Efficient Algorithm for K-Multiple-Means

YASUHIRO FUJIWARA, NTT Communication Science Laboratories, Japan
ATSUTOSHI KUMAGAI, NTT Computer and Data Science Laboratories, Japan
YASUTOSHI IDA, NTT Computer and Data Science Laboratories, Japan
MASAHIRO NAKANO, NTT Communication Science Laboratories, Japan
MAKOTO NAKATSUJI, NTT Human Informatics Laboratories, Japan
AKISATO KIMURA, NTT Communication Science Laboratories, Japan

K-Multiple-Means is an extension of K-means for the clustering of multiple means used in many applications, such as image segmentation, load balancing, and blind-source separation. Since K-means uses only one mean to represent each cluster, it fails to capture non-spherical cluster structures of data points. However, since K-Multiple-Means represents the cluster by computing multiple means and grouping them into specified c clusters, it can effectively capture the non-spherical clusters of the data points. To obtain the clusters, K-Multiple-Means updates a similarity matrix of a bipartite graph between the data points and the multiple means by iteratively computing the leading c singular vectors of the matrix. K-Multiple-Means, however, incurs a high computation cost for large-scale data due to the iterative SVD computations. Our proposal, *F-KMM*, increases the efficiency of K-Multiple-Means by computing the singular vectors from a smaller similarity matrix between the multiple means obtained from the similarity matrix of the bipartite graph. To compute the similarity matrix of the bipartite graph efficiently, we skip unnecessary distance computations and estimate lower bounding distances between the data points and the multiple means. Theoretically, the proposed approach guarantees the same clustering results as K-Multiple-Means since it can exactly compute the singular vectors from the similarity matrix between the multiple means. Experiments show that our approach is several orders of magnitude faster than previous clustering approaches that use multiple means.

CCS Concepts: • **Information systems → Clustering**; • **Computing methodologies → Machine learning algorithms**.

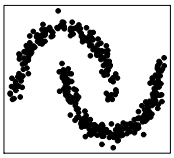
ACM Reference Format:
YASUHIRO FUJIWARA, ATSUTOSHI KUMAGAI, YASUTOSHI IDA, MASAHIRO NAKANO, MAKOTO NAKATSUJI, and AKISATO KIMURA. 2024. Efficient Algorithm for K-Multiple-Means. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 18 (February 2024), 26 pages. https://doi.org/10.1145/3639273

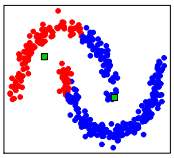
1 INTRODUCTION

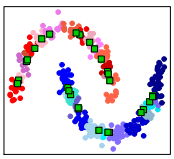
Massive datasets are now being stored day after day with the rapid development of database systems, and they must be managed in an effective manner [12–14]. Clustering is an essential data analysis process as it can extract clusters by grouping data with high similarity. K-means is one of

Authors' addresses: YASUHIRO FUJIWARA, NTT Communication Science Laboratories, Morinosato Wakamiya, Atsugi-shi, Japan, yasuhiro.fujiwara@ntt.com; ATSUTOSHI KUMAGAI, NTT Computer and Data Science Laboratories, Midori-cho, Musashino-shi, Japan, atsutoshi.kumagai@ntt.com; YASUTOSHI IDA, NTT Computer and Data Science Laboratories, Midori-cho, Musashino-shi, Japan, yasutoshi.ida@ieee.org; MASAHIRO NAKANO, NTT Communication Science Laboratories, Morinosato Wakamiya, Atsugi-shi, Japan, ma.nakano@ntt.com; MAKOTO NAKATSUJI, NTT Human Informatics Laboratories, Hikarinooka, Yokosuka-shi, Japan, makoto.nakatsuji@ntt.com; AKISATO KIMURA, NTT Communication Science Laboratories, Morinosato Wakamiya, Atsugi-shi, Japan, akisato@ieee.org.

ACM 2836-6573/2024/2-ART18
https://doi.org/10.1145/3639273


(1-1) Two-moon


(1-2) K-means


(1-3) Multi-prototypes

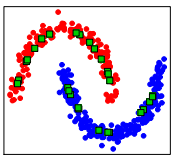
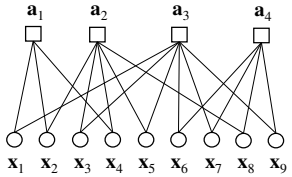
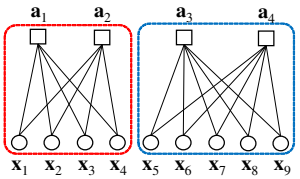

(1-4) K-Multiple-Means

Fig. 1. Comparison of K-means and K-Multiple-Means.



(2-1) Bipartite graph



(2-2) Connected component

Fig. 2. Bipartite graph partitioning problem

the most popular clustering approaches [26]. Theoretically, it is categorized as a prototype-based clustering approach that computes clusters by minimizing the sum of the squared errors of data points to the associated prototype (the mean of the cluster). Despite its simplicity, K-means has a major drawback: it does not work well for non-spherically separable datasets. For example, K-means fails to capture the clusters for the two-moon dataset of Fig. 1-1 with two non-spherical clusters, as shown in Fig. 1-2. This is because K-means assigns each data point to the nearest prototype; it assumes the dataset has hyper-spherical clusters. Note that, in Fig. 1, data points of the same cluster have the same color, and green square points are prototypes.

Nie et al. proposed K-Multiple-Means to overcome the drawback of K-means [30]. Unlike K-means, it uses the multi-prototype representation, which models clusters via multiple prototypes to compute specified c clusters. As shown in Fig. 1-3, it first separates the data points into $m(> c)$ sub-clusters and then groups the m sub-clusters into c clusters, as shown in Fig. 1-4. Since it uses more than one prototype per cluster, it can effectively represent the geometries of non-spherical shapes. As a result, K-Multiple-Means can more effectively capture cluster structures than K-means, as well as other clustering approaches such as Self-tuning spectral clustering [49], Mercer kernel-based clustering [16], RSFKM [48], MEAP [42], K-MEAP [45], and CLR [33]. Due to its effectiveness, K-Multiple-Means is used in many applications, such as image segmentation [20], load balancing [17], and blind source separation [25].

Theoretically, K-Multiple-Means formalizes the multiple-means clustering problem as a graph partitioning problem. In the problem, it considers a bipartite graph of data point $\mathbf{x}_i$ $(1 \leq i \leq n)$ and prototype $\mathbf{a}_j$ $(1 \leq j \leq m)$, as shown in Fig. 2-1, and it iteratively updates similarities between nodes in the bipartite graph so that the bipartite graph has c connected components, as shown in Fig. 2-2. Since the n data points with the m prototypes are partitioned into c connected components, the clusters correspond to the connected components in the bipartite graph. Besides, if the bipartite graph has c connected components, the rank of the normalized Laplacian matrix of the bipartite graph is $n + m - c$ [41]. Therefore, to obtain the clusters, K-Multiple-Means models the partitioning of n data points with m prototypes as a bipartite graph partitioning problem with constrained Laplacian rank. Note that the normalized Laplacian matrix of the bipartite graph is obtained from the similarities between the data points and prototypes [31].

To solve the problem, K-Multiple-Means uses an alternating optimization strategy for an $n \times m$ similarity matrix of the bipartite graph. Specifically, it iteratively updates the similarities using the

c singular vectors associated with the c largest singular values of the similarity matrix to satisfy the rank constraint. Then, it updates the prototypes of sub-clusters. However, it incurs excessive processing time for large-scale data since it iteratively computes SVD at $O(nm^2)$ time on the $n \times m$ similarity matrix to obtain the singular vectors. Although we can reduce the computational cost by exploiting randomized SVD [28], this approach yields different clustering results from the original approach of K-Multiple-Means since it approximately computes SVD. As a result, this approach sacrifices the clustering result to improve efficiency.

This paper proposes *F-KMM*, a novel and fast clustering approach that guarantees the same clustering results as K-Multiple-Means. To efficiently obtain the c singular vectors, the proposed approach exploits a low-rank property of the similarity matrix; the rank of the similarity matrix is at most m since the size of the similarity matrix is $n \times m$ where we have $m < n$ [46]. From this property, we compute an $m \times m$ similarity matrix between the prototypes from the similarity matrix of the bipartite graph, and we exactly compute the c singular vectors by computing the eigenvectors of the $m \times m$ similarity matrix between prototypes; we do not compute the SVD of the $n \times m$ similarity matrix of the bipartite graph. Since we have $m \ll n$ in practice, we can efficiently compute the eigenvectors of the $m \times m$ prototype matrix. Moreover, since the $m \times m$ prototype matrix is broken into small block matrices according to the graph partitions in the iterations, we can efficiently compute the singular vectors by processing the block matrices individually. To efficiently obtain the similarity matrix of the bipartite graph, we skip unnecessary distance computations and estimate lower bounding distances between the data points and the prototypes. Furthermore, we can optionally improve the clustering accuracy of our approach by initializing the prototypes effectively. In summary, the main contributions of this paper are as follows:

- We propose an efficient approach to K-Multiple-Means that computes the singular vectors of the similarity matrix of the bipartite graph by computing the eigenvectors of the smaller similarity matrix between the prototypes.
- Our approach does not sacrifice the clustering results to improve the efficiency of K-Multiple-Means. This is because it can exactly compute the singular vectors from the similarity matrix between the prototypes. In addition, the clustering accuracy of our approach can be optionally improved by initializing the prototypes effectively.
- Experiments confirm that our approach is up to 4,150 times faster than previous clustering approaches of the multi-prototype representation while it yields accurate clustering results as the original approach of K-Multiple-Means.

The remainder of this paper is organized as follows: Section 2 describes related work. Section 3 overviews the background. Section 4 introduces our approach. Section 5 shows experimental results. Section 6 provides our conclusions.

2 RELATED WORK

Despite the simpleness, K-means fails to capture non-spherical clusters since it exploits squared errors for prototype assignment. To overcome the problem, we can use nonlinear clustering methods such as kernel-based clustering and spectral clustering [8, 29, 32, 50]. Kernel-based clustering embeds the data points into a feature space where non-spherical clusters become linearly separable [35, 43]. Spectral clustering constructs a weighted graph of the data points and computes the eigenvectors of an affinity matrix to cluster the data points [9, 38]. However, the design of the appropriate kernel or the construction of the weighted graph is not easy to determine for each partition problem [9, 33].

Table 1. Definitions of main symbols.

Symbol	Definition
n	Number of data points
m	Number of prototypes
c	Number of clusters
d	Number of dimensions
b	Number of block matrices
$\lambda_{i,j}$	j-th largest eigenvalue of $\mathbf{M}'_i$
$s[i][j]$	Similarity of the i-th data point and the j-th prototype
$D[i,j]$	Distance between the i-th data point and j-th prototype
$\mathbf{x}_i$	i-th data point
$\mathbf{a}_j$	j-th prototype
$\mathbf{q}_{i,j}$	j-th eigenvector of $\mathbf{M}'_i$
$\mathbf{A}$	$m \times d$ prototype matrix
$\mathbf{S}$	$n \times m$ similarity matrix of the bipartite graph
$\mathbf{F}$	$(n+m) \times c$ matrix of singular vectors
$\mathbf{M}'_i$	i-th block matrix of $m_i \times m_i$

Multi-prototype clustering is a simple but effective alternative approach; it represents each cluster via multiple prototypes [2, 21, 23, 24, 30, 40, 44, 51]. K-Multiple-Means is categorized as a multi-prototype clustering. Multi-prototype clustering typically has two stages. In the split stage, the data points are divided into sub-clusters. In the merge stage, the sub-clusters are iteratively aggregated into a given number of clusters. Recently, Wang et al. proposed GKM-MPC as a multi-prototype clustering method [44]. In the split stage of GKM-MPC, sub-clusters are initialized using global K-means [22], and the sub-clusters are split by computing the shortest path in a nearest neighbor graph of prototypes to detect sub-clusters with low densities. In the merge stage, the densities around the boundaries of all the sub-cluster pairs are computed to form a density matrix between the prototypes, and c connected components are computed from the density matrix by iteratively truncating its elements. However, GKM-MPC has a high computation cost since it requires $O(n^2md)$ time to use global K-means to initialize the sub-clusters. Zhang recently proposed nKMM as a multi-prototype clustering method [51]. In the split stage, it initializes prototypes by randomly sampling data points. In the merge stage, it aggregates the sub-clusters if they have high similarities. It then uses the Chameleon algorithm to obtain the prespecified number of clusters [19]. Specifically, it computes a nearest neighbor graph of the data points and iteratively merges sub-clusters until it has c connected components in the graph. After the iteration, it updates the prototypes based on the clustering result. However, since it needs $O(n^2d)$ time to construct the nearest neighbor graph, nKMM incurs a high computation cost. As a result, since the computational costs of the recent approach to multi-prototype clustering are quadratic to the number of data points, they do not scale well enough to support large numbers of data points with high dimensionality.

3 PRELIMINARIES

We introduce here the background of this paper. Table 1 lists the main symbols. If d is the number of dimensions, the i-th data point is represented as $\mathbf{x}_i = [x_i[1], \ldots, x_i[d]]$. The j-th prototype is represented as $\mathbf{a}_j = [a_j[1], \ldots, a_j[d]]$ and $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_m]^\top$ is an $m \times d$ prototype matrix. The i-th data point $\mathbf{x}_i$ connects to the j-th prototype $\mathbf{a}_j$ with similarity $s[i][j]$. Let $\| \cdot \|_2$ be the L_2-norm, $s[i][j]$ has a large value as $\|\mathbf{x}_i - \mathbf{a}_j\|_2^2$ is small. If $\mathbf{S}$ is an $n \times m$ matrix whose (i,j)-th element is $s[i][j]$, $\mathbf{S}$ associates with the bipartite graph between the data points and the prototypes. To group the n data points with the m prototypes into c clusters, K-Multiple-Means computes the normalized Laplacian matrix $\mathbf{L}$ as follows:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}, \ \mathbf{W} = \begin{bmatrix} & \mathbf{S} \\ \mathbf{S}^\top & \end{bmatrix}. \tag{1}$$

In this equation, $\mathbf{D}$ is a $(n+m) \times (n+m)$ diagonal matrix whose i-th diagonal element $d[i]$ is set as follows:

$$d[i] = \sum_{j=1}^{n+m} w[i][j]. \tag{2}$$

If the rank of $\mathbf{L}$ is $n+m-c$, the bipartite graph has c connected components; the n data points with the m prototypes are grouped into the c clusters [41]. Therefore, K-Multiple-Means computes the clusters by solving the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{S},\mathbf{A}} \sum_{i=1}^{n} \sum_{j=1}^{m} s[i][j] \|\mathbf{x}_i - \mathbf{a}_j\|_2^2 + \alpha \|\mathbf{S}\|_F^2 + \beta \sum_{i=1}^{c} \rho_i \\ &\text{s.t. } \mathbf{S} \geq 0, \mathbf{S}\mathbf{1}_m = \mathbf{1}_m, \mathbf{A} \in R^{m \times d}, \end{aligned} \tag{3}$$

where $\| \cdot \|_F$ is the Frobenius-norm, $\alpha(\geq 0)$ and $\beta(\geq 0)$ are the regularization parameters of the optimization problem, ρ_i is the i-th smallest eigenvalue of $\mathbf{L}$, and $\mathbf{1}_m$ is a column vector of length m such that $\mathbf{1}_m = [1, \ldots, 1]^\top$. In the problem of (3), the first and second terms correspond to the assignment problem of the n data points to the m prototypes based on the weighted squared error. Regularization parameter α controls the sparsity of the connection of the data points to the multi-prototypes. The third term corresponds to the rank constraint of $\mathbf{L}$; it groups the m prototypes into c sets. When β is large enough, $\sum_{i=1}^{c} \rho_i$ would be small in the optimal solution, thus satisfying the rank constraint. According to Ky Fan's Theorem [11], if $\mathbf{F}$ is an $(n+m) \times c$ matrix of singular vectors, the optimization problem can be written as follows:

$$\begin{aligned} &\min_{\mathbf{F},\mathbf{S},\mathbf{A}} \sum_{i=1}^{n} \sum_{j=1}^{m} s[i][j] \|\mathbf{x}_i - \mathbf{a}_j\|_2^2 + \alpha \|\mathbf{S}\|_F^2 + \beta \operatorname{tr}(\mathbf{F}^\top \mathbf{L} \mathbf{F}) \\ &\text{s.t. } \mathbf{F} \in R^{(n+m) \times c}, \mathbf{F}^\top \mathbf{F} = \mathbf{I}, \mathbf{S} \geq 0, \mathbf{S}\mathbf{1}_m = \mathbf{1}_m, \mathbf{A} \in R^{m \times d}, \end{aligned} \tag{4}$$

where $\operatorname{tr}(\cdot)$ is the trace.

