This effective behavior is consistently observed in Fig. 1b and Fig. 1c until the final clustering with $k = 10$ clusters is achieved in Fig. 1d.

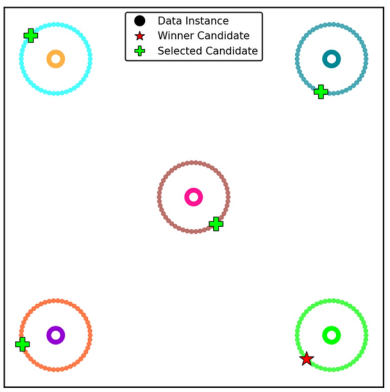
In Fig. 2 (left), we present the solution when the method is halted at $k = 8$. We observe that 6 of the 10 clusters have been correctly identified, while the remaining four require further refinement. At this stage, we expect data instances located farther from any cluster center to be more likely to be selected than those closer to the centers. On the right



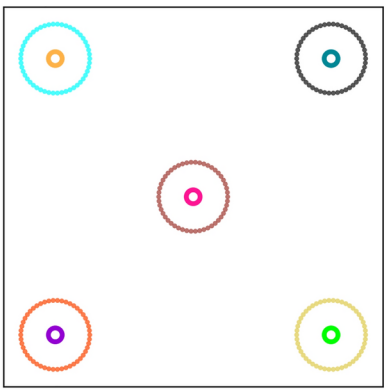
(a) Solution with $k = 1$ cluster.



(b) Solution with $k = 5$ clusters.

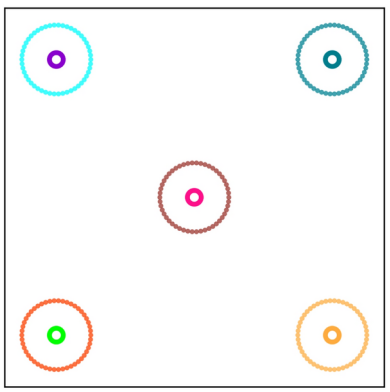


(c) Solution with $k = 8$ clusters.

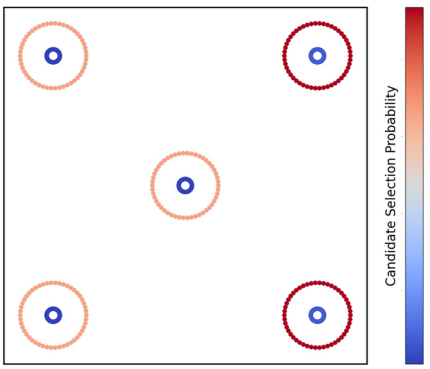


(d) Final solution with $k = 10$ clusters.

Fig. 1 Illustrative example of GKkM++ execution. Data instances are denoted with circles, red crosses indicate cluster centers and red star denotes the winner candidate corresponding to the best initialization



(a) Solution for $k = 8$ clusters.



(b) Heatmap with candidate selection probabilities.

Fig. 2 Illustratve example of GKkM++ execution. Circles denote the data instances

side of Fig. 2, a heatmap illustrates the selection probability of each data instance as a candidate for initializing the next cluster in the $k = 9$ subproblem. In the heatmap, dark blue indicates low selection probability, while red indicates high selection probability. As observed, data instances in the two outer rings, which do not form a cluster independently, exhibit a significantly higher selection probability. Conversely, data instances in the inner rings have a low selection probability, as their proximity to a cluster center in the feature space reduces the likelihood of their selection. Lastly, the data instances in outer rings that already form distinct clusters have a lower selection probability than those in outer rings that have not been partitioned.

Table 3 Kernels used in our experimental evaluation

Kernel type	Kernel function
Cosine Kernel	$\mathrm{K}(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x_i} \cdot \mathbf{x_j}}{\|\mathbf{x_i}\| \|\mathbf{x_j}\|}$
Polynomial Kernel	$\mathrm{K}(\mathbf{x_i}, \mathbf{x_j}) = (\gamma\,(\mathbf{x_i} \cdot \mathbf{x_j}) + c_0)^{deg}$
RBF Kernel	$\mathrm{K}(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|_2^2\right)$

3.3 Complexity analysis

Since GKkM++ is closely related to GkM++ and kernel k-means, it inherits their computational complexity. Specifically, the complexity of kernel k-means is $\mathcal{O}(N^2\tau)$, where τ represents the number of iterations until convergence. Assuming K clusters, GkM++ requires $\mathcal{O}(KL)$ executions of kernel k-means. Consequently, the computational complexity of GKkM++ is $\mathcal{O}(N^2KL\tau)$, assuming access to the kernel matrix K, where $L \ll N$. This is a significant speedup compared to GKkM, which has a $\mathcal{O}(N^3K\tau)$ complexity. It should be noted that we have empirically observed that kernel k-means tends to converge faster (in fewer iterations) as k increases in this sequential clustering framework. This speedup is reasonable because, when solving each k-cluster subproblem, the method utilizes the solution from the $(k-1)$-cluster subproblem, which is already well-positioned. As a result, the number of iterations τ required for convergence typically decreases as k increases, thereby improving the algorithm's overall efficiency.

As mentioned previously, the batch sampling strategy does not require extra distance computations. Sequential sampling adds $\mathcal{O}(NL)$ distance computations, similar to KkM++ for sampling the K cluster centers. It should be noted that an additional advantage emerges from the incremental solution of the kernel k-means problem: in many cases, the number of clusters is not given. Therefore, it is necessary to obtain solutions for a range of k values. Those solutions will be subsequently evaluated using clustering quality criteria (e.g. silhouette score [45], modularity [46], inclusion [47, 48] etc.) for cluster number estimation [22].

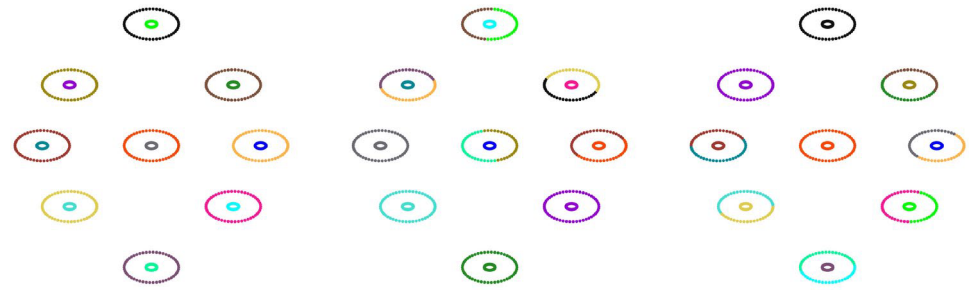
4 Experiments

We have conducted an extensive series of experiments to assess the performance of the proposed KkM++ and GKkM++ methods, the latter both with batch (b) and sequential (s) sampling strategy. The methods were compared against GKkM and RKkM using the kernels of Table 3.

The experimental evaluation is divided into three subsections. In Sect. 4.1, we demonstrate the clustering performance of each algorithm on synthetic two-dimensional datasets that are not linearly separable. Section 4.2 focuses on evaluating the algorithms on graph partitioning tasks, while in Sect. 4.3, real-world datasets are considered.

4.1 Synthetic data demonstration

At first, we considered two challenging synthetic datasets that are not linearly separable, as shown in Figs. 3 and 4, respectively. Specifically, the first synthetic dataset (Fig. 3) consists of nine pairs of concentric rings, each containing 50 data instances, resulting in a total of 900 points evenly



(a) GKkM++(b&s)/GKkM (b) RKkM (c) KkM++

Fig. 3 Clustering results for the 18 rings dataset

(a) GKkM++(b&s)/GKkM (b) RKkM (c) KkM++

Fig. 4 Clustering results for the three rings with six Gaussians dataset

Table 4 Clustering error comparison of the evaluated methods on 2D synthetic datasets

Dataset	RKkM	KkM++	GKkM	GKkM++ (b)	GKkM++ (s)
18 Rings	362.56	361.66	345.10	345.10	345.10
3 Rings & 6 Gaussians	128.02	127.21	121.75	121.75	121.75

distributed across 18 clusters. The second dataset (Fig. 4) comprises three rings and six Gaussian distributions, with each cluster containing 50 data instances. This results in a total of 450 points uniformly distributed across all nine clusters. In both cases, the RBF kernel was employed. Algorithms such as *k*-means, which rely on identifying linearly separable clusters in the data space, are therefore unsuitable for these datasets.

In our synthetic data demonstration, the GKkM is executed once since it produces deterministic clustering results. In contrast, RKkM and KkM++ were employed 100 times, from which we report the minimum clustering error, similar to the experimental setup of survey [22]. Furthermore, for the variants of GKkM++, we set the number of cluster candidates to $L = 100$ (equal to the number of restarts used for the rest of kernel *k*-means variants). We conducted the experiment a single time, as practiced in [9]. Finally, the quality of the solutions produced by the algorithms is assessed through clustering error (Eq. 2) and through visual inspection as shown in Figs. 3 and 4.

In these challenging clustering tasks, GKkM accurately identifies all clusters. Similarly, GKkM++ with sequential and batch sampling also successfully partitions both datasets, demonstrating the effectiveness of both sampling procedures. The solutions produced by GKkM and GKkM++ variants achieve the lowest clustering errors, as shown in Table 4. On the other hand, RKkM and KkM++ fail to identify all clusters in both datasets correctly. However, as shown in Table 4, KkM++ outperforms RKkM in both synthetic datasets in terms of clustering error.