K-Multiple-Means solves the problem of (4) by using an alternating optimization method that updates $\mathbf{F}$, $\mathbf{S}$, and $\mathbf{A}$ iteratively. Specifically, in the first step, it updates $\mathbf{F}$ by fixing $\mathbf{S}$ and $\mathbf{A}$. In the second step, it updates $\mathbf{S}$ by fixing $\mathbf{F}$ and $\mathbf{A}$. It iterates these two steps until the rank constraint of $\mathbf{L}$ is satisfied; the bipartite graph has c connected components. The third step updates $\mathbf{A}$ by fixing $\mathbf{F}$ and $\mathbf{S}$ to relocate each prototype. It repeats step 1, 2, and 3 until the assignments of the prototypes no longer change. After the convergence, it obtains the c clusters from the c connected components.

When $\mathbf{S}$ and $\mathbf{A}$ are fixed, since $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}}$, the optimization problem of (4) becomes as follows:

$$\max_{\mathbf{F} \in R^{(n+m) \times c}, \mathbf{F}^\top \mathbf{F} = \mathbf{I}} \operatorname{tr}(\mathbf{F}^\top \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}} \mathbf{F}). \tag{5}$$

As shown in [31], this problem can be solved by computing the singular vectors of the following matrix:

$$\tilde{\mathbf{S}} = \mathbf{D}_n^{-\frac{1}{2}} \mathbf{S} \mathbf{D}_m^{-\frac{1}{2}}, \tag{6}$$

where $\mathbf{D}_n \in R^{n \times n}$ and $\mathbf{D}_m \in R^{m \times m}$ are diagonal matrices calculated as follows:

$$\mathbf{D} = \begin{bmatrix} \mathbf{D}_n & \\ & \mathbf{D}_m \end{bmatrix}. \tag{7}$$

Note that $\tilde{\mathbf{S}}$ is a sparse matrix the same as $\mathbf{S}$ since $\mathbf{D}_n$ and $\mathbf{D}_m$ are diagonal matrices. Let $\mathbf{U}_c \in R^{n \times c}$ and $\mathbf{V}_c \in R^{m \times c}$ be matrices of the leading c left and right singular vectors of $\tilde{\mathbf{S}}$ associated with the largest c singular values, respectively, $\frac{\sqrt{2}}{2}\mathbf{U}_c$ and $\frac{\sqrt{2}}{2}\mathbf{V}_c$ are the optimal solutions to the problem. As a result, $\mathbf{F}$ is given as follows:

$$\mathbf{F} = \frac{\sqrt{2}}{2} \begin{bmatrix} \mathbf{U}_c \\ \mathbf{V}_c \end{bmatrix}. \tag{8}$$

Since it needs to compute SVD on $\tilde{\mathbf{S}}$ to obtain $\frac{\sqrt{2}}{2}\mathbf{U}_c$ and $\frac{\sqrt{2}}{2}\mathbf{V}_c$, it requires a considerable computation time to compute $\mathbf{F}$.

When $\mathbf{F}$ and $\mathbf{A}$ are fixed, the problem of (4) becomes as follows:

$$\begin{aligned} &\min_{\mathbf{S}} \sum_{i=1}^{n}\sum_{j=1}^{m} s[i][j]\|\mathbf{x}_i - \mathbf{a}_j\|_2^2 + \alpha\|\mathbf{S}\|_F^2 + \beta\,\mathrm{tr}\,(\mathbf{F}^\top\mathbf{L}\mathbf{F}) \\ &\text{s.t. } \mathbf{S} \geq 0, \mathbf{S}\mathbf{1}_m = \mathbf{1}_m. \end{aligned} \tag{9}$$

This problem has a closed-form solution [32]. Specifically, each element of $\mathbf{S}$ is given as follows:

$$s[i][j] = \begin{cases} (D_{i,l+1} - D[i,j])/(lD_{i,l+1} - \sum_{\mathbf{a}_k \in \mathbb{N}_i} D[i,k]) & \text{if } \mathbf{a}_j \in \mathbb{N}_i \\ 0 & \text{otherwise} \end{cases}, \tag{10}$$

where l is the predefined hyper-parameter that gives the number of neighbor prototypes, $\mathbb{N}_i$ is the set of l nearest prototypes to the i-th data points, $D[i,j]$ is the distance between $\mathbf{x}_i$ and $\mathbf{a}_j$, and $D_{i,l+1}$ is the distance to the $(l+1)$-th nearest prototype of $\mathbf{x}_i$. In Equation (10), distance $D[i,j]$ is given as follows:

$$D[i,j] = \|\mathbf{x}_i - \mathbf{a}_j\|_2^2 + \beta \left\| \frac{\mathbf{f}_i}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j}}{\sqrt{d[n+j]}} \right\|_2^2, \tag{11}$$

where $\mathbf{f}_i$ is the i-th row vector of $\mathbf{F}$. To obtain the nearest prototypes of each data point, it needs to compute the distance of Equation (11) for all the pairs of data points and prototypes. Note that $\mathbf{S}$ is a sparse matrix where each row has l non-zero elements normalized as follows:

$$\sum_{j=1}^{m} s[i][j] = 1. \tag{12}$$

Moreover, α can be set to $\alpha = \frac{1}{n}\sum_{i=1}^{n}(\frac{l}{2}D_{i,l+1} - \frac{1}{2}\sum_{\mathbf{a}_k \in \mathbb{N}_i} D[i,k])$. As a result, regularization parameter α can be set by tuning the number of nearest neighbors, l.

When $\mathbf{F}$ and $\mathbf{S}$ are fixed, the problem of (4) becomes as follows:

$$\min_{\mathbf{A} \in R^{m \times d}} \sum_{i=1}^{n}\sum_{j=1}^{m} s[i][j]\|\mathbf{x}_i - \mathbf{a}_j\|_2^2. \tag{13}$$

Therefore, each prototype is computed as follows:

$$\mathbf{a}_j = \frac{\sum_{i=1}^{n} s[i][j]\mathbf{x}_i}{\sum_{i=1}^{n} s[i][j]}. \tag{14}$$

Although K-Multiple-Means can effectively compute the clusters, it incurs excessive processing time. Since the size of $\tilde{\mathbf{S}}$ of Equation (6) is $n \times m$, step 1 requires $O(nm^2)$ time to update $\mathbf{F}$ from Equation (8) by computing SVD on $\tilde{\mathbf{S}}$ to obtain the leading singular vectors. Since it needs $O(nm(c+d))$ time to compute the distances between data points and prototypes from Equation (11), step 2 takes $O(nm(c+d+\log m))$ time to update $\mathbf{S}$ by computing the nearest prototypes from Equation (10). After iteratively updating $\mathbf{F}$ and $\mathbf{S}$ until c connected components are determined, step 3 takes $O(nmd)$ time to update the prototypes from Equation (14). As a result, if t_c is the number of iterations to obtain the c connected components and t_a is the number of iterations to obtain the converged assignments, it requires $O(nm(d+c+m)t_c + nmdt_a)$ time to compute the clusters. Since K-Multiple-Means needs to compute SVD at $O(nm^2)$ time in step 1 iteratively, the computational costs are excessive for large-scale data. In addition, the original algorithm requires $O(n(m+d))$ space. This is because it needs $O(nd)$ space to hold the data points, and the sizes of matrix $\mathbf{F}$, $\mathbf{S}$, and $\mathbf{A}$ are $c(n+m)$, nm, and md, respectively where $c < m < n$.

4 PROPOSED METHOD

This section explains our approach. Section 4.1 overviews the ideas that underlie our approach. Section 4.2 describes our approach to computing the leading c left and right singular vectors from the similarity matrix between the prototypes. Section 4.3 shows our approach for efficiently computing the similarity matrix of the bipartite graph. Section 4.4 details our clustering algorithm and its properties.

4.1 Main Ideas

We introduce the idea of using the following $m \times m$ matrix $\mathbf{M}$ instead of $\tilde{\mathbf{S}}$ in updating matrix $\mathbf{F}$:

$$\mathbf{M} = \tilde{\mathbf{S}}^\top \tilde{\mathbf{S}}. \tag{15}$$

Since $n \times m$ matrix $\tilde{\mathbf{S}}$ corresponds to the similarities between the data points and the prototypes from Equation (6), $m \times m$ matrix $\mathbf{M}$ corresponds to the similarities between the prototypes.

This approach has two advantages. First, we can exactly compute the leading singular vectors from $\mathbf{M}$ (Section 4.2). Since the size of $\tilde{\mathbf{S}}$ is $n \times m$ where $m < n$, the rank of $\tilde{\mathbf{S}}$ is at most m [46]. In addition, it is clear that $\mathbf{M}$ is symmetric from Equation (15). Therefore, eigenvalues of $\mathbf{M}$ are associated with singular values of $\tilde{\mathbf{S}}$. As a result, we can obtain singular vectors of $\tilde{\mathbf{S}}$ by computing eigenvectors of $\mathbf{M}$. Second, we can efficiently compute the leading singular vectors from $\mathbf{M}$ (Section 4.2). As mentioned in Section 3, K-Multiple-Means suffers from high computation costs since it iteratively computes SVD on $n \times m$ matrix $\tilde{\mathbf{S}}$ in updating matrix $\mathbf{F}$; each iteration takes $O(nm^2)$ time to obtain the leading singular vectors by computing SVD. On the other hand, since the number of prototypes, m, is much smaller than that of the data points, n, in practice (i.e., $m \ll n$), we can efficiently compute eigenvectors of the $m \times m$ matrix $\mathbf{M}$. Moreover, as each iteration breaks the $m \times m$ similarity matrix into small block matrices corresponding to connected components, we can improve the efficiency of computing the leading singular vectors from the eigenvectors of the small block matrices. Although it needs to compute $\tilde{\mathbf{S}}$ to obtain $\mathbf{M}$, as shown in Equation (15), we can efficiently compute $\tilde{\mathbf{S}}$ by skipping unnecessary distance computations and estimating lower bounding distances between the data points and the prototypes (Section 4.3).

4.2 Efficient Singular Vector Computation

In this section, we show how to compute the singular vectors from the block matrices efficiently. Specifically, Section 4.2.1 describes the approach used to compute the block matrices of the $m \times m$ similarity matrix between the prototypes. Section 4.2.2 shows how we efficiently compute the eigenvectors of the block matrices.

4.2.1 Block Matrix Computation. Our approach uses the $m \times m$ similarity matrix $\mathbf{M}$ of Equation (15) to avoid computing SVD on $n \times m$ matrix $\tilde{\mathbf{S}}$. We have the following property for $\mathbf{M}$:

Lemma 4.1. *Let λ_i be the i-th largest eigenvalue of symmetric matrix $\mathbf{M}$, $\mathbf{q}_i$ be the eigenvector associated with λ_i, and $\mathbf{u}_i$ and $\mathbf{v}_i$ be the leading left and right singular vectors associated with the i-th largest singular value σ_i of $\tilde{\mathbf{S}}$, respectively. $\mathbf{M}$ has non-negative eigenvalues such that $\lambda_i = \sigma_i^2$, and the i-th leading left and right singular vectors of $\tilde{\mathbf{S}}$ can be computed as follows:*

$$\mathbf{u}_i = \frac{1}{\sqrt{\lambda_i}} \tilde{\mathbf{S}} \mathbf{q}_i, \ \mathbf{v}_i = \mathbf{q}_i. \tag{16}$$

Proof. Let σ_i be the i-th largest singular value of $\tilde{\mathbf{S}}$, $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_m)$ be a diagonal matrix of singular values, and $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_m]$ and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ be matrices of the left and right singular vectors, respectively. Note that singular values are non-negative; $\sigma_i \geq 0$ holds [46]. $\mathbf{M}$ is computed as $\mathbf{M} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^\top\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^\top$. Moreover, if $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_m)$ is a diagonal matrix of eigenvalues of $\mathbf{M}$ and $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_m]$ is a matrix of eigenvectors, we have $\mathbf{M} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$. As a result, since $\mathbf{V}\mathbf{V}^\top = \mathbf{V}^\top\mathbf{V} = \mathbf{I}$ and $\mathbf{Q}\mathbf{Q}^\top = \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$, we have $\mathbf{V} = \mathbf{Q}$ and thus $\mathbf{\Sigma}^2 = \mathbf{\Lambda}$. Therefore, $\mathbf{v}_i = \mathbf{q}_i$ and $\lambda_i = \sigma_i^2$ hold. Moreover, since $\tilde{\mathbf{S}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, we have $\mathbf{U} = \tilde{\mathbf{S}}\mathbf{V}\mathbf{\Sigma}^{-1}$. Therefore, $\mathbf{u}_i = (\sigma_i)^{-1}\tilde{\mathbf{S}}\mathbf{v}_i = \frac{1}{\sqrt{\lambda_i}}\tilde{\mathbf{S}}\mathbf{q}_i$. □

Lemma 4.1 indicates that we can compute the leading left and right singular vectors of $\tilde{\mathbf{S}}$ from the eigendecomposition of $m \times m$ matrix $\mathbf{M}$; we do not need to compute SVD from $n \times m$ matrix $\tilde{\mathbf{S}}$.