4.2 Graph partitioning

Graph partitioning represents a different approach to data clustering. In this context, we are provided with a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of vertices and edges, respectively. Our goal is to partition the graph into disjoint clusters that satisfy specific conditions. Several objectives for graph partitioning have been proposed, including ratio association, ratio cut, normalized cut, and others. Spectral methods are commonly used to address these problems by computing the eigenvectors of the affinity matrix [28]. However, eigenvector computation is computationally intensive, requiring $\mathcal{O}(N^3)$ operations, and may become impractical for extensive graphs.

Table 5 Descriptions of utilized graphs

Graph	Description	# Nodes	# Edges	Source
EMAIL-EU-I-TD1	Email communication among first department members	309	1938	[51]
EMAIL-EU	Email exchanges among all institution members	1005	16,706	[51]
GRQC	Co-authorship in general relativity and quantum cosmology	5242	14,496	[51]

It is known that the kernel *k*-means objective is equivalent to graph cut objective once the kernel matrix is appropriately defined [11, 41, 49]. This proof is established by formulating the problem as trace maximization, following a similar methodology to that in [50], where the *k*-means objective is also framed as trace maximization. Kernel *k*-means bypasses the requirement to compute eigenvectors; however, it cannot ensure an optimal solution due to its dependence on cluster initialization. Even when eigenvector computations are feasible, experiments in [41] indicate that kernel *k*-means can further enhance the clustering results obtained from spectral methods.

GKkM can be effectively utilized for relatively small graph partitioning tasks [10]. Naturally, KkM++ and GKkM++ can also address the graph partitioning problem, as they are built upon the foundations of kernel *k*-means and GKkM, respectively. In the following experiments, we demonstrate the performance of each clustering method on the graph partitioning task using three graphs of increasing difficulty (Table 5). We evaluate the performance of the compared methods for a maximum number of clusters (or communities), denoted by K, and consider each solution for $k = 1, \ldots, K$. In all experiments, we fix $K = 50$. To ensure consistency in the comparison, each subproblem with k clusters is solved using $L = 100$, where L represents either the number of restarts or the number of candidate solutions explored. Specifically, for KkM++ and RKkM, L defines the number of restarts. For GKkM++, L specifies the number of candidates selected. Finally, due to its deterministic nature, GKkM is executed once.

4.2.1 Graph datasets

To evaluate the effectiveness of the proposed algorithms, we conducted a series of experiments using freely accessible graphs. We intentionally selected graphs that exhibit diverse characteristics, including the number of nodes and edges (see Table 5).

The EMAIL-EU-l-TD1 graph captures email communication exclusively among members of the first department within a European research institution, with edges indicating the sender–receiver relationships in both directions. In addition, EMAIL-EU graph is constructed from anonymized email data collected from the same institution, reflecting all incoming and outgoing communications among its members. An edge (u, v) exists in this graph if person u has sent at least one email to person v, thus representing communication solely within the institution, while excluding interactions with external entities. Finally, the GRQC collaboration graph illustrates scientific collaborations among authors with papers in the General Relativity and Quantum Cosmology category. An undirected edge exists between authors i and j if they co-authored a paper together; if a paper has k co-authors, it generates a fully connected subgraph of k nodes.

4.2.2 Graph partitioning evaluation

Let us define $links(\mathcal{A}, \mathcal{B})$ as the cumulative weight of the edges between the nodes in sets $\mathcal{A}$ and $\mathcal{B}$ as $links(\mathcal{A}, \mathcal{B}) = \sum_{i\in\mathcal{A}} \sum_{j\in\mathcal{B}} \mathrm{A}_{ij}$, where A is the affinity matrix that captures the pairwise similarities among the vertices. Similarly, let $degree(\mathcal{A})$ represent the sum of the edge weights between the nodes in $\mathcal{A}$ and all vertices, expressed as $degree(\mathcal{A}) = links(\mathcal{A}, \mathcal{V})$, where $\mathcal{V}$ is the set of all vertices. Let D be the diagonal $|\mathcal{V}| \times |\mathcal{V}|$ degree matrix, where $\mathrm{D}_{ii} = \sum_{j=1}^{|\mathcal{V}|} \mathrm{A}_{ij}$. Next, we define the graph partitioning objectives optimized in the experimental section: the ratio association and the normalized cut.

1. Ratio Association problem aims to maximize the internal connectivity of clusters in proportion to their size, and its objective function is presented in the following equation:

$$RA(G) = \max_{\mathcal{V}_1,\ldots,\mathcal{V}_M} \sum_{i=1}^{M} \frac{\mathrm{links}(\mathcal{V}_i, \mathcal{V}_i)}{|\mathcal{V}_i|}. \tag{6}$$

To align the objective function of the weighted kernel k-means algorithm with the ratio association problem, we set $w_i = 1$, $w_j = 1$, $w_l = 1$ and $\mathrm{K} = \mathrm{A}$ of the Eqs. (2) and (4). This makes the problem equivalent to the unweighted kernel k-means, where the affinity matrix is treated as the kernel matrix [41, 49].

2. Normalized Cut problem seeks to minimize the cut between clusters and the rest of the graph relative to the cluster's degree [52, 53]. This objective is widely used in graph partitioning, and its formulation is the following:

$$NC(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_M} \sum_{i=1}^{M} \frac{\mathrm{links}(\mathcal{V}_i, \mathcal{V}/\mathcal{V}_i)}{\mathrm{degree}(\mathcal{V}_i)}. \tag{7}$$

To transform the objective function of weighted kernel k-means to correspond to that of the normalized cut, we need to set $w_i = \mathrm{D}_{ii}$, $w_j = \mathrm{D}_{jj}$, $w_l = \mathrm{D}_{ll}$, and $\mathrm{K} = \mathrm{D}^{-1}\mathrm{A}\mathrm{D}^{-1}$ [41, 49].

The previously discussed definitions of the kernel matrix do not guarantee that it will be positive semidefinite, which is essential for its validity in our algorithms. Although being positive semidefinite is a sufficient condition, it is not necessary for the convergence of the examined algorithms. A solution to this problem is proposed in [41], which involves applying a diagonal shift to the kernel matrix. To address the ratio association problem, we define $\mathrm{K} = \lambda\mathrm{I} + \mathrm{A}$, where I represents the identity matrix and λ is a sufficiently large constant to guarantee that K is positive semidefinite. Additionally, to tackle the normalized cut problem, we formulate $\mathrm{K} = \lambda\mathrm{D}^{-1} + \mathrm{D}^{-1}\mathrm{A}\mathrm{D}^{-1}$. A notable point is that this adjustment to the kernel matrix does not alter the goal of the problem. However, as demonstrated in [41], it may adversely affect the performance of the algorithms if the shift λ is excessive [10].

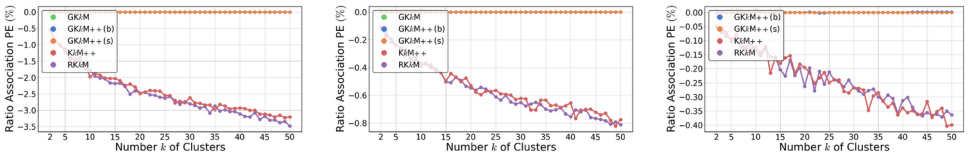
In our study, clustering is framed as an optimization problem. Our aim is to obtain solutions of minimum error in the feature space. Therefore, we evaluate the performance of each method using the clustering error. Minimizing clustering error is equivalent to maximizing the ratio association (Eq. 6) in the first case and minimizing the normalized cut (Eq. 7) in the second case. For performance comparison we calculate the relative Percentage Error:

$$\mathrm{PE} = \frac{E(\mathcal{C}_k) - E(\mathcal{C}_k^\star)}{E(\mathcal{C}_k^\star)} \times 100\%, \tag{8}$$

where $E(\mathcal{C}_k^\star)$ represents the clustering error of the baseline method, which we defined as GKkM, and $E(\mathcal{C}_k)$ denotes the error produced by each of the compared methods (Figs. 5, 6). However, in the case of GRQC where GKkM did not provide solutions in reasonable time due to its high computational burden, we utilized the GKkM++ with sequential sampling strategy as a baseline method.

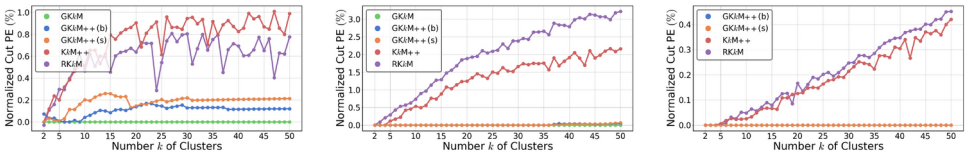
4.2.3 Graph partitioning experimental results

As it can be observed from the experimental results (Figs. 5, 6), GKkM++ produces results comparable to GKkM (with a Percentage Error smaller than 0.05 in most cases) which consistently achieves the best performance in both the ratio association and normalized cut objectives. Note that the ratio association objective is meant to be



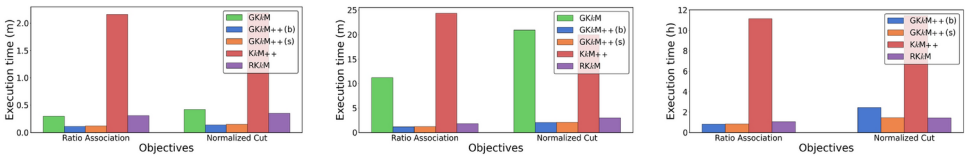
(a) EMAIL-EU-TD1 (b) EMAIL-EU (c) GRQC

Fig. 5 Relative percentage error in the ratio association objective across different graphs



(a) EMAIL-EU-TD1 (b) EMAIL-EU (c) GRQC

Fig. 6 Relative percentage error in the normalized cut objective across different graphs



(a) EMAIL-EU-TD1 (b) EMAIL-EU (c) GRQC

Fig. 7 CPU time comparison across different datasets and problems

maximized (higher values are better), while the normalized cut objective should be minimized (lower values are better). GKkM failed to complete within a reasonable time on the GRQC graph due to its high computational demands, while in such cases, GKkM++ produced the best outcomes. Additionally, the two GKkM++ variants consistently outperform KkM++ and RKkM. Notably, as the value of k increases, GKkM++ demonstrates progressively superior performance compared to the other two algorithms across both types of problems. This is particularly significant because higher values of k increase the complexity of optimizing the ratio association and normalized cut objectives.

In an attempt to compare KkM++ with RKkM on the ratio association objective, we observe that both algorithms exhibit similar behavior (Fig. 5). When addressing the normalized cut objective (Fig. 6), we notice that RKkM outperforms KkM++ on the EMAIL-EU-l-TD1 graph. In the other two graphs, which contain a larger number of nodes and thus increase the complexity of the problem, KkM++ consistently delivers better graph partitioning results. Notably, in the EMAIL-EU graph, as the value of k increases, the quality of KkM++ solutions becomes increasingly superior to that of RKkM.

Additionally, Fig. 7 illustrates the CPU time of each algorithm to compute all K clustering solutions across the three graphs. Each subfigure shows the time required for each objective on the compared graphs to solve all clustering problems for $k = 1, \ldots, 50$. Specifically, we observe that regarding the ratio association objective, the GKkM++ variations require the least time in all graphs. For the normalized cut objective, GKkM++ outperforms all other algorithms on the EMAIL-EU-l-TD1 and EMAIL-EU graphs, while achieving comparable runtime performance to RKkM on the GRQC graph.

Table 6 Descriptions of utilized datasets

Dataset	Description	N	d	Source
AVILA	Images of an XII century copy of the Bible	20,867	10	[54]
BREAST CANCER	Characteristics of breast cancer tumors	569	30	[54]
DERMATOLOGY	Type of Erythemato-squamous disease	366	34	[54]
ECOLI	Expression levels of proteins	336	7	[54]
IRIS	Characteristics of Iris flower species	150	4	[54]
OLIVETTI FACES	Face image dataset	400	4096	[55]
PENDIGITS	Handwritten digits	10,992	16	[54]
WAVEFORM-V1	Waveforms with multiple attributes	5000	21	[54]
WINE	Chemical analysis of wines	178	13	[54]

4.3 Real datasets

We also conducted a series of experiments on several publicly available real-world datasets. We intentionally chosen datasets encompassing various characteristics, including the number of samples (N), data dimensionality (d), complexity, and domain of origin. Table 6 provides a detailed description of each dataset.

Min–max normalization in the [0, 1] range has been applied to each dataset to avoid numerical instabilities in the

computations [56]. Additionally, for a more thorough investigation, we considered for each real dataset three different kernel functions, as presented in Table 3, which resulted in a total of 27 clustering problems.

4.3.1 Experimental setup

In our experimental study on real-world datasets, we evaluate the compared methods both for the maximum number of clusters K and for all intermediate clustering solutions $k = 1, \ldots, K$. We set the maximum number of clusters $K = 50$ in all cases. To ensure a fair comparison, in each k subproblem, we executed the algorithms with $L = 100$, where L specifies the number of restarts or the number of candidates examined. In particular, for KkM++ and RKkM, we used L to control the number of restarts. In the case of GKkM++, L defines the number of candidates selected. On the other hand, GKkM is executed once, which is equivalent to running GKkM++ with the number of candidates $L = N$, where N is the total number of data instances.

Additionally, we evaluated the clustering performance using three kernel functions: Cosine, Polynomial, and RBF (Radial Basis Function), as shown in Table 3. For simplicity, in the Polynomial and RBF kernels, we set the γ hyperparameter for each dataset using the following formula:

$$\gamma = \frac{1}{\sigma^2 d}, \tag{9}$$

where σ^2 is the variance and d the dimensionality. This equation accounts for both the variance of the data and the number of input features, ensuring an adaptive initialization of γ. This approach aligns with the standard γ initialization strategy of well-known software libraries such as scikit-learn [57]. However, more advanced techniques can also be applied [58]. Finally, for the Polynomial kernel, we set the degree *deg* and the coefficient c_0 to typical values, such as $deg = 3$ and $c_0 = 1$.

4.3.2 Evaluation

In this study, we propose kernel-based clustering methods that effectively minimize the feature space clustering error. Therefore, we use this error $E(\mathcal{C}_k)$ (Eq. 2) for method comparison that provides a direct evaluation of the optimization capability of each method. More specifically, we calculate the relative Percentage Error (Eq. 8), where $E(\mathcal{C}_k^\star)$ denotes the error corresponding to the baseline method which is GKkM (Fig. 8). However, due to high computational complexity, in some cases, GKkM did not provide solutions in a reasonable time. In such cases, we utilized the GKkM++ with sequential sampling strategy as the baseline method. Furthermore, for each method and k, we report the CPU execution time (Fig. 10) and the average number of kernel k-means iterations (Fig. 11). Such an evaluation approach illustrates both the error minimization capability of each method as well as its computational speed and efficiency.

4.3.3 Experimental results

GKkM++ exhibits performance comparable to that of GKkM with both batch and sequential sampling strategies across all datasets (Fig. 8). Notably, it clearly outperforms the GKkM method in several cases, which is more evident for large k values. It should be noted that GKkM did not terminate within a reasonable time frame in AVILA, PENDIGITS and WAVEFORM-V1 datasets due to its high computational complexity, where in these cases, the GKkM++ exhibited the best results. As the number of clusters k increases, the clustering problem becomes more challenging, and the performance difference between GKkM variants and the rest of the compared methods becomes more profound. Solving the clustering problem with a larger number of clusters is crucial, as datasets inherently contain many clusters, such as OLIVETTI FACES. Additionally, in most real-world scenarios, the number of clusters is unknown a priori. Therefore, the clustering problem should be addressed across a range of values for k, allowing us to determine the most suitable solution. In such cases, the minimization algorithm must produce good results for even larger k values.

Figure 9 presents the relative PE of each method compared to GKkM accumulated across all $k = 2, \ldots, K$ subproblems and the 18 (out of 27) datasets where GKkM successfully converged within a reasonable time frame. Note that positive values indicate that the GKkM had superior performance, while negative values mean that the compared algorithm performed better.

In Fig. 9a, it is evident that GKkM++ demonstrates highly effective optimization capabilities. In most cases, GK*k*M++, using both batch and sequential strategies, converged to solutions with a PE of 0.5% or less. Even in the worst case, the maximum PE did not exceed 1.4% in our experiments. The plot emphasizes how closely GK*k*M++ approximates the performance of GKkM across various clustering subproblems. Interestingly, there are several instances where GKkM++ outperforms the solution of GKkM, and this improvement is more pronounced for the sequential sampling strategy (orange histogram) where there are cases in which the solution of GKkM++ had -1.5 PE compared to GKkM.

Moreover, it is clear that the GKkM variants consistently outperform both KkM++ and RKkM as shown in Fig. 9b. In the case of RKkM, the Percentage Error (PE) reached as high as 100% (not included in the figure), whereas K*k*M++