In addition, since the similarities between the data points and the prototypes are updated to satisfy the rank constraint, the bipartite graph between the data points and the prototypes is divided into connected components in each iteration; $\mathbf{M}$ is broken into block matrices in the iterations. Specifically, if $\mathbf{M}'$ is a prototype-permutated matrix of $\mathbf{M}$, m_i is the number of prototypes included in the i-th connected component, $\mathbf{M}'_i$ is the i-th block matrix of $m_i \times m_i$, and b is the number of the block matrices corresponding to the connected components, we have the following equation:

$$\mathbf{M}' = \mathbf{P}\mathbf{M}\mathbf{P}^\top = \begin{bmatrix} \mathbf{M}'_1 & & \\ & \ddots & \\ & & \mathbf{M}'_b \end{bmatrix}, \tag{17}$$

where $\mathbf{P}$ is a prototype-permutation matrix that yields the block matrices. $\mathbf{P}$ is an $m \times m$ orthogonal matrix where every row and column contains a single 1 with 0s everywhere else; $p[i][j] = 1$ indicates that the i-th row is permutated into the j-th row.

Algorithm 1 shows the approach to computing the block matrices by obtaining $\mathbf{P}$. In this algorithm, $\mathbf{0}_n = [0, \ldots, 0]$ is a row vector of length n, $\tilde{\mathbf{c}}_i$ is the i-th column vector of $\tilde{\mathbf{S}}$, $\mathbf{r}$ is a row vector of length n, and $\mathbb{b}_i$ be the set of prototypes included in the i-th connected component. We first set elements of $\mathbf{r}$ from non-zero elements of $\tilde{\mathbf{S}}$ (lines 2-4). Since $\mathbf{M} = \tilde{\mathbf{S}}^\top \tilde{\mathbf{S}}$ from Equation (15), we compute element $m[i][j]$ of $\mathbf{M}$ by computing the inner product of $\mathbf{r}$ and $\tilde{\mathbf{s}}_i$ (line 5-8). Then, the connected components in $\mathbf{M}$ and prototype permutation matrix $\mathbf{P}$ (line 9-15) are computed. Finally, the prototype permutated matrix is computed from Equation (17) (line 16). Algorithm 1 can efficiently compute the block matrices using the sparsity of $\tilde{\mathbf{S}}$.

Lemma 4.2. *Algorithm 1 takes $O(nml)$ time to compute the block matrices from matrix $\tilde{\mathbf{S}}$.*

Proof. It needs $O(m^2)$ time to set vector $\mathbf{w}$ of length m. Since $\tilde{s}_i$ would have $\frac{nl}{m}$ non-zero elements and $\mathbf{M}$ is a matrix of $m \times m$, it takes $O(nml)$ time to compute $\mathbf{M}$. It requires $O(m^2)$ time to compute the connected components of $\mathbf{M}$ [6]. Since $\mathbf{M}'$ is a permutation matrix of $\mathbf{M}$, it needs $O(m^2)$ time to compute $\mathbf{M}'$. As a result, since we have $m \ll n$, the computation cost of Algorithm 1 is $O(nml)$. □

Since $\mathbf{M}'$ of Equation (17) is a block-diagonal matrix, we can compute eigenvectors of $\mathbf{M}$ from the block matrices [18]. Specifically, if $\lambda_{i,j}$ is the j-th largest eigenvalue of $\mathbf{M}'_i$, $\mathbf{q}_{i,j}$ is the eigenvector of $\mathbf{M}'_i$ associated with $\lambda_{i,j}$, $\lambda_{j'}$ is the eigenvalue of $\mathbf{M}$ corresponding to $\lambda_{i,j}$, and $\mathbf{q}_{j'}$ is the eigenvector of $\mathbf{M}$ associated with $\lambda_{j'}$, we have

$$\lambda_{j'} = \lambda_{i,j},\ \mathbf{q}_{j'} = \mathbf{P}^\top \left[\mathbf{0}_{\sum_{k=1}^{i-1} m_k}, \mathbf{q}_{i,j}^\top, \mathbf{0}_{\sum_{k=i+1}^{b} m_k}\right]^\top. \tag{18}$$

As a result, we can efficiently compute the eigenvalues and eigenvectors of $\mathbf{M}$ from the block matrices.

4.2.2 Eigenvector Computation. As shown in the previous section, we can compute the leading c left and right singular vectors of $\tilde{\mathbf{S}}$ from the c largest eigenvalues and their eigenvectors of the block matrices. From Lemma 4.1, the eigenvalues of $\mathbf{M}$ and the singular values of matrix $\tilde{\mathbf{S}}$ are non-negative. Therefore, a naive approach to obtain the leading c singular vectors is to compute the c largest eigenvalues and their eigenvectors for each block matrix using the power method [7]. However, since this approach computes c eigenvectors of b block matrices, it needs to use the power method bc time. To reduce the number of computations, the proposed approach uses the following property of $\mathbf{M}'_i$:

Lemma 4.3. *Let $\mathbf{b}$ be a column vector of length m such that $\mathbf{b} = \mathbf{D}_m^{\frac{1}{2}} \mathbf{1}_m = \left[\sqrt{d[n+1]}, \ldots, \sqrt{d[n+m]}\right]^\top$, $\mathbf{P}_i$ be the i-th $m \times m_i$ sub-matrix of $\mathbf{P}$ such that $\mathbf{P} = [\mathbf{P}_1, \ldots, \mathbf{P}_b]$, and $\mathbf{b}'_i$ be a column vector of length*

Algorithm 1 Block Matrix Computation

Input: matrix $\tilde{\mathbf{S}}$
Output: block matrices $\mathbf{M}'_1, \ldots, \mathbf{M}'_b$
for $i = 1$ to m **do**
  $\mathbf{r} = \mathbf{0}_n$;
  for each non-zero element $\tilde{c}[i][j] \in \tilde{\mathbf{c}}_i$ **do**
    $r[j] = \tilde{c}[i][j]$;
  for $j = 1$ to m **do**
    $m[i][j] = 0$;
    for each non-zero element $\tilde{c}[j][k] \in \tilde{\mathbf{c}}_j$ **do**
      $m[i][j] = m[i][j] + r[k]\tilde{c}[j][k]$;
compute connected components in $\mathbf{M}$;
$\mathbf{P} = \mathbf{0}$;
$k = 1$;
for $i = 1$ to b **do**
  for each prototype $a_j \in \mathbb{b}_i$ **do**
    $p[k][j] = 1$;
    $k = k + 1$;
$\mathbf{M}' = \mathbf{PMP}^\top$;

Algorithm 2 Eigenvector Computation

Input: number of clusters c, block matrices $\mathbf{M}'_1, \ldots, \mathbf{M}'_b$
Output: eigenvalues $\lambda_1, \ldots, \lambda_c$, eigenvectors $\mathbf{q}_1, \ldots, \mathbf{q}_c$
$\mathbb{E} = \emptyset$;
add c dummy eigenvalues to $\mathbb{E}$;
for each block matrix $\mathbf{M}'_i$ **do**
  set pair $(i, j) = (i, 1)$;
  $\lambda_{i,j} = 1$;
  $\mathbf{q}_{i,j} = \mathbf{b}'_i$;
  $\mathbf{M}'_i = \mathbf{M}'_i - \mathbf{q}_{i,j}\lambda_{i,j}\mathbf{q}_{i,j}^\top$;
  add $\lambda_{i,j}$ to $\mathbb{E}$;
  subtract $\lambda_{i',j'} = \min_{\mathbb{E}}\{\lambda_{i',j'}\}$ from $\mathbb{E}$;
  update pair $(i, j) = (i, j + 1)$;
if $b < c$ **then**
  repeat
    $\mathbf{M}'_i = \mathrm{argmax}_{\mathbb{M}'}\{\overline{\lambda}_{i,j}\}$;
    if $\overline{\lambda}_{i,j} \geq \min_{\mathbb{E}}\{\lambda_{i',j'}\}$ holds for $\mathbf{M}'_i$ **then**
      compute $\lambda_{i,j}$ and $\mathbf{q}_{i,j}$ by the power method;
      $\mathbf{M}'_i = \mathbf{M}'_i - \mathbf{q}_{i,j}\lambda_{i,j}\mathbf{q}_{i,j}^\top$;
      if $\lambda_{i,j} \geq \min_{\mathbb{E}}\{\lambda_{i',j'}\}$ **then**
        add $\lambda_{i,j}$ to $\mathbb{E}$;
        subtract $\lambda_{i',j'} = \min_{\mathbb{E}}\{\lambda_{i',j'}\}$ from $\mathbb{E}$;
      update pair $(i, j) = (i, j + 1)$;
  until $\overline{\lambda}_{i,j} < \min_{\mathbb{E}}\{\lambda_{i',j'}\}$ holds for $\mathbf{M}'_i$
for each $\lambda_{i,j} \in \mathbb{E}$ **do**
  compute $\mathbf{q}_{j'}$ from $\mathbf{q}_{i,j}$ associated with $\lambda_{i,j}$ by Equation (18);

m_i such that $\mathbf{b}'_i = \mathbf{P}_i^\top \mathbf{b}$. The largest eigenvalue of the i-th block matrix $\mathbf{M}'_i$ is $\lambda_{i,1} = 1$ and its associated eigenvector is $\mathbf{q}_{i,1} = \mathbf{b}'_i$.

Proof. Let $\mathbf{B}_i$ be the i-th $m_i \times m_i$ block matrix of matrix $\mathbf{PS}^\top\mathbf{SP}^\top$ such that

$$\mathbf{PS}^\top\mathbf{SP}^\top = \begin{bmatrix} \mathbf{B}_1 & & \\ & \ddots & \\ & & \mathbf{B}_b \end{bmatrix}. \tag{19}$$

Note that $\mathbf{B}_i$ corresponds to the similarities between the prototypes in the i-th connected component. If $\mathbf{L}'_i$ is the normalized Laplacian matrix of $\mathbf{B}_i$, it is given as follows:

$$\mathbf{L}'_i = \mathbf{I} - \mathbf{D}_i'^{-\frac{1}{2}}\mathbf{B}_i\mathbf{D}_i'^{-\frac{1}{2}},\ \mathbf{D}'_i = \begin{bmatrix} \sum_{j=1}^{m_i} b_i[1][j] & & \\ & \ddots & \\ & & \sum_{j=1}^{m_i} b_i[m_i][j] \end{bmatrix}. \tag{20}$$

Since $\mathbf{B}_i$ has a single connected component, $\mathbf{L}'_i$ has single eigenvalue 0 and its associated eigenvector is given as $\mathbf{D}_i'^{\frac{1}{2}}\mathbf{1}_{m_i}$ [41]. Moreover, since the (i, j)-th element of $\mathbf{S}^\top\mathbf{S}$ is given as $\sum_{k=1}^{n} s[k][i]s[k][j]$ and $\sum_{j=1}^{m} s[i][j] = 1$ from Equation (12), the sum of the i-th row elements of $\mathbf{S}^\top\mathbf{S}$ is computed as

$$\sum_{j=1}^{m}\sum_{k=1}^{n} s[k][i]s[k][j] = \sum_{k=1}^{n} s[k][i]. \tag{21}$$

In addition, from Equation (1) and (2), we have

$$d[n + i] = \sum_{j=1}^{n+m} w[i][j] = \sum_{k=1}^{n} s[k][i]. \tag{22}$$

Therefore, the sum of the i-th row elements of $\mathbf{S}^\top\mathbf{S}$ is given as

$$\sum_{j=1}^{m}\sum_{k=1}^{n} s[k][i]s[k][j] = d[n + i]. \tag{23}$$

As shown in Equation (19), matrix $\mathbf{P}\mathbf{S}^\top\mathbf{S}\mathbf{P}^\top$ is the row/column permutated matrix of $\mathbf{S}^\top\mathbf{S}$. Therefore, if the i-th row of matrix $\mathbf{S}^\top\mathbf{S}$ is permutated to the i'-th row of block matrix $\mathbf{B}_i$, we have the following equation for the i'-th diagonal element of $\mathbf{D}'_i$ of Equation (20):

$$\sum_{j=1}^{m_i} b_i[i'][j] = \sum_{j=1}^{m}\sum_{k=1}^{n} s[k][i]s[k][j] = d[n+i]. \tag{24}$$

As a result, since we have $\mathbf{b}'_i = \mathbf{P}_i^\top\mathbf{b} = \mathbf{P}_i^\top\mathbf{D}_m^{\frac{1}{2}}\mathbf{1}_m = \mathbf{D}_i'^{\frac{1}{2}}\mathbf{1}_{m_i}$, vector $\mathbf{b}'_i$ is the eigenvector of $\mathbf{L}'_i$ associated with eigenvalue 0.

In addition, since $\mathbf{D}_n = \mathbf{I}$ holds from Equation (1), (2), and (7), the following equation holds from Equation (6), (15), and (17):

$$\mathbf{M}' = \mathbf{P}\mathbf{D}_m^{-\frac{1}{2}}\mathbf{S}^\top\mathbf{S}\mathbf{D}_m^{-\frac{1}{2}}\mathbf{P}^\top. \tag{25}$$

As a result, from Equation (17) and (19), we have the following equation for each block matrix:

$$\mathbf{M}'_i = \mathbf{D}_i'^{-\frac{1}{2}}\mathbf{B}_i\mathbf{D}_i'^{-\frac{1}{2}}. \tag{26}$$

On the other hand, if $\lambda'_{i,j}$ is the j-th largest eigenvalue of $\mathbf{L}'_i$, the following property holds for $\mathbf{L}'_i$ [5]:

$$2 \geq \lambda'_{i,1} \geq \ldots \geq \lambda'_{i,m_{i-1}} > \lambda'_{i,m_i} = 0. \tag{27}$$

As a result, since $\mathbf{L}'_i = \mathbf{I} - \mathbf{M}'_i$ from Equation (20) and $\mathbf{M}'_i$ has non-negative eigenvalues from Lemma 4.1 and Equation (18), we have the following property for each eigenvalue of $\mathbf{M}'_i$:

$$1 = \lambda_{i,1} > \lambda_{i,2} \geq \ldots \geq \lambda_{i,m_{i-1}} = 0. \tag{28}$$

Moreover, since $\mathbf{L}'_i = \mathbf{I} - \mathbf{M}'_i$ has an eigenvector of $\mathbf{b}'_i$ associated with eigenvalue 0, we have $\mathbf{q}_{i,1} = \mathbf{b}'_i$ associating with $\lambda_{i,1} = 1$ for matrix $\mathbf{M}'_i$. □

Note that, the eigenvector obtained by Lemma 4.3 is not normalized. Therefore, the proposed approach uses the following normalized eigenvector $\bar{\mathbf{q}}_{i,1}$ instead of $\mathbf{q}_{i,1}$:

$$\bar{\mathbf{q}}_{i,1} = \frac{1}{\|\mathbf{q}_{i,1}\|_2}\mathbf{q}_{i,1}. \tag{29}$$

Since we can compute vector $\mathbf{q}_{i,1}$ from the elements of diagonal matrix $\mathbf{D}$, Lemma 4.3 indicates that we can efficiently compute the largest eigenvalues and their eigenvectors of the block matrices without using the power method. However, since we need to compute the c largest eigenvalues and their eigenvectors of the block matrices, if the number of block matrices, b, is smaller than the number of clusters, c (i.e., $b < c$), we need to additionally compute $c - b$ eigenvalues and eigenvectors of the block matrices by using the power method. To efficiently compute the additional eigenvalues and eigenvectors, we prune unnecessary computations by computing the upper bounds of eigenvalues as follows:

Definition 4.4. For block matrix $\mathbf{M}'_i$, let $\overline{\lambda}_{i,j}$ be the upper bound of $\lambda_{i,j}$ such that $j > 1$, we compute $\overline{\lambda}_{i,j}$ as follows:

$$\overline{\lambda}_{i,j} = \min(\lambda_{i,j-1}, \delta_{i,j}),\ \delta_{i,j} = \begin{cases} \sum_{k=1}^{m_i} m'_i[k][k] - \lambda_{i,1} & \text{if } j = 2 \\ \delta_{i,j-1} - \lambda_{i,j-1} & \text{otherwise} \end{cases}. \tag{30}$$

We have the following property for bound $\overline{\lambda}_{i,j}$:

Lemma 4.5. *For a block matrix $\mathbf{M}'_i$, it holds that $\overline{\lambda}_{i,j} \geq \lambda_{i,j}$.*

Proof. We have $\delta_{i,j} = \sum_{k=1}^{m_i} m'_i[k][k] - \lambda_{i,1}$ if $j = 2$, and we have $\delta_{i,j} = \delta_{i,j-1} - \lambda_{i,j-1}$ otherwise. Therefore, $\delta_{i,j} = \sum_{k=1}^{m_i} m'_i[k][k] - \sum_{k=1}^{j-1}\lambda_{i,k}$. Moreover, $\sum_{k=1}^{m_i} m'_i[k][k] = \mathrm{tr}(\mathbf{M}'_i) = \sum_{k=1}^{m_i}\lambda_{i,k}$ holds [7], and the eigenvalues have non-negative property, as shown in Lemma 4.1. Therefore, we have

$$\lambda_{i,j} \leq \sum_{k=j}^{m_i}\lambda_{i,k} = \sum_{k=1}^{m_i}\lambda_{i,k} - \sum_{k=1}^{j-1}\lambda_{i,k} = \delta_{i,j}. \tag{31}$$

In addition, it is clear that $\lambda_{i,j-1} \geq \lambda_{i,j}$ holds. Therefore, we have $\overline{\lambda}_{i,j} = \min(\lambda_{i,j-1}, \delta_{i,j}) \geq \lambda_{i,j}$. □

Algorithm 2 details the approach that efficiently computes the eigenvectors of the block matrices corresponding to the c largest eigenvalues by using Lemma 4.3 and 4.5. In Algorithm 2, $\mathbb{E}$ is a set of the c largest eigenvalues and $\mathbb{M}'$ is the set of b block matrices. Algorithm 2 initializes $\mathbb{E}$ by adding c dummy eigenvalues whose values are 0 (line 1-2). It then obtains the largest eigenvalues and their eigenvectors of the block matrices by using Lemma 4.3 (line 3-10). Specifically, for block matrix $\mathbf{M}'_i$, it sets pair (i, j) to $(i, 1)$ in order to specify the largest eigenvalue of the block matrix (line 4). It then sets the largest eigenvalue and its eigenvector (line 5-6). From $\mathbf{M}'_i$, it subtracts $\mathbf{q}_{i,j}\lambda_{i,j}\mathbf{q}_{i,j}^\top$ that corresponds to the matrix of the largest eigenvalues to compute the second largest eigenvalue (line 7). It then updates $\mathbb{E}$ by $\lambda_{i,j}$ (line 8-9). It also updates pair (i, j) to $(i, j+1)$ in order to specify the second largest eigenvalue (line 10). If $b < c$, we efficiently compute the additional eigenvalues and eigenvectors using Lemma 4.5 (line 11-21). Specifically, it specifies the block matrix that gives the maximum upper bound (line 13). If the maximum upper bound is not smaller than the smallest obtained eigenvalue, it uses the power method to compute the eigenvalue and eigenvector of the block matrix (line 15). It then updates $\mathbf{M}'_i$, $\mathbb{E}$, and (i, j) (line 16-20). It iterates these procedures until the maximum upper bound is smaller than the smallest obtained eigenvalue (line 21). It finally computes the c largest eigenvalues and their eigenvectors of $\mathbf{M}$ from Equation (18) (line 22-23). We have the following property for Algorithm 2:

Lemma 4.6. *Let t_p denote the number of iterations used in the power method. Algorithm 2 computes the c largest eigenvalues and their eigenvectors of matrix $\mathbf{M}$ in time of $O((\frac{m}{b})^2 t_p + \frac{mc}{b})$.*

Proof. Algorithm 2 needs $O(m)$ time to obtain the largest eigenvalue and its eigenvector of each block matrix. It takes $O(m)$ time to compute the upper bounds of eigenvalues from Equation (30). Since the size of each block matrix would be $\frac{m}{b} \times \frac{m}{b}$, the power method requires $O((\frac{m}{b})^2 t_p)$ time [7]. Since the length of $\mathbf{q}_{i,j}$ would be $\frac{m}{b}$, it needs $O(\frac{mc}{b})$ time to obtain eigenvectors of $\mathbf{M}$ by Equation (18). As a result, Algorithm 2 takes $O((\frac{m}{b})^2 t_p + \frac{mc}{b})$ time to compute the c largest eigenvalues and their eigenvector of $\mathbf{M}$. □

As described in Section 3, the original approach of K-Multiple-Means computes SVD on matrix $\tilde{\mathbf{S}}$ of $n \times m$ to obtain the leading c left and right singular vectors. On the other hand, as shown in Algorithm 2, the proposed approach uses the smaller block matrices to compute the largest eigenvalues and their eigenvectors. As a result, since the leading singular vectors can be obtained from the eigenvalues and their eigenvectors as shown in Lemma 4.1, our approach can compute the leading singular vectors more efficiently than the original approach.

4.3 Efficient Similarity Matrix Computation

The proposed approach uses $\tilde{\mathbf{S}}$ to obtain the block matrix, as shown in Equation (15). From Equation (6), we have $\tilde{\mathbf{S}} = \mathbf{D}_n^{-\frac{1}{2}} \mathbf{S} \mathbf{D}_m^{-\frac{1}{2}}$ and $\mathbf{S}$ is obtained by computing the nearest prototypes for each data point, as shown in Equation (10). Therefore, we need to compute the nearest prototypes based on the following distance to obtain $\tilde{\mathbf{S}}$:

$$D[i, j] = D_E[i, j] + \beta D_F[i, j], \tag{32}$$

where

$$D_E[i, j] = \|\mathbf{x}_i - \mathbf{a}_j\|_2^2 \tag{33}$$

and

$$D_F[i, j] = \left\| \frac{\mathbf{f}_i}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j}}{\sqrt{d[n+j]}} \right\|_2^2. \tag{34}$$

Note that β is the regularization parameter of the optimization problem. $D_E[i, j]$ is the distance of d-length vectors $\mathbf{x}_i$ and $\mathbf{a}_j$. $D_F[i, j]$ is the distance of c-length vectors $\mathbf{f}_i/\sqrt{d[i]}$ and $\mathbf{f}_{n+j}/\sqrt{d[n+j]}$. Therefore, if the number of dimensions, d, and the number of clusters, c, have large values, the computation costs of distance $D[i, j]$ would become high, necessitating a considerable computation time to obtain $\tilde{\mathbf{S}}$. This section describes our approach that efficiently computes $\tilde{\mathbf{S}}$ to obtain $\mathbf{M}$. Section 4.3.1 describes the approach to computing $D_F[i, j]$ by skipping unnecessary distance computations. Section 4.3.2 shows how our approach estimates lower bounding distances between the data points and the prototypes.

4.3.1 Skipping Distance Computations. In this section, to simplify the presentation, we assume that $b < c$, that is, the number of block matrices is smaller than the number of clusters. As described in Section 4.2.1, the block matrices correspond to the connected components. Therefore, if $b < c$, the number of connected components is smaller than the number of clusters. Let $\mathbf{f}_{i,b}$ be a b-length sub-vector of $\mathbf{f}_i$ such that $\mathbf{f}_i = [\mathbf{f}_{i,b}, \mathbf{f}_{i,c-b}]$, and $\mathbf{f}_{n+j,b}$ be a b-length sub-vector of $\mathbf{f}_{n+j}$ such that $\mathbf{f}_{n+j} = [\mathbf{f}_{n+j,b}, \mathbf{f}_{n+j,c-b}]$. We have the following property for $\mathbf{f}_{i,b}$ and $\mathbf{f}_{n+j,b}$ such that $b < c$:

Lemma 4.7. *If $\mathbb{c}_i$ is the connected component that includes $\mathbf{x}_i$ and $\mathbb{c}_j$ is the connected component that includes $\mathbf{a}_j$, we have*

$$\left\| \frac{\mathbf{f}_{i,b}}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j,b}}{\sqrt{d[n+j]}} \right\|_2^2 = \begin{cases} 0 & \text{if } \mathbb{c}_i = \mathbb{c}_j \\ \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right) & \text{otherwise} \end{cases}, \tag{35}$$

where n_i and n_j are the numbers of data points included in $\mathbb{c}_i$ and $\mathbb{c}_j$, respectively.

Proof. The b eigenvectors of the largest eigenvalues of the block matrices associate with the leading singular vectors of $\tilde{\mathbf{S}}$, as shown in Equation (16). Therefore, sub-vector $\mathbf{f}_{i,b}$ is associated with the b eigenvectors of the largest eigenvalues from Equation (8). As a result, let $\mathbf{u}_k$ be the k-th column vector of $\mathbf{U}_c$ such that $k \leq b$, we have the following equation from Lemma 4.3 since we have $\mathbf{U} = \tilde{\mathbf{S}}\mathbf{V}\Sigma^{-1}$ and $\mathbf{V} = \mathbf{Q}$ as shown in Lemma 4.1:

$$\mathbf{u}_k = \tilde{\mathbf{S}}\mathbf{q}_k. \tag{36}$$

In the case that $\mathbf{x}_i$ is included in the k-th connected component, we have the following equation from Equation (18) and (29) if connected component $\mathbb{c}_i$ associates with $\mathbf{M}'_{k'}$:

$$\mathbf{u}_k = \frac{1}{\|\mathbf{q}_{k',1}\|_2}\tilde{\mathbf{S}}\mathbf{P}^\top \left[\mathbf{0}_{\sum_{k''=1}^{k'-1} m_{k''}}, \mathbf{q}_{k',1}^\top, \mathbf{0}_{\sum_{k''=k'+1}^{b} m_{k''}}\right]^\top. \tag{37}$$

Since $\mathbf{D}_n = \mathbf{I}$ holds from Equation (1), (2), and (7), we have $d[i] = 1$ and $\tilde{\mathbf{S}}\mathbf{P}^\top = \mathbf{S}\mathbf{D}_m^{-\frac{1}{2}}\mathbf{P}^\top$ from Equation (6). Therefore, if $\mathbf{s}'_i$ is an $m_{k'}$ length sub-vector of the i-th row vector of matrix $\mathbf{S}\mathbf{D}_m^{-\frac{1}{2}}\mathbf{P}^\top$ associated with $\mathbf{M}'_{k'}$, we have the following equation from Equation (8):

$$\frac{f_{i,b}[k]}{\sqrt{d[i]}} = \frac{\sqrt{2}}{2\|\mathbf{q}_{k',1}\|_2}\mathbf{s}'_i\mathbf{q}_{k',1}. \tag{38}$$

As a result, since i-th row vector $\mathbf{s}'_i$ of matrix $\mathbf{S}\mathbf{D}_m^{-\frac{1}{2}}\mathbf{P}^\top$ is associated with block matrix $\mathbf{M}'_{k'}$, we have

$$\frac{f_{i,b}[k]}{\sqrt{d[i]}} = \frac{\sqrt{2}\left(\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} \frac{s[i][k'']}{\sqrt{d[n+k'']}}\sqrt{d[n+k'']}\right)}{2\sqrt{\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}}\left(\sqrt{d[n+k'']}\right)^2}} = \frac{\sqrt{2}(\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} s[i][k''])}{2\sqrt{\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} d[n+k'']}}. \tag{39}$$

We have $\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} s[i][k''] = 1$ since each row of $\mathbf{S}$ is normalized as shown in Equation (12). Therefore, from Equation (1), (2), and (12), we have

$$\sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} d[n+k''] = \sum_{\mathbf{x}_{i'} \in \mathbb{c}_k} \sum_{\mathbf{a}_{k''} \in \mathbb{b}_{k'}} s[i'][k''] = n_k. \tag{40}$$

As a result, in the case that $\mathbf{x}_i$ is included in the k-th connected component, we have the following equation from Equation (40):

$$\frac{f_{i,b}[k]}{\sqrt{d[i]}} = \frac{1}{\sqrt{2n_k}}. \tag{41}$$

On the other hand, in the case that $\mathbf{x}_i$ is not included in the k-th connected component, from Equation (18), we have,

$$\frac{f_{i,b}[k]}{\sqrt{d[i]}} = 0. \tag{42}$$

In addition, since each eigenvector is normalized as per Equation (29), in the case that $\mathbf{a}_j$ is included in the k-th connected component, we have the following equation from Equation (8), (16) and (29) if $\mathbf{a}_j$ associates with the j'-th row of $\mathbf{M}'_{k'}$:

$$f_{n+j,b}[k] = \frac{\sqrt{2}}{2}\frac{1}{\|\mathbf{q}_{k',1}\|_2} q_{k',1}[j']. \tag{43}$$

We have the following equation from Lemma 4.3:

$$\frac{f_{n+j,b}[k]}{\sqrt{d[n+j]}} = \frac{1}{\sqrt{d[n+j]}}\frac{\sqrt{2}}{2}\frac{1}{\sqrt{\sum_{\mathbf{a}_{k''}\in\mathbb{b}_{k'}}\left(\sqrt{d[n+k'']}\right)^2}}\sqrt{d[n+j]} = \frac{1}{\sqrt{2\sum_{\mathbf{a}_{k''}\in\mathbb{b}_{k'}} d[n+k'']}}. \tag{44}$$

As a result, from Equation (40), in the case that $\mathbf{a}_j$ is included in the k-th connected component, we have

$$\frac{f_{n+j,b}[k]}{\sqrt{d[n+j]}} = \frac{1}{\sqrt{2n_k}}. \tag{45}$$

On the other hand, in the case that $\mathbf{a}_j$ is not included in the k-th connected component, from Equation (18), we have

$$\frac{f_{n+j,b}[k]}{\sqrt{d[n+j]}} = 0. \tag{46}$$

Consequently, from Equation (41), (42), (45), and (46), if $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in the same connected component, we have

$$\frac{\mathbf{f}_{i,b}}{\sqrt{d[i]}} = \frac{\mathbf{f}_{n+j,b}}{\sqrt{d[n+j]}}. \tag{47}$$

Therefore, we have

$$\left\|\frac{\mathbf{f}_{i,b}}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j,b}}{\sqrt{d[n+j]}}\right\|_2^2 = 0. \tag{48}$$