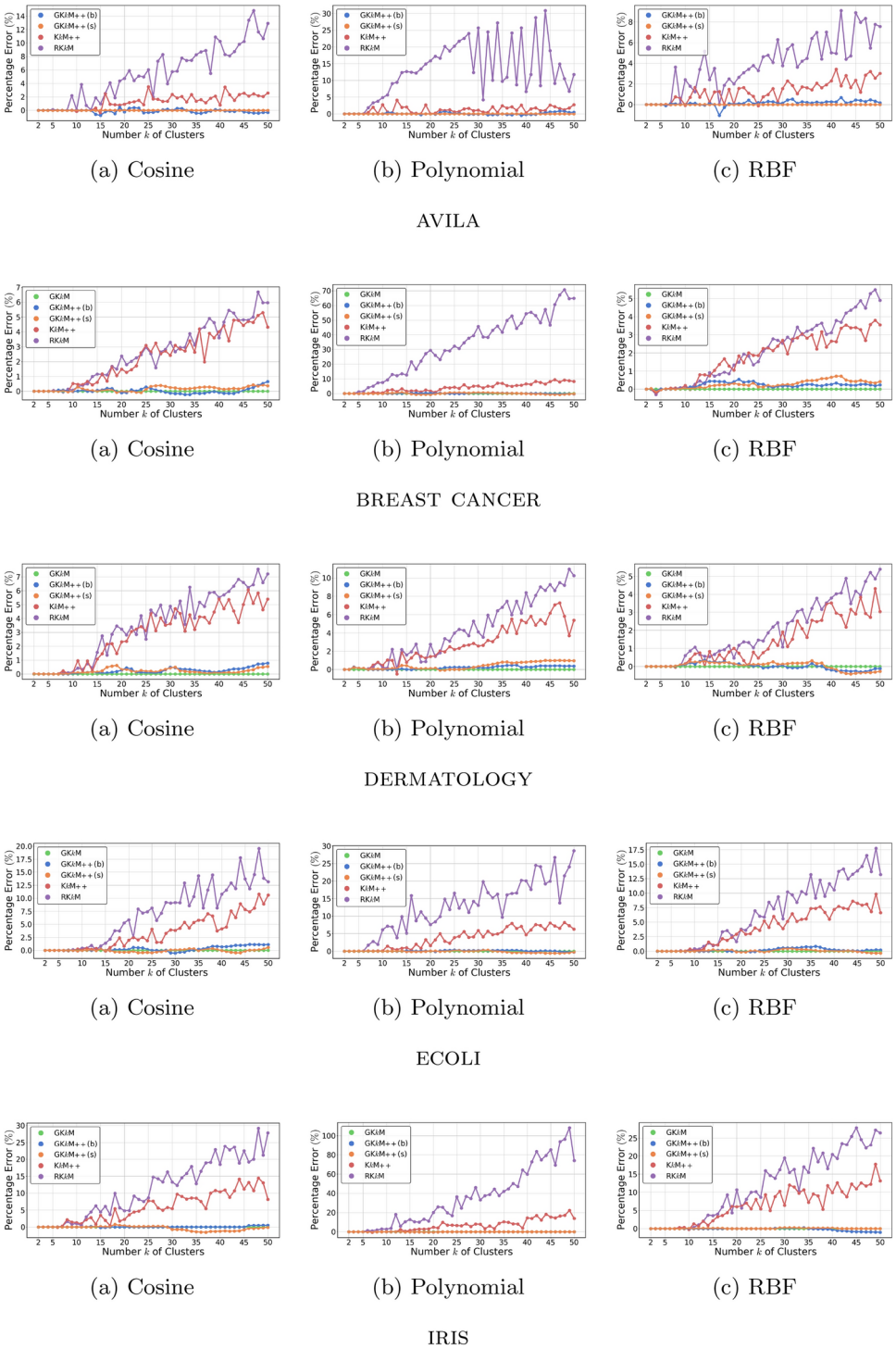
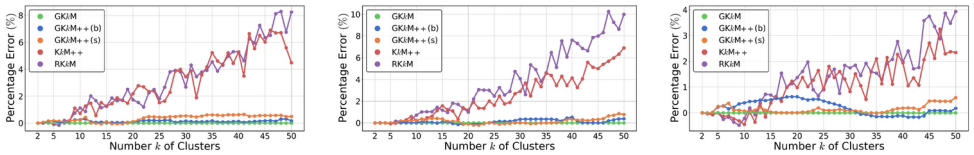


Fig. 8 Comparison of the relative percentage error for each algorithm (relative to the GKkM method) across various datasets and kernel functions. Lower values indicate better clustering performance, with global optimization variants achieving the lowest error in most cases

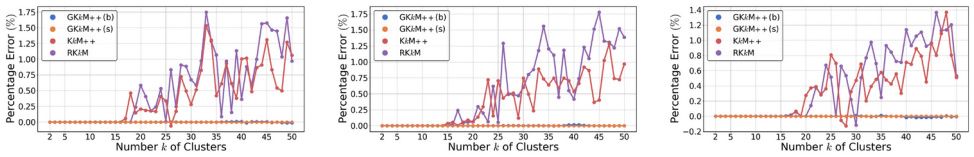
exhibited a maximum PE of 22%. It should be noted that the dashed red line denotes the Maximum PE of GKkM++ algorithm. As anticipated, KkM++ significantly outperforms RKkM. However, it cannot match the optimization capabilities of the GKkM variants.

Figure 10 illustrates the time the CPU needs for each algorithm to compute each one of K clustering solutions for the datasets. Specifically, each subfigure presents the time required for each dataset and kernel to provide solutions for all $k = 1, \ldots, 50$. Overall, the GKkM++ variants demonstrate the highest efficiency, requiring the least execution time, often just a fraction of the time needed by the other methods. Some notable cases are the AVILA, WAVEFORM-V1 and PENDIGITS datasets in which the difference of GKkM++ is several days of execution compared to the second fastest algorithm. In these datasets, the GKkM failed to converge



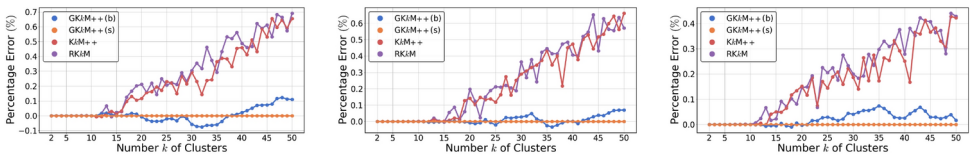
(a) Cosine (b) Polynomial (c) RBF

OLIVETTI FACES



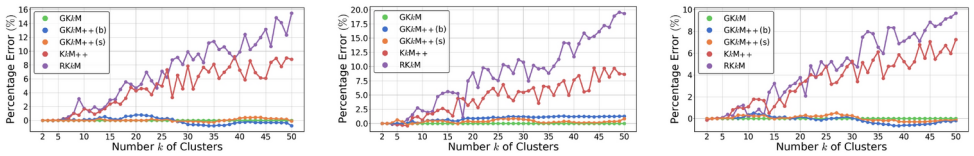
(a) Cosine (b) Polynomial (c) RBF

PENDIGITS



(a) Cosine (b) Polynomial (c) RBF

WAVEFORM-V1

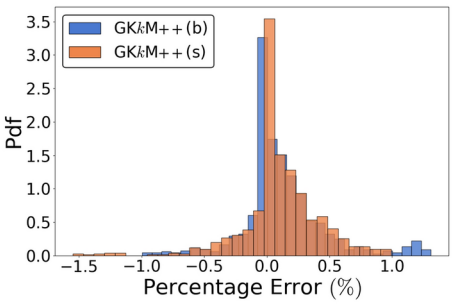


(a) Cosine (b) Polynomial (c) RBF

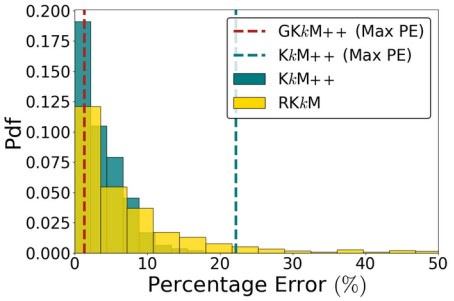
WINE

Fig. 8 (continued)

Fig. 9 Distribution of relative percentage error for different clustering methods compared to GKkM. **a** Shows the performance of GKkM++ (using both batch and sequential sampling), which closely aligns with the GKkM method. **b** Compares KkM++ and RKkM, highlighting their higher error values



(a) Relative PE distribution of GKkM++, compared to GKkM, using both batch and sequential sampling strategies.



(b) Relative PE distribution of KkM++ and RKkM, compared to GKkM. The red and the blue dash lines indicate the maximum PE of the GKkM++ and KkM++ algorithms, respectively.

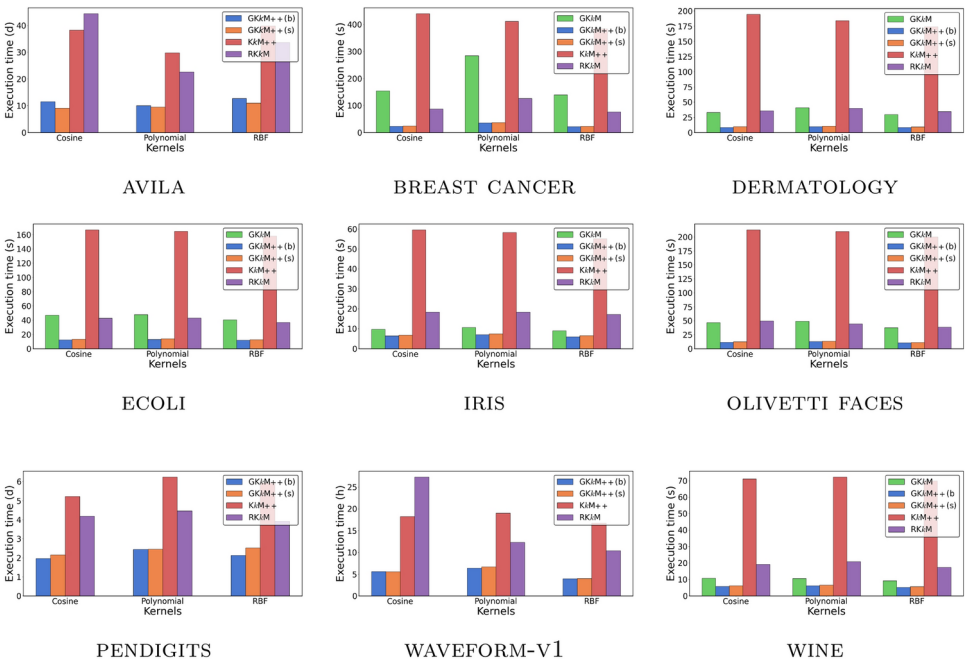


Fig. 10 Comparison of CPU execution time required to compute all clustering solutions for different datasets and kernels. GKkM++ demonstrates significantly reduced computational cost compared to other methods

after weeks of execution due to the large number of data instances N.

This speedup in execution is also highlighted in Fig. 11, which shows the average number of kernel k-means iterations required for convergence by each method. As observed, the GKkM variants tend to require fewer iterations as k increases. This is mainly because, in each k subproblem, the $k-1$ clusters are already well partitioned. However, GKkM requires more iterations at lower values of k due to its exhaustive search nature, although its behavior is close to that of the GKkM++ variants. In contrast, KkM++ and RKkM generally require more iterations as k grows, or their iteration plateau is significantly higher than that of the global variants. This trend is particularly evident in the AVILA, BREAST CANCER, OLIVETTI FACES, DERMATOLOGY, PENDIGITS and WAVEFORM-V1 datasets. Generally, it can be noticed that GKkM++ requires considerably fewer kernel k-means iterations in all cases.