If $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in different connected components,

$$\left\|\frac{\mathbf{f}_{i,b}}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j,b}}{\sqrt{d[n+j]}}\right\|_2^2 = \left(\frac{1}{\sqrt{2n_i}}\right)^2 + \left(\frac{1}{\sqrt{2n_j}}\right)^2 = \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right), \tag{49}$$

which completes the proof. □

Since $\mathbf{f}_{i,b}$ and $\mathbf{f}_{n+j,b}$ correspond to the first b elements of $\mathbf{f}_i$ and $\mathbf{f}_{n+j}$, respectively, this lemma indicates that we can compute distance $D_F[i,j]$ in $O(1)$ time instead of $O(b)$ time for the first b elements. From Lemma 4.7, we introduce the following property:

Lemma 4.8. *Let $f_{i,c-b} = \|\mathbf{f}_{i,c-b}\|_2^2$ and $f_{j,c-b} = \|\mathbf{f}_{j,c-b}\|_2^2$, if $b < c$, distance $D_F[i,j]$ is computed as*

$$D_F[i,j] = \begin{cases} \left\|\mathbf{f}_{i,c-b} - \frac{1}{\sqrt{d[n+j]}}\mathbf{f}_{j,c-b}\right\|_2^2 & \text{if } \mathbb{c}_i = \mathbb{c}_j \\ \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right) + f_{i,c-b} + \frac{1}{d[n+j]} f_{j,c-b} & \text{otherwise} \end{cases}. \tag{50}$$

Proof. Since $\mathbf{f}_i = [\mathbf{f}_{i,b}, \mathbf{f}_{i,c-b}]$, we have

$$D_F[i,j] = \left\| \frac{\mathbf{f}_{i,b}}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j,b}}{\sqrt{d[n+j]}} \right\|_2^2 + \left\| \frac{\mathbf{f}_{i,c-b}}{\sqrt{d[i]}} - \frac{\mathbf{f}_{n+j,c-b}}{\sqrt{d[n+j]}} \right\|_2^2. \tag{51}$$

If $\mathbb{c}_i = \mathbb{c}_j$, since $d[i] = 1$, we have the following equation from Lemma 4.7:

$$D_F[i,j] = \left\| \mathbf{f}_{i,c-b} - \frac{1}{\sqrt{d[n+j]}} \mathbf{f}_{j,c-b} \right\|_2^2. \tag{52}$$

If $\mathbb{c}_i \neq \mathbb{c}_j$, we have the following equation from Lemma 4.7:

$$D_F[i,j] = \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right) + \left\| \mathbf{f}_{i,c-b} - \frac{1}{\sqrt{d[n+j]}} \mathbf{f}_{j,c-b} \right\|_2^2. \tag{53}$$

Since $\mathbb{c}_i \neq \mathbb{c}_j$ holds, $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in different connected components. Therefore, if $\langle \cdot \rangle$ represents the inner product of two vectors, $\langle \mathbf{f}_{i,c-b}, \mathbf{f}_{j,c-b} \rangle = 0$ from Equation (18). As a result,

$$\begin{aligned} &\left\| \mathbf{f}_{i,c-b} - \frac{1}{\sqrt{d[n+j]}} \mathbf{f}_{j,c-b} \right\|_2^2 = \|\mathbf{f}_{i,c-b}\|_2^2 + \frac{1}{d[n+j]} \|\mathbf{f}_{j,c-b}\|_2^2 - \frac{2}{\sqrt{d[n+j]}} \langle \mathbf{f}_{i,c-b}, \mathbf{f}_{j,c-b} \rangle \\ =& f_{i,c-b} + \frac{1}{d[n+j]} f_{j,c-b}. \end{aligned} \tag{54}$$

Therefore, if $\mathbb{c}_i \neq \mathbb{c}_j$, we have

$$D_F[i,j] = \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right) + f_{i,c-b} + \frac{1}{d[n+j]} f_{j,c-b}. \tag{55}$$

which completes the proof. □

Lemma 4.8 indicates that, if $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in the same connected component, since $\mathbf{f}_{i,c-b}$ is a vector of length $c - b$, we can compute distance $D_F[i,j]$ in $O(c-b)$ time by skipping the distance computations for the first b elements of $\mathbf{f}_i$. Furthermore, if $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in different connected components, we can compute distance $D_F[i,j]$ in $O(1)$ time from Lemma 4.8. As a result, if $b < c$, we can efficiently compute $D_F[i,j]$ from Lemma 4.8.

If $b \geq c$, since the eigenvectors of the largest eigenvalues of the block matrices are associated with vector $\mathbf{f}_i$, it is clear that $D_F[i,j]$ can be computed as follows from Lemma 4.8:

$$D_F[i,j] = \begin{cases} 0 & \text{if } \mathbb{c}_i = \mathbb{c}_j \\ \frac{1}{2}\left(\frac{1}{n_i} + \frac{1}{n_j}\right) & \text{otherwise} \end{cases}. \tag{56}$$

This equation indicates that we can compute $D_F[i,j]$ in $O(1)$ time if $b \geq c$. In addition, as shown in Equation (56), if $b \geq c$, we can compute $D_F[i,j]$ without using either $\mathbf{f}_i$ or $\mathbf{f}_{n+j}$. As a result, if $b \geq c$, we can directly compute $D_F[i,j]$ by skipping the computation of $\mathbf{F}$. Consequently, we can significantly improve the efficiency if we have $b \geq c$. In Section 4.4, we will detail the direct computation of $D_F[i,j]$ by skipping the computation of $\mathbf{F}$.

4.3.2 Lower Bounding Similarity. Even though we can efficiently compute distance $D_F[i,j]$ as described in the previous section, the high computation cost of $O(d)$ is incurred in computing $D_E[i,j]$ to obtain the nearest prototypes due to the high dimensionality of the data points. This section describes our approach that efficiently computes the nearest prototypes to each data point.

To improve the efficiency, we approximate each data point and prototype by SVD since it gives the smallest error in approximating high-dimensional data [36]. To approximate data point $\mathbf{x}_i = [x_i[1], \ldots, x_i[d]]$ and prototype $\mathbf{a}_j = [a_j[1], \ldots, a_j[d]]$ of d dimensions, we compute SVD of rank d' for sampled data points. Specifically, let $\mathbf{X}' = [\mathbf{x}'_1, \ldots, \mathbf{x}'_m]^\top$ be an $m \times d$ matrix of m randomly sampled data points, we compute SVD of rank d' on $\mathbf{X}'$ as $\mathbf{U}_{d'} \mathbf{\Sigma}_{d'} \mathbf{V}_{d'}^\top$, and we obtain

the approximations $\tilde{\mathbf{x}}_i = [\tilde{x}_i[1], \ldots, \tilde{x}_i[d']]$ and $\tilde{\mathbf{a}}_j = [\tilde{a}_j[1], \ldots, \tilde{a}_j[d']]$ with d' dimensions as $\tilde{\mathbf{x}}_i = \mathbf{x}_i\mathbf{V}_{d'}$ and $\tilde{\mathbf{a}}_j = \mathbf{a}_j\mathbf{V}_{d'}$, respectively. To efficiently compute the nearest prototypes, we introduce the following lower bounding distance:

Definition 4.9. Let $x'_i = \sqrt{\|\mathbf{x}_i\|_2^2 - \|\tilde{\mathbf{x}}_i\|_2^2}$ and $a'_j = \sqrt{\|\mathbf{a}_j\|_2^2 - \|\tilde{\mathbf{a}}_j\|_2^2}$, lower bounding distance $\tilde{D}[i, j]$ between $\mathbf{x}_i$ and $\mathbf{a}_i$ is given by

$$\tilde{D}[i, j] = \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{a}}_j\|_2^2 + (x'_i - a'_j)^2 + \beta D_F[i, j]. \tag{57}$$

We show the following lemma for the lower bounding distance:

Lemma 4.10. *$\tilde{D}[i, j]$ has the following property:*

$$\tilde{D}[i, j] \leq D[i, j]. \tag{58}$$

Proof. From Equation (32) and (33), we have

$$D[i, j] = \|\mathbf{x}_i - \mathbf{a}_j\|_2^2 + \beta D_F[i, j] = \textstyle\sum_{k=1}^{d}(x_i[k] - a_j[k])^2 + \beta D_F[i, j]. \tag{59}$$

Since SVD is an orthonormal transformation [36], we have

$$\begin{aligned}&\textstyle\sum_{k=1}^{d}(x_i[k] - a_j[k])^2 = \sum_{k=1}^{d'}(\tilde{x}_i[k] - \tilde{a}_j[k])^2 + \sum_{k=d'+1}^{d}(\tilde{x}_i[k] - \tilde{a}_j[k])^2 \\ =&\|\tilde{\mathbf{x}}_i - \tilde{\mathbf{a}}_j\|_2^2 + \textstyle\sum_{k=d'+1}^{d}(\tilde{x}_i[k] - \tilde{a}_j[k])^2.\end{aligned} \tag{60}$$

From the Cauchy-Schwarz inequality [39], we have

$$\begin{aligned}&\textstyle\sum_{k=d'+1}^{d}(\tilde{x}_i[k] - \tilde{a}_j[k])^2 = \sum_{k=d'+1}^{d}(\tilde{x}_i[k])^2 + \sum_{k=d'+1}^{d}(\tilde{a}_j[k])^2 - 2\sum_{k=d'+1}^{d}\tilde{x}_i[k]\tilde{a}_j[k] \\ \geq&\left(\sqrt{\textstyle\sum_{k=d'+1}^{d}(\tilde{x}_i[k])^2} - \sqrt{\textstyle\sum_{k=d'+1}^{d}(\tilde{a}_j[k])^2}\right)^2 = (x'_i - a'_j)^2.\end{aligned} \tag{61}$$

Therefore, we have

$$D[i, j] \geq \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{a}}_j\|_2^2 + (x'_i - a'_j)^2 + \beta D_F[i, j], \tag{62}$$

which completes the proof. □

Since $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{a}}_j$ are vectors of length d', computing lower bounding distance $\tilde{D}[i, j]$ is more efficient than computing distance $D[i, j]$. Algorithm 3 depicts our approach that efficiently computes $\tilde{\mathbf{S}}$. In Algorithm 3, $\mathbb{A}_i$ is the set of $l + 1$ nearest prototypes to data point $\mathbf{x}_i$. If $\beta \neq 0$, it computes distance $D_F[i, j]$ for each pair of a data point and a prototype (line 1-7). Specifically, if $b < c$, it computes $D_F[i, j]$ from Equation (50) (line 4-5); it uses Equation (32) to compute $D_F[i, j]$, otherwise (line 6-7). It then computes the nearest prototypes of each data point (line 8-17). Specifically, it initializes $\mathbb{A}_i$ by adding $l + 1$ dummy prototypes whose distances to $\mathbf{x}_i$ are ∞ (line 9-10). It picks up prototypes to compute the lower bounding distances (line 11-12). If a prototype can be the nearest prototype, it accurately computes the distance for the prototype (line 13-14), and then updates the nearest prototypes if necessary (line 15-17). Then, it computes each element of S from the obtained nearest prototypes (line 18-19). Finally, it computes $\tilde{\mathbf{S}}$ from Equation (6) (line 20). The computational cost of Algorithm 3 is given as follows:

Lemma 4.11. *Algorithm 3 takes $O(n(md' + ld))$ time.*

Proof. In the case of $b < c$, if $\mathbf{x}_i$ and $\mathbf{a}_j$ are present in the same connected component, it requires $O(c - b)$ time to compute $D_F[i, j]$ from Equation (50). In addition, if $\mathbf{x}_i$ and $\mathbf{a}_j$ are included in different connected components, it takes $O(1)$ time to compute $D_F[i, j]$ from Equation (50). Since $\frac{m}{c}$ prototypes would be present in the same connected component as $\mathbf{x}_i$, it requires $O(nm)$ time to compute $D_F[i, j]$ for the pairs of the data points and the prototypes. On the other hand, in the case of $b \geq c$, it needs $O(1)$ time to compute $D_F[i, j]$ from Equation (56). Therefore, it takes $O(nm)$ time

Algorithm 3 Similarity Matrix Computation

Input: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, prototypes $\mathbf{a}_1, \ldots, \mathbf{a}_m$, approximate data points $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$, approximate prototypes $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_m$, number of clusters c, number of neighbor prototypes l, number of block matrix b, matrix $\mathbf{F}$, regularization parameter β
Output: matrix $\tilde{\mathbf{S}}$

if $\beta \neq 0$ **then**
  for $i = 1$ to n **do**
    for $j = 1$ to m **do**
      if $b < c$ **then**
        compute $D_F[i, j]$ from Equation (50);
      else
        compute $D_F[i, j]$ from Equation (56);
for $i = 1$ to n **do**
  $\mathbb{A}_i = \emptyset$;
  add $l + 1$ dummy prototypes to $\mathbb{A}_i$;
  for $j = 1$ to m **do**
    compute $\tilde{D}[i, j]$ from Equation (57);
    if $\tilde{D}[i, j] \leq \max_{\mathbb{A}_i} \{D[i, k]\}$ **then**
      compute $D[i, j]$ from Equation (32);
      if $D[i, j] \leq \max_{\mathbb{A}_i} \{D[i, k]\}$ **then**
        add $\mathbf{a}_j$ to $\mathbb{A}_i$;
        subtract $\mathbf{a}_k = \text{argmax}_{\mathbb{A}_i} \{D[i, k]\}$ from $\mathbb{A}_i$;
for $i = 1$ to n **do**
  compute the i-th row of $\mathbf{S}$ from $\mathbb{A}_i$ and Equation (10);
compute $\tilde{\mathbf{S}}$ from Equation (6);

Algorithm 4 F-KMM

Input: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, prototypes $\mathbf{a}_1, \ldots, \mathbf{a}_m$, number of clusters c, number of prototypes m, number of nearest prototypes l
Output: c clusters

1: set $\mathbf{X}'$ by randomly sampling m data points;
2: compute rank-d' SVD on $\mathbf{X}'$;
3: **for** $i = 1$ to n **do**
4: $\tilde{\mathbf{x}}_i = \mathbf{x}_i \mathbf{V}_{d'}$;
5: $\mathbf{A} = \mathbf{X}'$;
6: **for** $i = 1$ to m **do**
7: $\tilde{\mathbf{a}}_i = \mathbf{a}_i \mathbf{V}_{d'}$;
8: compute $\tilde{\mathbf{S}}$ by Algorithm 3 by setting $\beta = 0$;
9: $\beta = \alpha$;
10: **repeat**
11: compute $\tilde{\mathbf{S}}$ by Algorithm 3;
12: **repeat**
13: compute $\mathbf{M}'_1, \ldots, \mathbf{M}'_b$ by Algorithm 1;
14: **if** $b \neq c$ **then**
15: **if** $b < c$ **then**
16: $\beta = 2\beta$;
17: compute $\mathbf{q}_1, \ldots, \mathbf{q}_c$ by Algorithm 2;
18: compute $\mathbf{F}$ from Equation (18);
19: **else**
20: $\beta = \beta/2$;
21: compute $\tilde{\mathbf{S}}$ by Algorithm 3;
22: **until** $b = c$
23: **for** $i = 1$ to m **do**
24: update $\mathbf{a}_i$ from Equation (14);
25: $\tilde{\mathbf{a}}_i = \mathbf{a}_i \mathbf{V}_{d'}$;
26: **until** assignments of prototypes converge
27: obtain clusters from c connected components;

to compute $D_F[i, j]$ for the pairs of the data points and prototypes. In addition, it needs $O(nmd')$ time to compute the lower bounding distances of Equation (57) and $O(nld)$ time to compute the distances using Equation (32) accurately. Since the elements of $\mathbf{S}$ are obtained from the distances of the nearest prototypes, it needs $O(nl)$ time to compute $\mathbf{S}$ from Equation (10). Since the number of non-zero elements in $\mathbf{S}$ is nl, it takes $O(nl)$ time to compute $\tilde{\mathbf{S}}$ from $\mathbf{S}$ using Equation (6). As a result, $O(n(md' + ld))$ time is needed to compute each element of matrix $\tilde{\mathbf{S}}$. □