4.3.4 Sensitivity analysis

In this section, we investigate the effect of the number of candidates L on the performance of the proposed method. Since L determines the number of candidates evaluated at each k-cluster subproblem, it directly influences clustering performance and computational efficiency. Specifically, a higher L value allows for a more extensive exploration of alternative solutions, increasing the possibility of selecting high-quality cluster initialization. Naturally, this comes at the cost of additional computational overhead.

To analyze this trade-off, we conducted using various datasets and examining four values of L (10, 25, 50, 100). For each dataset and L value, 30 runs were conducted and, for each run, the clustering error attained for three different values of K (10, 25, 50) was used for our analysis. Note that both the sequential and batch sampling strategy were considered.

Figure 12 displays the influence of L on clustering performance, as measured by the clustering error. As expected, the increase of L leads to decrease in clustering error, since the exploration of more candidate centers improves the possibility of finding an optimal cluster placement. Additionally, as L grows, the variance of the clustering error decreases, indicating greater stability and robustness of the obtained solutions. Notably, there is no significant difference in clustering performance between the sequential and batch sampling strategies, suggesting that both selection methods are equally effective in the selected datasets.

Figure 13 examines the impact of L on computational efficiency, measured by execution time. As expected, increasing L results in a higher computational cost due to the greater number of kernel k-means runs. It can also be observed that there is no significant difference in the execution time of the two sampling strategies.

5 Conclusions and future works

Kernel k-means extends the k-means clustering algorithm to detect nonlinearly separable clusters. In order to address the inherent cluster initialization problem of this procedure, we

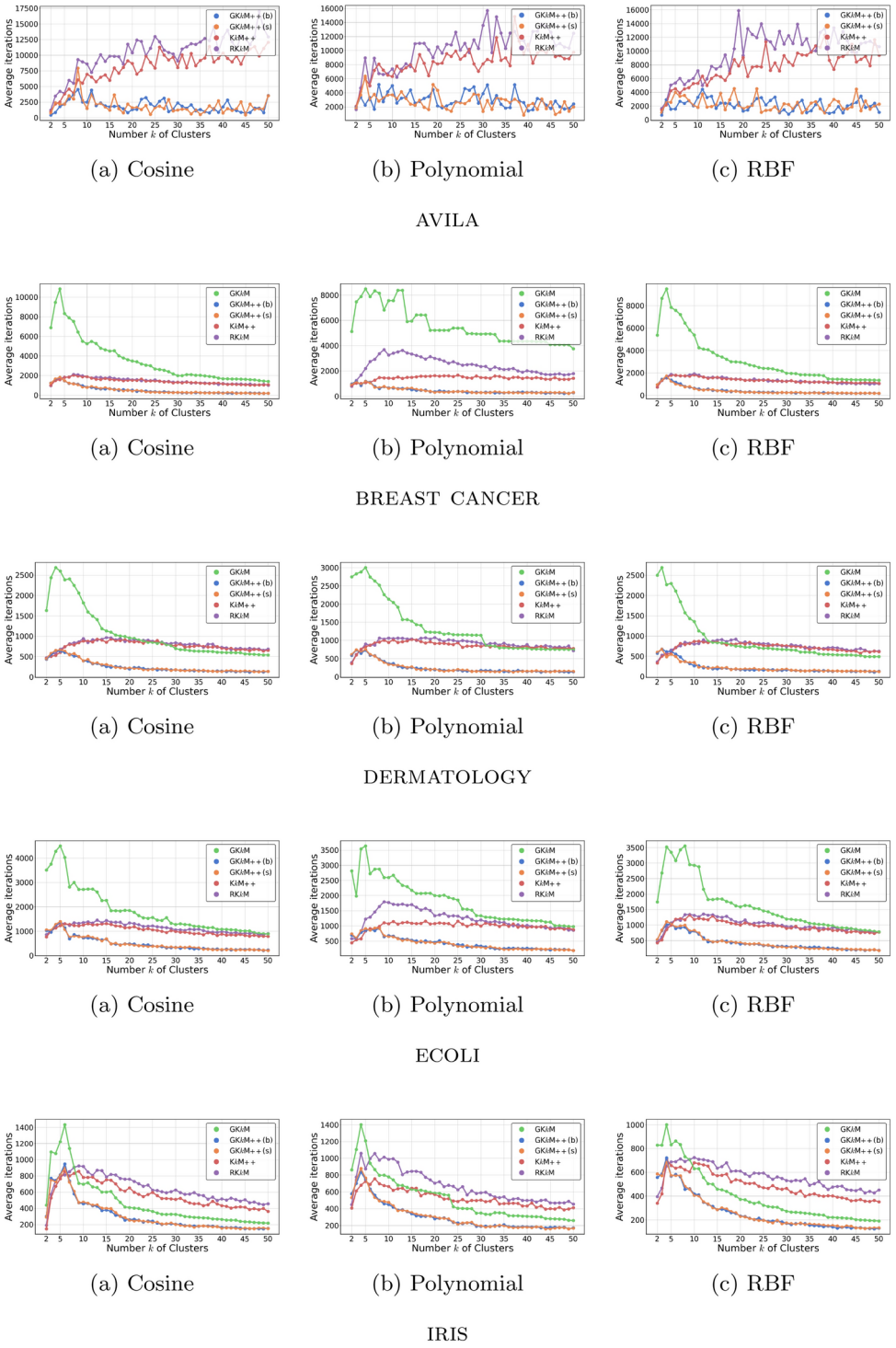


Fig. 11 Comparison of the average number of iterations required for kernel k-means to converge across different datasets and kernel functions. GKkM++ requires fewer iterations as k increases

first formulated the *kernel k-means* ++ (KkM++) method, which conveys the efficient center initialization strategy of k-means++ from Euclidean to kernel space.

Additionally, inspired by global k-means++, a method that integrates the sequential clustering solutions of the global k-means algorithm with the k-means++ initialization procedure in Euclidean space, we have proposed *global kernel k-means* ++ (GKkM++) that operates in feature space. GKkM++ constitutes a novel clustering algorithm that balances high-quality clustering in the feature space with reduced computational cost.

Specifically, GKkM++ is an incremental clustering algorithm that extends the well-established global kernel k-means algorithm by incorporating the stochastic

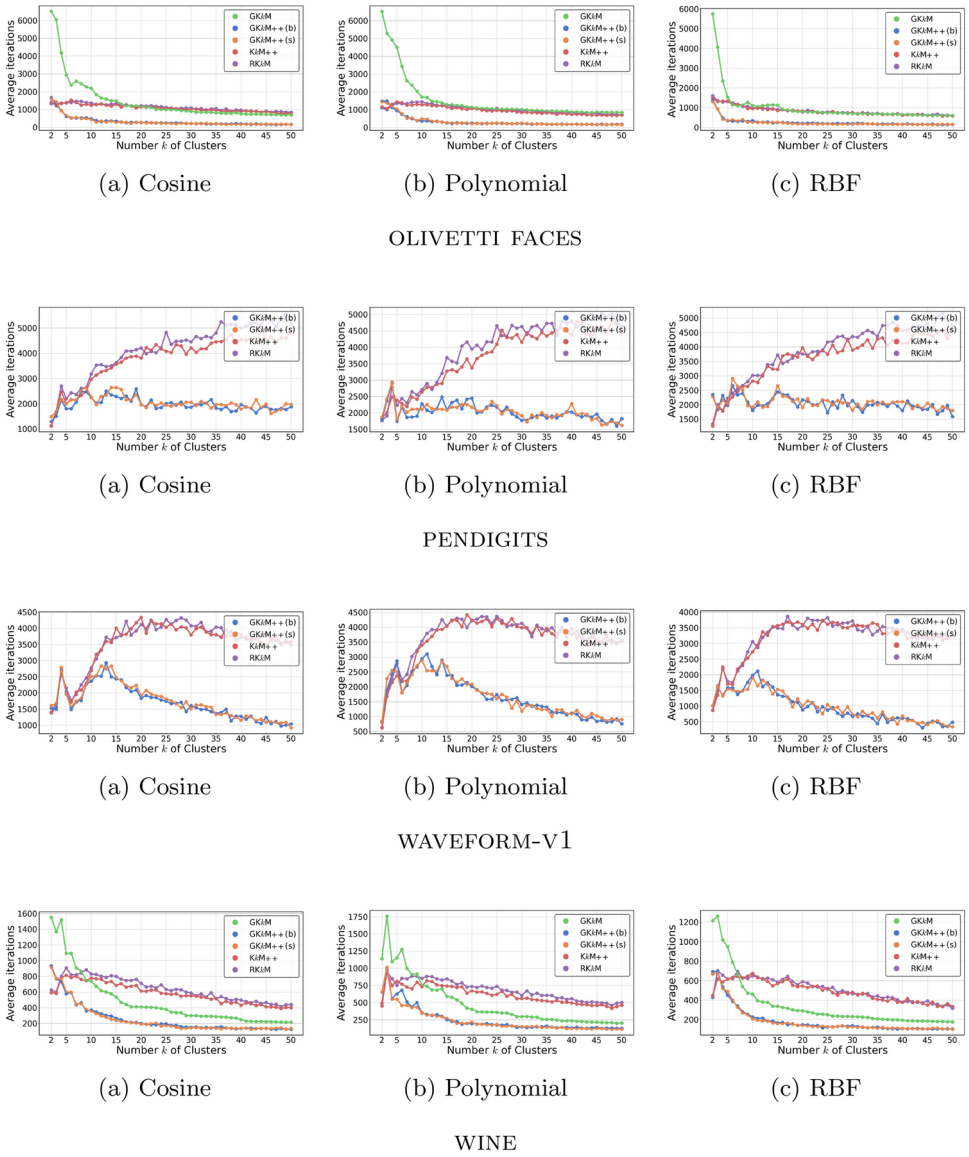


Fig. 11 (continued)