4.4 Clustering Algorithm

Algorithm 4 gives a complete description of our approach. It first computes SVD for the sampled data points (line 1-2). It then computes approximate data points (line 3-4). Following the original approach, it sets the sampled data points as the prototypes and computes approximate prototypes (line 5-7). Following the original approach, it initializes the regularization parameter as $\beta = \alpha$ by computing $\tilde{\mathbf{S}}$ (line 8-9). It then iterates the computations (line 10-26). Specifically, it computes $\tilde{\mathbf{S}}$ by Algorithm 3 (line 11) and the block matrices by Algorithm 1 (line 13). If $b < c$, it doubles β as in the original approach and computes $\mathbf{F}$ after computing eigenvalues and eigenvectors by Algorithm 2 (line 15-18). If $b > c$, it reduces β by half, following the original approach (line 19-20). Note that it does not compute $\mathbf{F}$ if $b > c$. It then computes $\tilde{\mathbf{S}}$ by Algorithm 3 (line 21). If $b = c$, it terminates the iterations and updates the prototypes and their approximations (line 22-25). It iterates these procedures until the assignments of the prototypes no longer change (line 26). After the iterations terminate, it computes the c clusters from the connected components of the bipartite graph (line 27). As shown in Algorithm 4, if $b > c$, we skip the computations of $\mathbf{F}$ to improve the efficiency of the proposed approach as described in Section 4.3.1.

4.4.1 Theoretical Analysis. The proposed approach has the following properties:

Theorem 4.12. *If t_f is the number of iterations needed to update* **F**, *Algorithm 4 takes $O(ndd' + md^2 + ((\frac{m}{b})^2 t_p + cn)t_f + n(md' + ld + ml)t_c + (nd + mdd')t_a)$ time.*

Proof. It needs $O(md^2)$ time to compute SVD for the m sampled data points. It takes $(n+m)dd'$ time to compute the approximate data points and prototypes. Since it needs $O((\frac{m}{b})^2 t_p + \frac{mc}{b})$ time to compute the eigenvalues and eigenvectors from Lemma 4.6 and it requires $O(cn)$ time to obtain **F**, it needs $O((\frac{m}{b})^2 t_p + cn)t_f$ time to update **F** if $b < c$. Otherwise, it takes $O(t_c - t_f)$ time to update β. Since it needs $O(n(md' + ld))$ time to compute $\tilde{\mathbf{S}}$ from Lemma 4.11 and $O(nml)$ time to compute the block matrices from Lemma 4.2, it needs $O(n(md' + ld + ml)t_c)$ times to compute $\tilde{\mathbf{S}}$ and the block matrices in iterated calculations. In addition, it requires $O(nd + mdd')t_a$ time to compute the prototypes and their approximations. Consequently, the computation cost of Algorithm 4 is $O(ndd' + md^2 + ((\frac{m}{b})^2 t_p + cn)t_f + n(md' + ld + ml)t_c + (nd + mdd')t_a)$. □

Theorem 4.13. *Algorithm 4 takes $O(n(m + d) + dd')$ space.*

Proof. Relative to the original approach, the proposed approach additionally needs to hold matrix **M**, **Q**, **P**, Σ, and $\mathbf{V}_{d'}$. Moreover, it additionally holds the upper bounds of eigenvalues, approximate data points, and approximate prototypes. Matrix **M**, **Q**, and **P** have the size of $m \times m$. It needs spaces of $O(m)$ and $O(dd')$ to hold Σ and $\mathbf{V}_{d'}$, respectively. It takes spaces of $O(m)$, $O(nd')$, and $O(md')$ to hold the upper bounds of eigenvalues, approximate data points, and approximate prototypes, respectively. Therefore, the additional memory cost is $O(nm + (n + d)d')$. On the other hand, the original approach needs $O(n(m + d))$ space, as described in Section 3. As a result, the memory cost of Algorithm 4 is $O(n(m + d) + dd')$. □

Theorem 4.14. *Algorithm 4 yields the same clustering results as the original approach.*

Proof. It uses Algorithm 2 to compute the c eigenvalues and eigenvectors from the block matrices. For each block matrix $\mathbf{M}'_i$, it can exactly compute the largest eigenvalues and their eigenvectors since we have $\lambda_{i,j} = 1$ and $\mathbf{q}_{i,j} = \mathbf{b}'_i$ from Lemma 4.3. Moreover, if the c largest eigenvalues of the block matrices are not obtained in the process of Algorithm 2, we must have $\overline{\lambda}_{i,j} \geq \min_{\mathbb{B}}\{\lambda_{i',j'}\}$ due to the upper bounding property of Lemma 4.5. Therefore, Algorithm 2 cannot prune the computations of the c largest eigenvalue and their eigenvectors. As a result, since Algorithm 2 can exactly compute the c largest eigenvalues and their eigenvectors of **M** by using Equation (18), we can exactly compute **F**.

In addition, we use Algorithm 3 to obtain matrix **S** by computing the nearest prototypes of each data point. As shown in Lemma 4.10, $\tilde{D}[i, j]$ has the lower bounding property such that $\tilde{D}[i, j] \leq D[i, j]$. Therefore, we must have $\tilde{D}[i, j] \leq \max_{\mathbb{A}_i}\{D[i, k]\}$ for the nearest prototypes in Algorithm 3. As a result, Algorithm 3 cannot prune the nearest prototypes in computing **S**. Therefore, it can exactly compute **S**. Since we can exactly compute **F** and **S**, Algorithm 4 guarantees the same clustering result as K-Multiple-Means. □

Theorem 4.12 and 4.13 indicate that, for large-scale data, our approach need $O(nm(\log d + l)t_c)$ time and $O(n(m + d))$ space. This is because, (1) $m \ll n$, $d' < d \ll n$, and $l \ll n$ holds for large-scale data, (2) we have $t_f < t_c$ and $t_a < t_c$, (3) we set $m = \sqrt{nc}$ and $d' = \log d$ as we describe in the next section. On the other hand, for large-scale data, the original approach takes $O(nm^2 t_c)$ time and $O(n(m + d))$ space since $d \ll m$, $c \ll m$ and $t_a < t_c$. As a result, for large-scale data, our approach has a smaller computation cost than the original approach while making memory overhead negligible.

Theorem 4.12 and 4.14 theoretically show that the proposed approach offers better efficiency than K-Multiple-Means while guaranteeing the equivalence of clustering results.

4.4.2 Extension. This section shows the extensions of our approach.

Clustering Accuracy. As described in the previous section, we can yield the same clustering results as the original approach. This section proposes an approach to improving clustering accuracy. As shown in Algorithm 4, we randomly initialize the prototypes. To improve clustering accuracy, we initialize the prototypes using k-means++ [1]. Since k-means++ selects data points as prototypes based on the distances to the closest prototypes, it can effectively spread prototypes to match the data distribution and improve clustering accuracy. However, k-means++ incurs the high computation cost of $O(nmd)$ to identify the closest prototype of each data point. In the proposed approach, we use SVD to reduce this cost.

Algorithm 5 shows the approach used to initializing the prototypes. In this algorithm, $D_c[i]$ is the distance of data point $\mathbf{x}_i$ to its closest prototype. Algorithm 5 first samples a data point as a prototype and computes its approximation (line 1-2). It initializes $D_c[i]$ for each data point (line 3-4). It then iteratively determines prototypes using $D_c[i]$, the same as k-means++ (line 6-7). It uses SVD to update $D_c[i]$ for each data point efficiently (line 9-14). Specifically, since we have $\|\tilde{\mathbf{x}}_j - \tilde{\mathbf{a}}_{i+1}\| + (x'_j - a'_{i+1})^2 \leq \|\mathbf{x}_j - \mathbf{a}_{i+1}\|_2^2$ for $\mathbf{x}_j$ and $\mathbf{a}_{i+1}$ as shown in the proof of Lemma 4.10, it updates $D_c[i]$ only if $\|\tilde{\mathbf{x}}_j - \tilde{\mathbf{a}}_{i+1}\| + (x'_j - a'_{i+1})^2 \leq D_c[j]$ holds (line 11). Note that we can use Algorithm 5 instead of line 5-7 of Algorithm 4. The computation and memory costs of Algorithm 5 are given as follows:

Lemma 4.15. *Algorithm 5 requires $O(nd \log m + nmd' + mdd')$ time and $O(n)$ space to initialize the prototypes.*

Proof. It takes $O(mdd')$ and $O(nmd')$ time to compute approximate prototypes and approximate distances, respectively. It requires $O(nd' \log m)$ time to compute the distances accurately even if we randomly sample prototypes [6]. It needs $O(nm)$ time to sample data points by using $D_c[i]$. In addition, it needs $O(n)$ space to hold $D_c[i]$ for each data point. As a result, the computational and memory costs of Algorithm 5 are $O(nd \log m + nmd' + mdd')$ and $O(n)$, respectively. □

Number of Prototypes. As shown in Algorithm 4, number of prototypes m is a hyper-parameter of our approach. Since each cluster is obtained by grouping prototypes in our approach, the number of prototypes impacts clustering accuracy. A simple approach for determining m is to use cross-validation that iteratively computes the clusters by changing the number of prototypes. However, since the number of prototypes ranges from $c+1$ to n (i.e., $c+1 \leq m \leq n$), it needs a significantly high computational cost to perform the cross-validation for large-scale data. In addition, if a dataset does not have labels, we cannot use the cross-validation. In this section, we describe the approach to determining the number of prototypes according to the data distribution.

We modify Algorithm 5 to determine m according to the data distribution, as shown in Algorithm 6. Specifically, if $\mathbb{X}_{\mathbf{a}_i}$ is a set of data points whose closest prototype is $\mathbf{a}_i$, it iteratively adds prototypes until we have $|\mathbb{X}_{\mathbf{a}_m}| = 1$; the added prototype is the closest prototype of a single data point (line 3-6). This is because, if we have $|\mathbb{X}_{\mathbf{a}_m}| = 1$, data points are well represented by other prototypes; there is no need to add prototypes. As a result, we can determine m according to the data distribution. Note that, since each prototype is sampled from the data points in Algorithm 5, each prototype is the prototype closest to at least a single data point (i.e., $|\mathbb{X}_{\mathbf{a}_i}| \geq 1$). Algorithm 6 can be used instead of Algorithm 5 to determine the prototypes. Since Algorithm 6 uses Algorithm 5, its computational and memory costs are the same as Algorithm 5.

Outlier Detection. Our approach can capture non-spherical clusters since it has the same clustering results as the original approach. In obtaining non-spherical clusters, DBSCAN is a popular density-based approach that detects not only clusters but also outliers [10]. An outlier is a data point that

Algorithm 5 Prototype Initialization

Input: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate data points $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$, number of prototypes m, matrix $\mathbf{V}_{d'}$
Output: prototypes $\mathbf{a}_1, \ldots, \mathbf{a}_m$, approximate prototypes $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_m$
1: set $\mathbf{a}_1$ by randomly sampling a data point;
2: $\tilde{\mathbf{a}}_1 = \mathbf{a}_1\mathbf{V}_{d'}$;
3: **for** $i = 1$ to n **do**
4: $D_c[i] = \|\mathbf{x}_i - \mathbf{a}_1\|_2^2$;
5: **for** $i = 1$ to $m - 1$ **do**
6: sample data point $\mathbf{x}_j$ with probability $\frac{D_c[j]}{\sum_{1 \le k \le n} D_c[k]}$;
7: $\mathbf{a}_{i+1} = \mathbf{x}_j$;
8: $\tilde{\mathbf{a}}_{i+1} = \mathbf{a}_{i+1}\mathbf{V}_{d'}$;
9: **for** $j = 1$ to n **do**
10: compute $\|\tilde{\mathbf{x}}_j - \tilde{\mathbf{a}}_{i+1}\| + (x'_j - a'_{i+1})^2$;
11: **if** $\|\tilde{\mathbf{x}}_j - \tilde{\mathbf{a}}_{i+1}\| + (x'_j - a'_{i+1})^2 \le D_c[j]$ **then**
12: compute $\|\mathbf{x}_j - \mathbf{a}_{i+1}\|_2^2$;
13: **if** $\|\mathbf{x}_j - \mathbf{a}_{i+1}\|_2^2 \le D_c[j]$ **then**
14: $D_c[j] = \|\mathbf{x}_j - \mathbf{a}_{i+1}\|_2^2$;

Algorithm 6 Adaptive Prototype Initialization

Input: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate data points $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$, matrix $\mathbf{V}_{d'}$, number of clusters c
Output: prototypes $\mathbf{a}_1, \ldots, \mathbf{a}_m$, approximate prototypes $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_m$
1: compute $c + 1$ prototypes by Algorithm 5;
2: $m = c + 1$;
3: **repeat**
4: compute $\mathbf{a}_{m+1}$ by line 6-14 of Algorithm 5;
5: $m = m + 1$;
6: **until** $|\mathbb{X}_{\mathbf{a}_m}| = 1$

Algorithm 7 Outlier Detection

Input: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, approximate data points $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$, matrix $\mathbf{V}_{d'}$, number of clusters c, number of neighbor prototypes l
Output: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$, prototypes $\mathbf{a}_1, \ldots, \mathbf{a}_m$, approximate data points $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$, approximate prototypes $\tilde{\mathbf{a}}_1, \ldots, \tilde{\mathbf{a}}_m$, set of outliers $\mathbb{O}$
1: compute prototypes by Algorithm 6;
2: compute nearest prototype set $\mathbb{N}_i$ of each data point by line 8-17 of Algorithm 3;
3: compute average distance θ;
4: $\mathbb{X} = \emptyset$;
5: $\mathbb{O} = \emptyset$;
6: **for** $i = 1$ to n **do**
7: **for** each nearest prototype $\mathbf{a}_j \in \mathbb{N}_i$ **do**
8: **if** $D_E[i, j] > \theta$ **then**
9: subtract $\mathbf{a}_j$ from $\mathbb{N}_i$;
10: **if** $|\mathbb{N}_i| < l/2$ **then**
11: add $\mathbf{x}_i$ to $\mathbb{O}$;
12: **else**
13: add $\mathbf{x}_i$ to $\mathbb{X}$;
14: $n = n - |\mathbb{O}|$;
15: $i = 1$;
16: **for** each data point $\mathbf{x}_j \in \mathbb{X}$ **do**
17: $\mathbf{x}_i = \mathbf{x}_j$;
18: $\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}}_j$;
19: $i = i + 1$;

lies alone in a low-density region [4], and outlier detection is used in many applications, such as flight anomaly detection [37], wind power prediction [52], and weather anomaly detection [47]. In this section, we extend the proposed approach to find outliers as well as clusters.