initialization strategy of kernel k-means++ to select L initial cluster candidates. To produce the solution with K clusters, it sequentially solves all intermediate subproblems for $k = 1, \ldots, K$, by sampling L initial cluster candidates at each k subproblem, where $L \ll N$. We presented two strategies for the sampling selection procedure: batch and sequential sampling. Specifically, the batch selection strategy samples L candidates at once without replacement. At the same time, sequential sampling selects L candidates one by one, also without replacement, updating the probability distribution accordingly at each sampling step. GKkM++ significantly reduces computational complexity while preserving superior minimization capabilities akin to those of traditional global kernel k-means (GKkM), making it practical for addressing clustering problems in larger datasets, where global kernel k-means may not terminate within a reasonable time frame. Nonetheless, it is important to recognize that reducing the computational complexity of the GKkM algorithm by sampling initial cluster candidates comes at the cost of losing its deterministic nature.

We evaluated the proposed algorithm on synthetic and on several publicly available benchmark datasets and compared it to various methods, including global kernel k-means, kernel k-means++ and kernel k-means with random uniform initialization (RKkM). In all cases, GKkM++ has demonstrated its superior clustering performance and reduced computational cost. In addition, we evaluate its performance on the graph partitioning problem. Overall, the experimental results demonstrate that GKkM++ consistently achieves significantly better clustering optimization

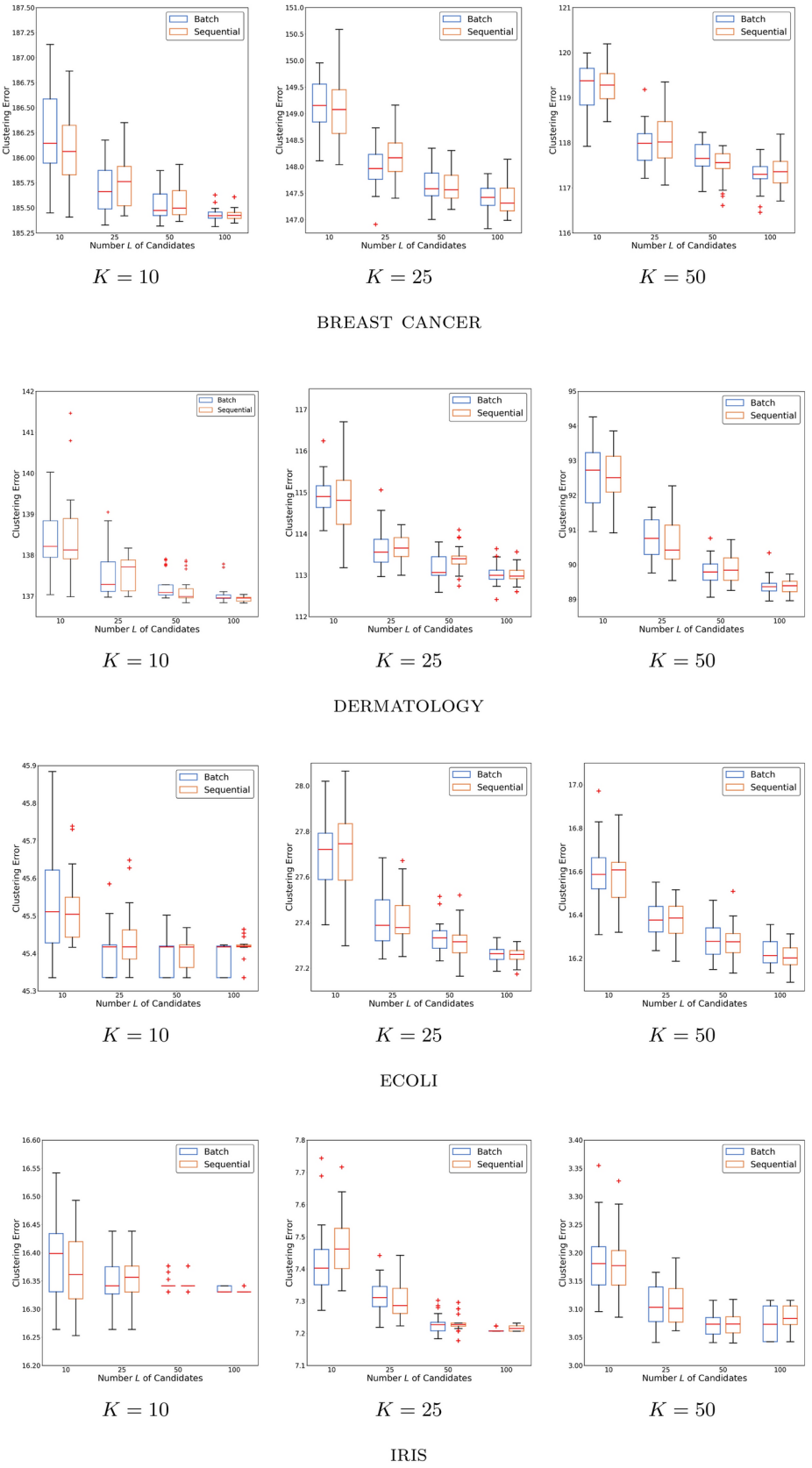


Fig. 12 Effect of the number of candidates L on clustering performance of the proposed method for several datasets. For each dataset the clustering error statistics (over 30 runs) is presented for different values of L and number of clusters $K = 10, 25, 50$ using both the sequential and the batch sampling strategy

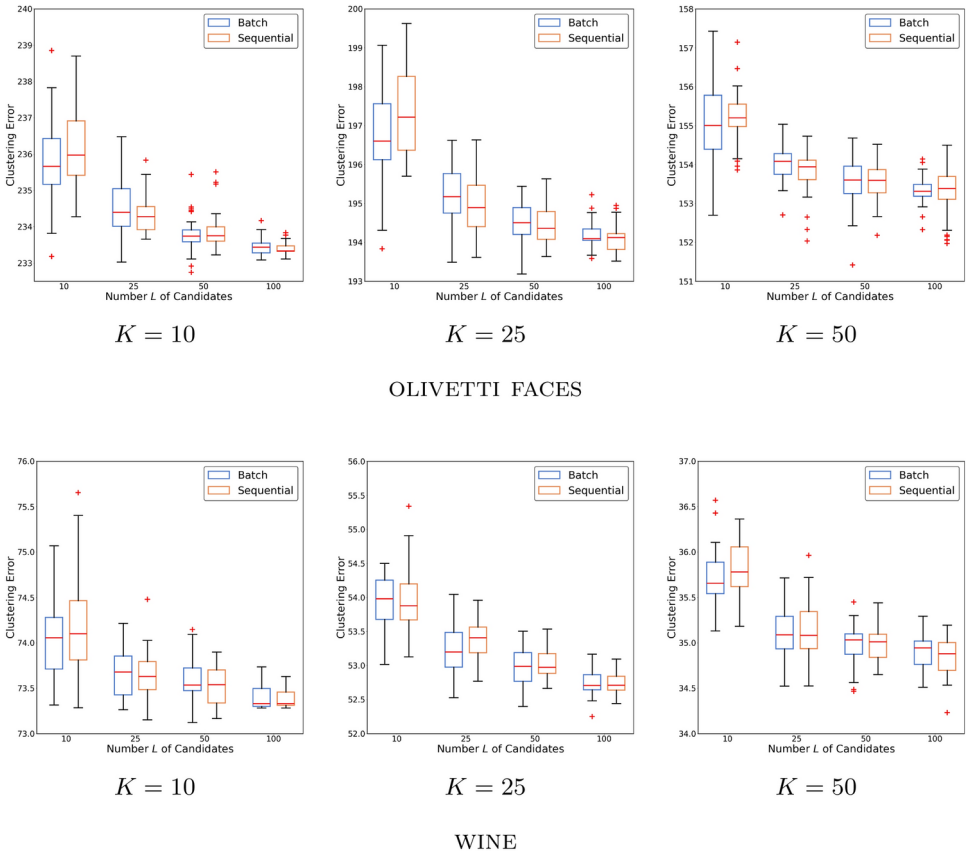


Fig. 12 (continued)

capabilities than KkM++ and RKkM. Furthermore, its performance is comparable to that of the GKkM method, with a maximum percentage error of less than 1.4% in the real datasets while achieving a maximum percentage error of less than 0.25% on the graph partitioning task. Surprisingly, in many cases, it even exceeds the performance of the GKkM. Overall, the GKkM++ variants demonstrate the highest efficiency, requiring the least execution time, often just a fraction of the time needed by the other methods.

In several applications, the number of clusters is unknown and is estimated by applying clustering for several values of k and selecting the final solution using an appropriate quality score. Since the proposed method generates clustering solutions for all values of $k = 1, \ldots, K$, it is ideal for this task. In such cases, our method requires smaller execution time even with respect to RKkM and KkM++ that are not incremental. In conclusion, the proposed algorithm strikes an optimal balance between the strengths of both GKkM and KkM++, delivering high-quality clustering at a significantly lower computational cost.