Since an outlier is in a low-density region [10], it is distant from its nearest prototypes. Therefore, if we compute distances between the data points and prototypes, outliers have larger distances to their nearest prototypes than non-outliers. We detect outliers based on this observation, as shown in Algorithm 7. In this algorithm, θ is the average distance between data points and their nearest prototypes, $\mathbb{O}$ is a set of outliers, and $\mathbb{X}$ is a set of data points to compute clusters. Algorithm 7 first computes the prototypes by Algorithm 6 (line 1). It then obtains the nearest prototypes of each data point by Algorithm 3 and computes the average distance between data points and their nearest prototypes (line 2-3). It detects outliers by using the distances of each data point to their nearest prototypes. Specifically, if a data point is distant from its nearest prototypes, it adds the data point to $\mathbb{O}$ as an outlier (line 10-11). Otherwise, it adds the data point to $\mathbb{X}$ (line 12-13). It sets the IDs of data points by using $\mathbb{X}$ to compute clusters (line 16-19). Note that we can use Algorithm 7 instead of line 5-7 of Algorithm 4. The computation and memory costs of Algorithm 7 are as follows:

Lemma 4.16. *Algorithm 7 takes $O(nd \log m + nmd' + nld + mdd')$ time and $O(nl)$ space to detect outliers.*

Proof. It takes $O(nd \log m + nmd' + mdd')$ time to obtain prototypes by using Algorithm 5. It requires $O(n(md' + ld))$ time to obtain the nearest prototype of each data point by Algorithm 3. It needs $O(nl)$ time to compute $\mathbb{O}$ and $\mathbb{X}$. Moreover, it requires $O(n)$ space to hold $\mathbb{O}$ and $\mathbb{X}$. It takes

Table 2. Characteristics of the experimental data.

Dataset (Abbreviation)	#Data points	#Dimensions	#Clusters
BinAlpha (BN)	1,854	256	10
Abalone (AB)	4,177	8	28
Fashion-MNIST (FS)	70,000	784	10
FARS (FR)	100,968	29	8
Spoken-Arabic-Digit (SP)	263,256	14	10
Kuzushiji-49 (KZ)	270,912	784	49
Covertype (CV)	581,012	54	7
Poker (PK)	1,025,010	10	10

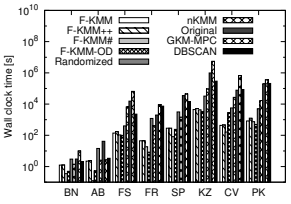


Fig. 3. Clustering time of each approach.

$O(nl)$ space to hold $\mathbb{N}_i$ for each data point. Therefore, Algorithm 7 needs $O(nd \log m + nmd' + nld + mdd')$ time and $O(nl)$ space. □

5 EXPERIMENTAL EVALUATION

We performed experiments to confirm the effectiveness of our approaches. In this section, "F-KMM" represents the results of our clustering approach of Algorithm 4, and "F-KMM++" represents the results of our approach with prototype initialization by Algorithm 5. "F-KMM#"' represents the results of our clustering approach that determines the number of prototypes by Algorithm 6. "F-KMM-OD" represents the results of our approach that finds outliers as well as clusters by Algorithm 7. "Original", "Randomized", "GKM-MPC", "nKMM", and "DBSCAN" represent the results of the original approach [30], the randomized SVD-based approach [28], GKM-MPC [44], nKMM [51], and DBSCAN [10], respectively. GKM-MPC and nKMM are recent proposals for multi-prototype clustering, as described in Section 2. For the randomized SVD-based approach, we used randomized SVD to compute $\mathbf{F}$ by approximately computing the leading singular vectors [28]. DBSCAN is a popular density-based clustering approach [10].

In the experiments, we used eight real-world data sets of various domains. We summarize the main statistics of these datasets in Table 2. BinAlpha (BN) and Abalone (AB) are the datasets used in the original paper of K-Multiple-Means [30], and other datasets were downloaded from the website of OpenML[1]. By following the original paper of K-Multiple-Means [30], the number of clusters, c, was set to be the ground truth, the number of nearest prototypes was set to $l = 5$, and the number of prototypes was set to $m = \sqrt{nc}$ in the experiment[2]. For the proposed approaches, we set the target rank of SVD to $d' = \log d$ and the tolerance of the power method to 0.00001. For GKM-MPC, we set the number of nearest neighbors to five to construct the nearest neighbor graph of prototypes, as shown in [44]. Similarly, we set the number of nearest neighbors to five for nKMM to construct the nearest neighbor graph of data points, the same as in [51]. For DBSCAN, we set the minimum number of neighborhood data points to $2d$ and set the radius of a neighborhood by sorting distances of a nearest neighbor graph in accordance with [10, 34]; we used the median of the nearest distances as the neighborhood radius. All the approaches were implemented in C++. We conducted all experiments on a Linux server with Intel Xeon Platinum 8280 CPU with 2.70GHz processors. The server had 1.5TB of memory and a 1TB hard disk.

5.1 Clustering Time

We assessed the clustering time of each approach in Fig. 3. Table 3 shows a breakdown of the processing time of our approaches, as well as the original and randomized SVD-based approaches,

[1] https://www.openml.org/

[2] In real applications, the number of prototypes, m, is set to $m = \sqrt{nc}$ as is recommended in the papers on multi-prototype clustering [30, 51]. The number of clusters, c, is set differently to match the application. In image segmentation, c is a user-specified number of image segments [20]. In load balancing of vehicular networks, c is set as the number of shards wherein each base station communicates [17]. In blind source separation, c is set to the number of estimated sources [25].

Table 3. Breakdown of processing time [s].

Processing step	F-KMM	F-KMM++	F-KMM#	F-KMM-OD	Randomized	Original
Compute SVD	0.0140	0.0140	0.0140	0.0140	–	–
Initialize Prototype	0.0003	1.1231	0.1699	0.1699	0.0003	0.0003
Detect outlier	–	–	–	0.8932	–	–
Compute $\tilde{\mathbf{S}}$	46.033	45.293	15.960	6.4464	91.506	91.801
Compute $\mathbf{F}$	5.5019	5.5541	2.3729	0.8001	1115.9	1906.2
Compute $\mathbf{A}$	0.0462	0.0454	0.0260	0.0166	0.0477	0.0496

Table 4. Number of prototypes.

Approach	BN	AB	FS	FR	SP	KZ	CV	PK
F-KMM	136	341	836	898	1,622	3,643	2,016	3,201
F-KMM++	136	341	836	898	1,622	3,643	2,016	3,201
F-KMM#	78	76	221	243	670	1,851	910	1,444
F-KMM-OD	78	76	221	243	670	1,851	910	1,444

Table 5. Number of outliers.

Approach	BN	AB	FS	FR	SP	KZ	CV	PK
DBSCAN	461	2,040	17,814	50,637	89,727	54,252	289,273	122,300
F-KMM-OD	687	2,598	23,278	50,672	86,010	67,504	235,913	304,615

for the FARS dataset. Note that we attained almost the same results for other datasets. Table 4 shows the number of prototypes used in our approaches, and Table 5 shows the number of outliers detected by F-KMM-OD and DBSCAN.

As shown in Fig. 3, our approaches offer higher efficiency than the previous approaches. Specifically, our approaches are up to 144.8, 166.5, 475.6, 4151.8, and 888.2 times faster than the randomized SVD-based approach, nKMM, the original approach, GKM-MPC, and DBSCAN, respectively. As shown in Table 3, the original approach incurs a high computation cost to iteratively compute SVD on $n \times m$ matrix $\tilde{\mathbf{S}}$ to obtain $\mathbf{F}$. Since the computation cost of SVD is $O(nm^2)$, the original approach is slower than our approaches. Although the randomized SVD-based approach approximately computes SVD to reduce the computation cost as shown in Table 3, it still needs $O(nm(c + d + \log m))$ time to compute the nearest prototypes of each data point to obtain $\tilde{\mathbf{S}}$. Therefore, it is slower than our approaches. As described in Section 2, in the merge stage of nKMM, it needs to construct the nearest neighbor graph of the data points. Since the computation cost of the nearest neighbor graph is quadratic to the number of data points, nKMM is much slower than our approaches. Moreover, since the split stage of GKM-MPC needs $O(n^2md)$ time to initialize the sub-clusters, it incurs significantly higher computation times than our approaches. Moreover, since DBSCAN iteratively performs range searches from the data points, its computation cost is high.

On the other hand, to efficiently compute $\mathbf{F}$, we compute the eigenvectors from the block matrices of $m \times m$ matrix $\mathbf{M}'$ as described in Section 4.2. In addition, as described in Section 4.3, we compute $\tilde{\mathbf{S}}$ by skipping unnecessary distance computations and estimating lower bounding distances between the data points and the prototypes. Note that, although we have the additional computation cost for SVD, it is relatively short, as shown in Table 3. Besides, F-KMM++ can efficiently initialize the prototypes, as shown in Table 3. In addition, F-KMM# can more efficiently initialize the prototypes than F-KMM++, as shown in Table 3. This is because F-KMM# uses fewer prototypes than F-KMM++, as shown in Table 4. Since F-KMM# uses fewer prototypes, it can more efficiently compute matrix $\tilde{\mathbf{S}}$, $\mathbf{F}$, and $\mathbf{A}$ than F-KMM++, as shown in Table 3. As a result, F-KMM# is more efficient than F-KMM and F-KMM++, as shown in Fig. 3. Furthermore, F-KMM-OD needs less computation time for matrix $\tilde{\mathbf{S}}$, $\mathbf{F}$, and $\mathbf{A}$ than F-KMM# although it needs additional processing time to detect outliers, as shown in Table 3. This is because it can reduce the number of data points needed to compute clusters by effectively detecting outliers the same as DBSCAN, as shown in Table 5. Note that F-KMM# and

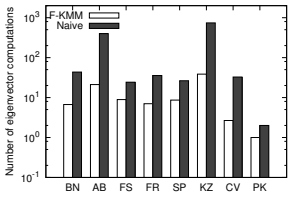


Fig. 4. Number of eigenvector computations.

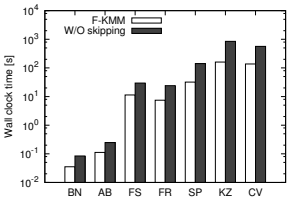


Fig. 5. Effectiveness of skipping distance computations.

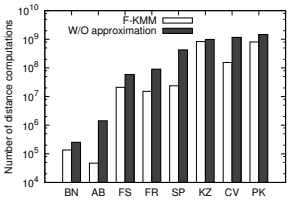


Fig. 6. Effectiveness of the approximation approach.

F-KMM-OD use the same number of prototypes to compute clusters, as shown in Table 4, since they use the same algorithm (Algorithm 6) to compute the prototypes.

5.2 Eigenvector Computation

As shown in Section 4.2.2, we compute the largest eigenvalues and their eigenvectors of the block matrices from the elements of diagonal matrix $\mathbf{D}$ to reduce the number of eigenvector computations. In addition, we use the upper bounds of eigenvalues to reduce the number of the power method computations. To show the effectiveness of these approaches, we evaluated the number of eigenvector computations by the power method needed to obtain the c eigenvectors. Note that the proposed approach uses the power method to compute the eigenvectors only if $b < c$ holds, as shown in Section 4.4. In this experiment, we compared the proposed approach to the naive approach that computes the c largest eigenvalues and their eigenvectors for each block matrix. Fig. 4 shows the average of the numbers of eigenvector computations in the iterations that use the power method where $b < c$. In this figure, "Naive" indicates the result of the naive approach.

Fig. 4 shows that our approach reduces the number of eigenvector computations by up to 94.8% from the naive approach. Since the naive approach computes the c eigenvectors for b block matrices, it must apply the power method bc times, although only c eigenvectors are needed for computing the clusters. On the other hand, our approach can obtain the b eigenvectors from the elements of matrix $\mathbf{D}$ from Lemma 4.3 without using the power method. Even if we need to compute the $c - b$ eigenvectors by the power method, our proposal prunes unnecessary computations by exploiting the upper bounding property of Lemma 4.5. Since our approach can effectively reduce the number of eigenvector computations, it can efficiently obtain the c largest eigenvalues and their eigenvectors.

5.3 Skipping Distance Computations

As mentioned in Section 4.3.1, if $b > c$, we directly compute distance $D_F[i, j]$ by skipping the computations of $\mathbf{F}$ to compute $\tilde{\mathbf{S}}$ efficiently. To show the effectiveness of this approach, Fig. 5 plots the computation time taken to obtain $\tilde{\mathbf{S}}$ for the case of $b > c$. In this figure, "W/O skipping" represents the results of the approach that does not skip the computations of $\mathbf{F}$; this approach computes $\mathbf{F}$ even if $b > c$ holds. We omit the experimental results for the Poker dataset, since $b > c$ did not occur in any iteration.

Fig. 5 indicates that our approach can effectively reduce the computation time to obtain $\tilde{\mathbf{S}}$. Specifically, it reduced the computation time by up to 81.1% against the comparative approach. Since the comparative approach does not use the skipping approach, it must compute $\mathbf{F}$ at $O(c(n+m))$ time. In addition, this approach needs $O(nmc)$ time to compute $D_F[i, j]$ for the pairs of the data points and the prototypes. However, if $b > c$, we do not need to compute $\mathbf{F}$. Therefore, we can compute $D_F[i, j]$ at $O(nm)$ time for the pairs of the data points and the prototypes from Equation (56). Since we can skip the computations of $\mathbf{F}$, our approach can more efficiently compute $\tilde{\mathbf{S}}$ than the comparative approach.

Table 6. Clustering performance of each approach [%].