Future work could include investigating a technique for adapting the number of candidates L as k increases. We observed that fewer candidates are required to be examined for small values of k since the problem is simpler. As k increases, the problem becomes more difficult, necessitating more extensive exploration using more candidates. We believe that dynamically tuning L in each k-cluster subproblem could further improve both the attained clustering error and the execution time of the method.

In addition, we plan to investigate the estimation of the number of clusters in this algorithmic framework using criteria such as silhouette coefficient [45], modularity [46], and inclusion [47, 48]. Furthermore, future research could investigate the proposed approach's convergence properties and error bounds, as these represent significant challenges in a global optimization setting.

The proposed method is applicable to a wide range of clustering and graph partitioning problems. For example, it can be used for image segmentation [59], where grouping pixels or regions with similar features is crucial, and bioinformatics [60] where analyzing gene expression patterns requires accurate non-linear clustering. In a graph partitioning framework, the method can be applied to social network analysis to detect communities in large-scale networks and to recommendation systems for clustering users based on shared preferences.

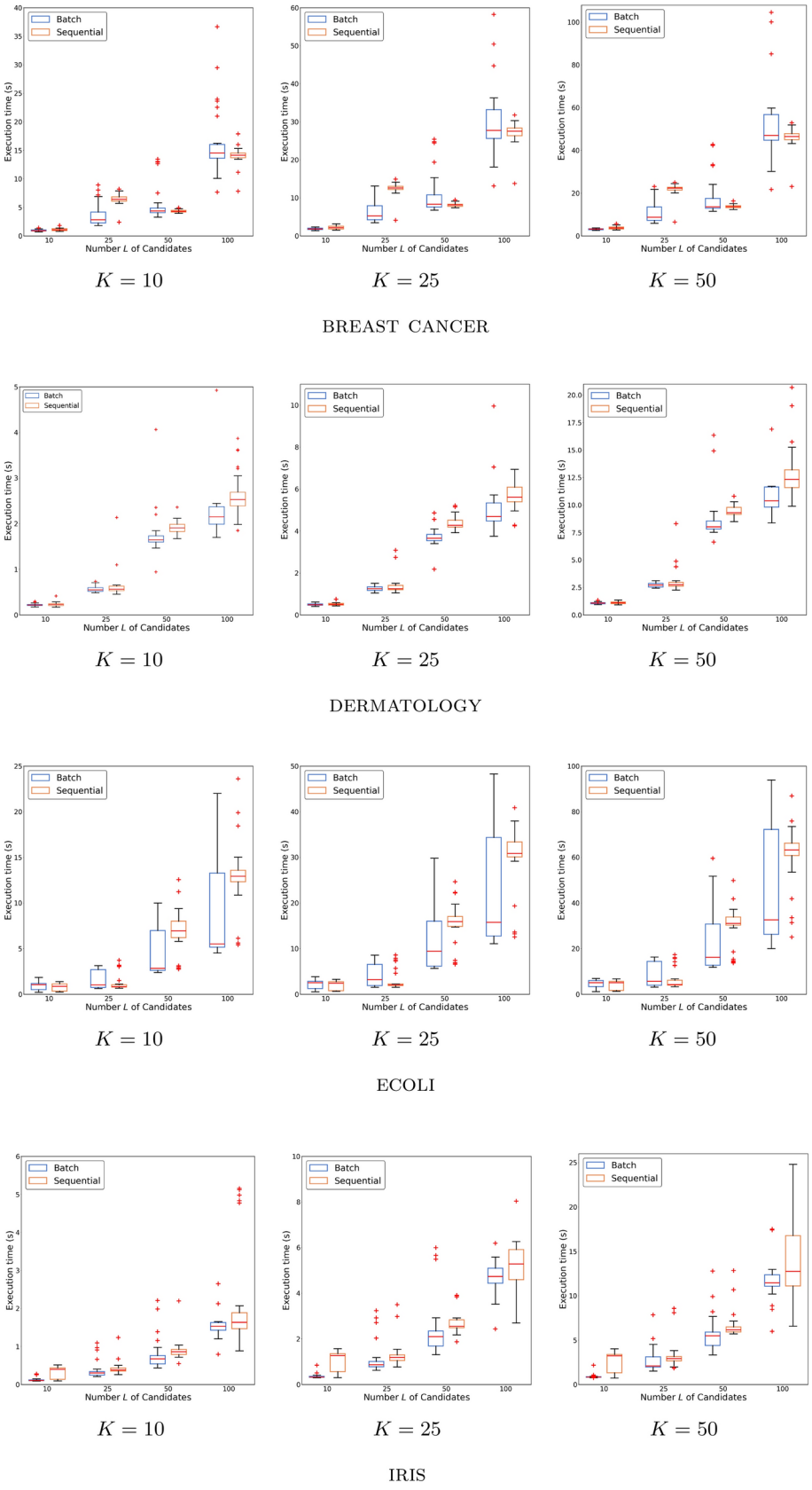


Fig. 13 Effect of the number of candidate initializations L on computational efficiency for several datasets. For each dataset the execution time statistics (over 30 runs) is presented for different values of L and number of clusters $K = 10, 25, 50$ using both the sequential and the batch sampling strategy

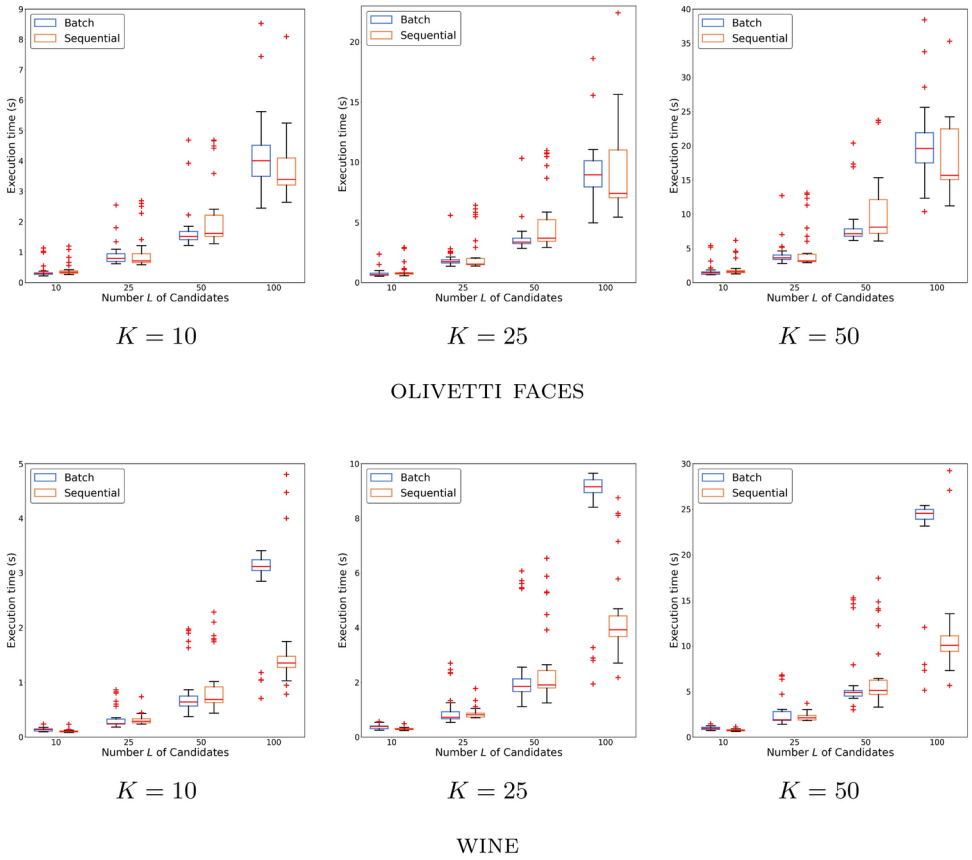


Fig. 13 (continued)

Finally, we aim to extend this method to other kernel *k*-means variations such as fuzzy kernel *k*-means [61, 62] and use the method in real-world non-linear clustering applications.

Acknowledgements This research project is implemented in the framework of H.F.R.I. call "Basic research Financing (Horizontal support of all Sciences)" under the National Recovery and Resilience Plan "Greece 2.0" funded by the European Union - NextGenerationEU (H.F.R.I. ProjectNumber: 15940).

Author Contributions Conceptualization: GV, IP, AL; Formal analysis: GV; Funding acquisition: AL; Investigation: GV, IP; Methodology: GV, IP, AL; Software: GV, IP; Supervision: AL; Validation: GV, IP, AL; Visualization: GV, IP; Writing—original draft: GV; Writing—review and editing: GV, IP, AL.

Funding Open access funding provided by HEAL-Link Greece.

Data Availability The datasets analysed during the current study are available in the https://archive.ics.uci.edu/ml/index.php UCI repository, and in the https://snap.stanford.edu/data/ SNAP database.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethics approval and consent to participate Not applicable.

Consent for publication Not applicable.

Code availability The code will be released on https://github.com/gvardakas/global-kernel-k-means-pp GitHub.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv (CSUR) 31(3):264–323
2. Saxena A, Prasad M, Gupta A, Bharill N, Patel OP, Tiwari A, Er MJ, Ding W, Lin C-T (2017) A review of clustering techniques and developments. Neurocomputing 267:664–681
3. Ezugwu AE, Ikotun AM, Oyelade OO, Abualigah L, Agushaka JO, Eke CI, Akinyelu AA (2022) A comprehensive survey of clustering algorithms: state-of-the-art machine learning applications,

taxonomy, challenges, and future research prospects. Eng Appl Artif Intell 110:104743
4. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1. Oakland, CA, USA, pp 281–297
5. Lloyd S (1982) Least squares quantization in pcm. IEEE Trans Inf Theory 28(2):129–137
6. Celebi ME, Kingravi HA, Vela PA (2013) A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Syst Appl 40(1):200–210
7. Likas A, Vlassis N, Verbeek JJ (2003) The global k-means clustering algorithm. Pattern Recogn 36(2):451–461
8. Arthur D, Vassilvitskii S (2006) k-means++: the advantages of careful seeding. Technical report, Stanford
9. Vardakas G, Likas A (2024) Global k-means++: an effective relaxation of the global k-means clustering algorithm. Appl Intell 54(19):8876–8888
10. Tzortzis GF, Likas AC (2009) The global kernel k-means algorithm for clustering in feature space. IEEE Trans Neural Netw 20(7):1181–1194
11. Dhillon IS, Guan Y, Kulis B (2004) Kernel k-means: spectral clustering and normalized cuts. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, pp 551–556
12. Lai JZ, Huang T-J (2010) Fast global k-means clustering using cluster membership and inequality. Pattern Recogn 43(5):1954–1963
13. Bagirov AM, Ugon J, Webb D (2011) Fast modified global k-means algorithm for incremental cluster construction. Pattern Recogn 44(4):866–876
14. Xie J, Jiang S, Xie W, Gao X (2011) An efficient global k-means clustering algorithm. J Comput 6(2):271–279
15. Agrawal A, Gupta H (2013) Global k-means (gkm) clustering algorithm: a survey. Int J Comput Appl 79(2):20–24
16. Bahmani B, Moseley B, Vattani A, Kumar R, Vassilvitskii S (2012) Scalable k-means++. Proc VLDB Endowment 5(7):622–633
17. Bachem O, Lucic M, Hassani H, Krause A (2016) Fast and provably good seedings for k-means. Adv Neural Inf Process Syst 29:55–63
18. Lattanzi S, Sohler C (2019) A better k-means++ algorithm via local search. In: International conference on machine learning. PMLR, pp 3662–3671
19. Choo D, Grunau C, Portmann J, Rozhon V (2020) k-means++: few more steps yield constant approximation. In: International conference on machine learning. PMLR, pp 1909–1917
20. Grunau C, Özüdoğru AA, Rozhoň V, Tětek J (2023) A nearly tight analysis of greedy k-means++. In: Proceedings of the 2023 Annual ACM-SIAM symposium on discrete algorithms (SODA). SIAM, pp 1012–1070
21. Beretta L, Cohen-Addad V, Lattanzi S, Parotsidis N (2023) Multi-swap k-means++. Adv Neural Inf Process Syst 36:26069–26091
22. Arbelaitz O, Gurrutxaga I, Muguerza J, Pérez JM, Perona I (2013) An extensive comparative study of cluster validity indices. Pattern Recogn 46(1):243–256
23. Fränti P, Sieranoja S (2019) How much can k-means be improved by using better initialization and repeats? Pattern Recogn 93:95–112
24. Ikotun AM, Ezugwu AE, Abualigah L, Abuhaija B, Heming J (2023) K-means clustering algorithms: a comprehensive review, variants analysis, and advances in the era of big data. Inf Sci 622:178–210
25. Schölkopf B, Smola A, Müller K-R (1998) Nonlinear component analysis as a kernel eigenvalue problem. Neural Comput 10(5):1299–1319
26. Filippone M, Camastra F, Masulli F, Rovetta S (2008) A survey of kernel and spectral methods for clustering. Pattern Recogn 41(1):176–190
27. Paul D, Chakraborty S, Das S, Xu J (2022) Implicit annealing in kernel spaces: a strongly consistent clustering approach. IEEE Trans Pattern Anal Mach Intell 45(5):5862–5871
28. Ng A, Jordan M, Weiss Y (2001) On spectral clustering: analysis and an algorithm. Adv Neural Inf Process Syst 14:1–8
29. Jia H, Ding S, Xu X, Nie R (2014) The latest research progress on spectral clustering. Neural Comput Appl 24:1477–1486
30. Wang D, Li T, Deng P, Liu J, Huang W, Zhang F (2022) A generalized deep learning algorithm based on nmf for multi-view clustering. IEEE Trans Big Data 9(1):328–340
31. Wang D, Li T, Huang W, Luo Z, Deng P, Zhang P, Ma M (2023) A multi-view clustering algorithm based on deep semi-nmf. Inf Fusion 99:101884
32. Wang D, Li T, Deng P, Zhang F, Huang W, Zhang P, Liu J (2023) A generalized deep learning clustering algorithm based on non-negative matrix factorization. ACM Trans Knowl Discov Data 17(7):1–20
33. Vardakas G, Likas A (2023) Neural clustering based on implicit maximum likelihood. Neural Comput Appl 35(29):21511–21524
34. Wang D, Zhang P, Deng P, Wu Q, Chen W, Jiang T, Huang W, Li T (2024) An autoencoder-like deep nmf representation learning algorithm for clustering. Knowl-Based Syst 305:112597
35. Wang D, Li T, Deng P, Luo Z, Zhang P, Liu K, Huang W (2024) Dnsrf: deep network-based semi-nmf representation framework. ACM Trans Intell Syst Technol 15(5):1–20
36. Vardakas G, Papakostas I, Likas A (2024) Deep clustering using the soft silhouette score: towards compact and well-separated clusters. arXiv preprint arXiv:2402.00608 (2024)
37. Tzortzis G, Likas A (2008) The global kernel k-means clustering algorithm. In: 2008 IEEE international joint conference on neural networks (ieee world congress on computational intelligence). IEEE, pp 1977–1984
38. Fritzke B (2020) Breathing k-means. arXiv preprint arXiv:2006.15666 (2020)
39. Bai L, Liang J, Sui C, Dang C (2013) Fast global k-means clustering based on local geometrical information. Inf Sci 245:168–180
40. Müller K-R, Mika S, Tsuda K, Schölkopf K (2018) An introduction to kernel-based learning algorithms. In: Handbook of neural network signal processing. CRC Press, Boca Raton, FL, pp 4-1
41. Dhillon IS, Guan Y, Kulis B (2004) A unified view of kernel k-means, spectral clustering and graph cuts. Citeseer, University Park, PA
42. Forgy EW (1965) Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics 21:768–769
43. Tsapanos N, Tefas A, Nikolaidis N, Pitas I (2015) A distributed framework for trimmed kernel k-means clustering. Pattern Recogn 48(8):2685–2698
44. Jiang SH-C, Krauthgamer R, Lou J, Zhang Y (2024) Coresets for kernel clustering. Mach Learn 113(8):5891–5906
45. Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65
46. Newman ME (2006) Modularity and community structure in networks. Proc Natl Acad Sci 103(23):8577–8582
47. Koufos N, Likas A (2018) The inclusion measure for community evaluation and detection in unweighted networks. In: 2018 IEEE/ACM international conference on advances in social networks analysis and mining (ASONAM). IEEE, pp 1053–1056
48. Kornelakis N, Likas A (2024) The inclusion criterion for data clustering quality. In: Proceedings of the 13th hellenic conference on artificial intelligence, pp 1–4

49. Dhillon IS, Guan Y, Kulis B (2007) Weighted graph cuts without eigenvectors a multilevel approach. IEEE Trans Pattern Anal Mach Intell 29(11):1944–1957
50. Zha H, He X, Ding C, Gu M, Simon H (2001) Spectral relaxation for k-means clustering. Adv Neural Inf Process Syst 14
51. Leskovec J, Krevl A (2014) SNAP datasets: Stanford large network dataset collection. http://snap.stanford.edu/data
52. Shi J, Malik J (1997) Normalized cuts and image segmentation. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition. IEEE, pp 731–737
53. Shi (2003) Multiclass spectral clustering. In: Proceedings ninth IEEE international conference on computer vision. IEEE, pp 313–319
54. Kelly M, Longjohn R, Nottingham K (2023) The UCI machine learning repository. https://archive.ics.uci.edu
55. Samaria FS, Harter AC (1994) Parameterisation of a stochastic model for human face identification. In: Proceedings of 1994 IEEE workshop on applications of computer vision. IEEE, pp 138–142
56. Milligan GW, Cooper M (1988) A study of standardization of variables in cluster analysis. J Classif 5:181–204
57. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. J Mach Learn Res 12:2825–2830
58. Kitayama S, Yamazaki K (2011) Simple estimate of the width in gaussian kernel with adaptive scaling technique. Appl Soft Comput 11(8):4726–4737
59. Khan Z, Yang J (2024) Nonparametric k-means clustering-based adaptive unsupervised colour image segmentation. Pattern Anal Appl 27(1):17
60. Jothi R, Mohanty SK, Ojha A (2019) Dk-means: a deterministic k-means clustering algorithm for gene expression analysis. Pattern Anal Appl 22:649–667
61. Das P, Das A (2019) A fast and automated segmentation method for detection of masses using folded kernel based fuzzy c-means clustering algorithm. Appl Soft Comput 85:105775
62. Graves D, Pedrycz W (2010) Kernel-based fuzzy clustering and fuzzy clustering: a comparative experimental study. Fuzzy Sets Syst 161(4):522–543

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.