Approach	NMI								Purity							
	BN	AB	FS	FR	SP	KZ	CV	PK	BN	AB	FS	FR	SP	KZ	CV	PK
F-KMM#	78.96	16.28	60.56	9.803	0.036	47.55	5.814	0.260	89.61	37.28	80.32	65.39	96.25	74.76	91.15	99.98
F-KMM++	76.91	17.17	59.94	9.468	0.027	50.70	4.747	0.308	73.50	26.10	48.92	45.05	10.17	36.20	50.34	50.20
F-KMM	73.62	16.87	56.70	9.468	0.020	48.80	3.264	0.027	70.98	25.54	42.30	45.05	10.15	35.36	49.84	50.13
Original	73.62	16.87	56.70	9.468	0.020	48.80	3.264	0.027	70.98	25.54	42.30	45.05	10.15	35.36	49.84	50.13
Randomized	36.87	14.73	1.085	1.464	0.018	1.909	2.086	0.021	37.40	25.16	10.10	41.94	10.07	2.705	48.79	50.12
nKMM	81.46	4.772	48.08	4.182	0.030	28.68	4.483	0.020	79.31	18.29	21.67	42.24	10.09	14.53	49.04	50.12
GKM-MPC	51.07	9.804	44.98	1.114	0.019	40.12	17.04	0.013	60.29	19.94	48.81	41.71	10.46	37.45	54.50	50.56
F-KMM-OD	53.61	41.06	37.30	25.77	23.33	28.60	21.80	35.81	62.13	13.14	60.63	35.18	65.51	71.71	56.54	67.64
DBSCAN	33.01	40.26	24.08	24.40	24.07	25.28	27.42	10.68	59.28	27.87	74.55	40.15	64.69	79.97	15.03	88.01

5.4 Lower Bounding Similarity

As described in Section 4.3.2, we approximate the data points and the prototypes to compute the lower bounding distances. We can efficiently compute the nearest prototypes using the lower bounding distances since we can avoid computing the distances accurately. In this experiment, we evaluated the number of accurate distance computations to show the effectiveness of this approach. Fig. 6 shows the average of the numbers of the accurate distance computations to find the nearest prototypes in the iterations. In this figure, "W/O approximation" is the result of the approach that computes the accurate distances for all pairs.

As shown in Fig. 6, our approach reduced the number of accurate distance computations by up to 96.6% against the comparative approach. Since the comparative approach computes accurate distances for all the pairs, it needs $O(nmd)$ time to compute the nearest prototypes of the data points. On the other hand, our approach takes $O(nmd')$ time to compute the lower bound distances for all the pairs. However, since $d' \ll d$, it can efficiently compute the lower bound distances for the pairs. In addition, our approach can prune the accurate distance computations for unlikely pairs; unlikely to be the nearest prototype. As shown in the figure, Kuzushiji-49 needs many distance computations compared to the other datasets. This is because Kuzushiji-49 has more clusters than the other datasets, as shown in Table 2. Since SVD represents the clusters using small numbers of singular vectors, if data points have many clusters, the approximation error of SVD would increase; it could be difficult to prune distance computations effectively. However, if data points have a small number of clusters, we can effectively prune distance computations using the lower bound distances, as shown in Fig. 6. As a result, the proposed approach can efficiently find the nearest prototypes for each data point.

5.5 Clustering Performance

Since the original approach of K-Multiple-Means has the computational bottleneck of the iterative SVD computations, computation time can be reduced by using randomized SVD [28]. However, this approach sacrifices clustering performance to improve efficiency since it approximately computes SVD. One major advantage of the proposed approach is that it is guaranteed to yield the same clustering results as the original approach, as described in Section 4.4. In addition, as described in Section 4.4.2, the proposed approach can improve clustering performance by initializing the prototypes more effectively. Furthermore, we extended our approach to determine the number of prototypes according to the data distribution in Section 4.4.2. Moreover, we extended our approach to detect outliers the same as DBSCAN, as described in Section 4.4.2. Therefore, we compared our approaches to DBSCAN. In this experiment, we evaluated NMI (Normalized Mutual Information) and the purity of each approach. They are popular metrics of clustering performance [27]. NMI is a normalization of the Mutual Information score of the clustering result, and purity is the fraction of the majority class of the cluster relative to the size of the cluster. NMI and purity range from 0

to 1, and they have larger values as the clustering result better matches the ground truth. Table 6 shows NMI and purity of each approach. Note that we treated each outlier detected by DBSCAN and F-KMM-OD as a single cluster in this experiment.

As shown in Table 6, the randomized SVD-based approach yields lower clustering performance than the proposed approaches. Since the randomized SVD-based approach approximately computes SVD, matrix $\mathbf{F}$ obtained by the approach is different from the original approach; it is not the optimal solution for the problem of (5). As a result, since the distances between data points and prototypes are computed from $\mathbf{F}$, the randomized SVD-based approach erroneously computes the nearest prototypes for each data point. Consequently, the randomized SVD-based approach and the original approach yield different clustering results. On the other hand, we can accurately compute the leading c singular vectors from the eigenvectors of the block matrices. Therefore, we can exactly compute matrix $\mathbf{F}$. In addition, we can accurately compute the nearest prototypes by using the lower bounding distances between the data points and the prototypes. Therefore, we can exactly compute matrix $\mathbf{S}$. Since matrix $\mathbf{F}$ and $\mathbf{S}$ obtained by our approach are the same as those by the original approach, we can, unlike the randomized SVD-based approach, obtain the same clustering results as the original approach. In addition, the clustering performance of the proposed approach is competitive with recent multi-prototype clustering approaches since our approach can obtain the same clustering result as the original approach.

Furthermore, as shown in Table 6, F-KMM++ is more accurate than F-KMM. This indicates that we can improve clustering performance by effectively initializing the prototypes. Moreover, F-KMM# can yield higher clustering accuracy than F-KMM++, as shown in Table 6. This is because it effectively determines the prototypes according to the data distribution. The results in Table 6, as well as Fig. 3 indicate that we can increase the efficiency and clustering performance of K-Multiple-Means by determining the number of prototypes according to the data distribution. Besides, as shown in Table 6, DBSCAN yields relatively different clustering accuracy from multi-prototype clustering approaches. This is because we treated each outlier as a single cluster. As shown in Table 6, F-KMM-OD is competitive with DBSCAN in terms of clustering accuracy. This is because F-KMM-OD can effectively detect outliers similar to DBSCAN, as shown in Table 5. Note that F-KMM-OD is significantly faster than DBSCAN, as shown in Fig. 3.

6 CONCLUSIONS

K-Multiple-Means is a simple but effective extension of K-means that represents each cluster via multiple prototypes. However, it does not efficiently handle large-scale data since it needs to iteratively compute SVD on the similarity matrix of the bipartite graph. In this paper, we addressed the problem of reducing the processing time of K-Multiple-Means. Our approach efficiently computes the singular vectors from the eigenvectors of the block matrices obtained from the similarity matrix between prototypes. Moreover, the proposed approach skips unnecessary distance computations and estimates the lower bounding distances between the data points and the prototypes to improve efficiency. Experiments showed that the proposed approach is significantly faster than the previous approaches while still matching the clustering performance of K-Multiple-Means. In future work, we will investigate how to improve clustering accuracy of the proposed approach. Although this paper proposed to initialize prototypes using k-means++, which has been studied in discrete algorithms, we plan to use the approaches studied for computational geometry [3, 15]. By improving clustering accuracy as well as efficiency, we can provide an attractive alternative to the research community for the extraction of clusters to gain insights into large-scale data.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 22H03596.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The Advantages of Careful Seeding. In *SODA*. 1027–1035.
[2] Shenglan Ben, Zhong Jin, and Jingyu Yang. 2011. Guided Fuzzy Clustering with Multi-prototypes. In *IJCNN*. 2430–2436.
[3] Tuhin Kr. Biswas, Kinsuk Giri, and Samir Roy. 2023. ECKM: An Improved K-means Clustering Based on Computational Geometry. *Expert Syst. Appl.* 212 (2023), 118862.
[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *SIGMOD*. 93–104.
[5] Fan R. K. Chung. 1996. *Spectral Graph Theory*. American Mathematical Society.
[6] Thomas H. Cormen, Charles E. Leiserson Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*. The MIT Press.
[7] James W. Demmel. 2017. *Applied Numerical Linear Algebra*. Orient Blackswan.
[8] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2004. Kernel K-means: Spectral Clustering and Normalized Cuts. In *KDD*. ACM, 551–556.
[9] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. 2007. Weighted Graph Cuts without Eigenvectors A Multilevel Approach. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 11 (2007), 1944–1957.
[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *KDD*. 226–231.
[11] Ky Fan. 1949. On a Theorem of Weyl Concerning Eigenvalues of Linear Transformations I. *Proceedings of the National Academy of Sciences* 35, 11 (1949), 652–655.
[12] Yasuhiro Fujiwara, Yasutoshi Ida, Junya Arai, Mai Nishimura, and Sotetsu Iwamura. 2016. Fast Algorithm for the Lasso based L1-Graph Construction. *Proc. VLDB Endow.* 10, 3 (2016), 229–240.
[13] Yasuhiro Fujiwara, Go Irie, Shari Kuroyama, and Makoto Onizuka. 2014. Scaling Manifold Ranking Based Image Retrieval. *Proc. VLDB Endow.* 8, 4 (2014), 341–352.
[14] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Yasutoshi Ida, and Machiko Toyoda. 2015. Adaptive Message Update for Fast Affinity Propagation. In *KDD*. ACM, 309–318.
[15] Kinsuk Giri and Tuhin Biswas. 2022. *Applications of Computational Geometry in Clustering: A Review*. 209–214.
[16] Mark A. Girolami. 2002. Mercer Kernel-based Clustering in Feature Space. *IEEE Trans. Neural Networks* 13, 3 (2002), 780–784.
[17] Pengwenlong Gu, Dingjie Zhong, Cunqing Hua, Farid Naït-Abdesselam, Ahmed Serhrouchni, and Rida Khatoun. 2021. Scaling A Blockchain System For 5G-based Vehicular Networks Using Heuristic Sharding. In *GLOBECOM*. IEEE, 1–6.
[18] David A. Harville. 2008. *Matrix Algebra From a Statistician's Perspective*. Springer.
[19] G. Karypis, Eui-Hong Han, and V. Kumar. 1999. Chameleon: Hierarchical Clustering Using Dynamic Modeling. *Computer* 32, 8 (1999), 68–75.
[20] Xuelong Li, Yunxing Zhang, and Rui Zhang. 2023. Self-Weighted Unsupervised LDA. *IEEE Trans. Neural Networks Learn. Syst.* 34, 3 (2023), 1627–1632.
[21] Jiye Liang, Liang Bai, Chuangyin Dang, and Fuyuan Cao. 2012. The K -Means-Type Algorithms Versus Imbalanced Data Distributions. *IEEE Trans. Fuzzy Syst.* 20, 4 (2012), 728–745.
[22] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. 2003. The Global K-means Clustering Algorithm. *Pattern Recognit.* 36, 2 (2003), 451–461.
[23] Manhua Liu, Xudong Jiang, and Alex C. Kot. 2009. A Multi-prototype Clustering Algorithm. *Pattern Recognit.* 42, 5 (2009), 689–698.
[24] Ting Luo, Caiming Zhong, Hong Li, and Xia Sun. 2010. A Multi-prototype Clustering Algorithm Based on Minimum Spanning Tree. In *FSKD*. 1602–1607.
[25] Baoze Ma and Tianqi Zhang. 2021. Underdetermined Blind Source Separation Based on Source Number Estimation and Improved Sparse Component Analysis. *Circuits Syst. Signal Process.* 40, 7 (2021), 3417–3436.
[26] J. B. MacQueen. 1967. Some Methods for Classification and Analysis of MultiVariate Observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1. University of California Press, 281–297.
[27] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schuetze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.
[28] Per-Gunnar Martinsson and Joel A. Tropp. 2020. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numer.* 29 (2020), 403–572.
[29] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. 2001. On Spectral Clustering: Analysis and an Algorithm. In *NIPS*. 849–856.
[30] Feiping Nie, Cheng-Long Wang, and Xuelong Li. 2019. K-Multiple-Means: A Multiple-Means Clustering Method with Specified K Clusters. In *SIGKDD*. 959–967.
[31] Feiping Nie, Xiaoqian Wang, Cheng Deng, and Heng Huang. 2017. Learning A Structured Optimal Bipartite Graph for Co-Clustering. In *NIPS*. 4129–4138.

[32] Feiping Nie, Xiaoqian Wang, and Heng Huang. 2014. Clustering and Projected Clustering with Adaptive Neighbors. In *KDD*. 977–986.
[33] Feiping Nie, Xiaoqian Wang, Michael I. Jordan, and Heng Huang. 2016. The Constrained Laplacian Rank Algorithm for Graph-Based Clustering. In *AAAI*. 1969–1976.
[34] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. 1998. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Min. Knowl. Discov.* 2, 2 (1998), 169–194.
[35] Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. 1998. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Comput.* 10, 5 (1998), 1299–1319.
[36] Dennis Shasha and Yunyue Zhu. 2004. *High Performance Discovery In Time Series: Techniques And Case Studies*. SpringerVerlag.
[37] Kevin Sheridan, Tejas G. Puranik, Eugene Mangortey, Olivia J. Pinon-Fischer, Michelle Kirby, and Dimitri N. Mavris. 2020. *An Application of DBSCAN Clustering for Flight Anomaly Detection During the Approach Phase*.
[38] Jianbo Shi and Jitendra Malik. 2000. Normalized Cuts and Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (2000), 888–905.
[39] J. Michael Steele. 2004. *The Cauchy-Schwarz Master Class*. Cambridge University Press.
[40] Chin-Wang Tao. 2002. Unsupervised Fuzzy Clustering with Multi-center Clusters. *Fuzzy Sets Syst.* 128, 3 (2002), 305–322.
[41] Ulrike von Luxburg. 2007. A Tutorial on Spectral Clustering. *Stat. Comput.* 17, 4 (2007), 395–416.
[42] Chang-Dong Wang, Jian-Huang Lai, Ching Y. Suen, and Jun-Yong Zhu. 2013. Multi-Exemplar Affinity Propagation. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 9 (2013), 2223–2237.
[43] Chang-Dong Wang, Jian-Huang Lai, and Jun-Yong Zhu. 2010. A Conscience On-line Learning Approach for Kernel-Based Clustering. In *ICDM*. 531–540.
[44] Lu Wang, Huidong Wang, and Chuanzheng Bai. 2021. A New Multi-Prototype Based Clustering Algorithm. In *ICIST*. 598–603.
[45] Yangtao Wang and Lihui Chen. 2016. K-MEAP: Multiple Exemplars Affinity Propagation With Specified K Clusters. *IEEE Trans. Neural Networks Learn. Syst.* 27, 12 (2016), 2670–2682.
[46] David S. Watkins. 2010. *Fundamentals of Matrix Computations*. Wiley.
[47] S Wibisono, M Anwar, Aji Supriyanto, and I Amin. 2021. Multivariate Weather Anomaly Detection Using DBSCAN Clustering Algorithm. *Journal of Physics: Conference Series* 1869 (2021), 012077.
[48] Jinglin Xu, Junwei Han, Kai Xiong, and Feiping Nie. 2016. Robust and Sparse Fuzzy K-Means Clustering. In *IJCAI*. 2224–2230.
[49] Lihi Zelnik-Manor and Pietro Perona. 2004. Self-Tuning Spectral Clustering. In *NIPS*. 1601–1608.
[50] Hongyuan Zha, Xiaofeng He, Chris H. Q. Ding, Ming Gu, and Horst D. Simon. 2001. Spectral Relaxation for K-means Clustering. In *NIPS*. 1057–1064.
[51] Jingyuan Zhang. 2022. A New K-Multiple-Means Clustering Method. In *KSEM*. 621–632.
[52] Yiyi Zhang, Wang, Xujun Liang, Li, and Duan. 2020. Short-Term Wind Power Prediction Using GA-BP Neural Network Based on DBSCAN Algorithm Outlier Identification. *Processes* 8 (2020), 157.

Received July 2023; revised October 2023; accepted November 2023