Algorithm 1038: KCC: A MATLAB Package for *k*-Means-based Consensus Clustering

HAO LIN, School of Economics and Management, Beihang University, China
HONGFU LIU, Michtom School of Computer Science, Brandeis University, USA
JUNJIE WU and HONG LI, School of Economics and Management, Beihang University, China
STEPHAN GÜNNEMANN, Department of Informatics, Technical University of Munich, Germany

Consensus clustering is gaining increasing attention for its high quality and robustness. In particular, *k*-means-based Consensus Clustering (KCC) converts the usual computationally expensive problem to a classic *k*-means clustering with generalized utility functions, bringing potentials for large-scale data clustering on different types of data. Despite KCC's applicability and generalizability, implementing this method such as representing the binary dataset in the *k*-means heuristic is challenging and has seldom been discussed in prior work. To fill this gap, we present a MATLAB package, KCC, that completely implements the KCC framework and utilizes a sparse representation technique to achieve a low space complexity. Compared to alternative consensus clustering packages, the KCC package is of high flexibility, efficiency, and effectiveness. Extensive numerical experiments are also included to show its usability on real-world datasets.

CCS Concepts: • **Information systems → Clustering**; • **Computing methodologies → Ensemble methods**; • **Mathematics of computing → Statistical software;**

Additional Key Words and Phrases: Consensus clustering, *k*-means, utility functions, MATLAB

ACM Reference format:
Hao Lin, Hongfu Liu, Junjie Wu, Hong Li, and Stephan Günnemann. 2023. Algorithm 1038: KCC: A MATLAB Package for *k*-Means-based Consensus Clustering. *ACM Trans. Math. Softw.* 49, 4, Article 40 (December 2023), 27 pages.
https://doi.org/10.1145/3616011

H. Lin, J. Wu, and H. Li also with Key Laboratory of Data Intelligence and Management (Beihang University), Ministry of Industry and Information Technology, China.
Dr. Junjie Wu's work was supported by the National Natural Science Foundation of China (72031001, 72242101, 72021001). Dr. Hao Lin's work was partially supported by Sino-German (CSC-DAAD) Postdoc Scholarship Program under CSC (China Scholarship Council) Grant 201906020375 and DAAD (Deutscher Akademischer Austauschdienst) Grant 57531629.
Authors' addresses: H. Lin, J. Wu (corresponding author), and H. Li, School of Economics and Management, Beihang University, 37 Xueyuan Rd, Haidian District, Beijing, 100191, China; emails: {haolin, wujj, hong_lee}@buaa.edu.cn; H. Liu, Michtom School of Computer Science, Brandeis University, 415 South St, Waltham, MA, 02453, USA; email: hongfuliu@brandeis.edu; S. Günnemann, Department of Informatics, Technical University of Munich, Boltzmannstr. 3, Garching, 85748, Germany; email: guennemann@in.tum.de.

0098-3500/2023/12-ART40 $15.00
https://doi.org/10.1145/3616011

1 INTRODUCTION

Cluster analysis aims at separating instances into various clusters so that instances within a cluster are more similar to each other than to those in other clusters and thus serves as a fundamental technique in the fields of data mining, machine learning, and artificial intelligence. There are numerous successful clustering applications in diverse domains, including information retrieval [25], recommender systems [48], and image segmentation [49]. Researchers have proposed different clustering algorithms under varied assumptions, e.g., subspace [43], density [14], distribution [39], centroid [37], and connectivity [16]. It is difficult for users to choose between them in practice. Consensus clustering, a.k.a. ensemble clustering, therefore emerges, with the goal of fusing several partitions given by single clustering algorithms into an integrated one. This intuitive idea, however, faces the challenge of high computational complexity, because it is essentially a combinatorial optimization problem [50]. Nevertheless, consensus clustering has been broadly observed to achieve robust clustering performance with the ability to handle noise, outliers, and sample variations, due to its nature inherited from ensemble learning [41].

Extensive computational methods have been investigated to solve consensus clustering. Most previous studies could be categorized into two classes. The first class achieves a final consensus partition with an explicit objective of maximizing a defined utility function, which quantifies the similarity between multiple existing **basic partitions (BPs)** and the consensus partition. For example, Topchy et al. [53] designed a consensus clustering objective function based on Quadratic Mutual Information and utilized k-means clustering to solve it. Their work was further extended to an **Expectation-Maximization (EM)** consensus clustering algorithm [54]. This approach utilized the information from basic partition results as the feature vectors of data points, assumed the feature vectors follow a finite mixture model, and solved consensus clustering by maximum likelihood estimation with the EM algorithm. Wu et al. [58] offered a unified framework to convert ensemble clustering to k-means clustering with easily derivable **k-means-based Consensus Clustering (KCC)** utility functions. Additionally, other interesting consensus clustering objective functions have been investigated in References [28, 29, 36, 55].

The second class first constructs a co-association matrix by calculating the co-occurrence of each instance pair with the same assignment in the BPs. Then, different graph partitioning algorithms could be employed on the obtained co-association matrix for finding a consensus partition. Typical algorithms include agglomerative hierarchical clustering [17] and graph-based algorithms [50]. Recently, Liu et al. [32] suggested to perform spectral clustering on a co-association matrix and solved consensus clustering with a weighted k-means algorithm. Other methods include Locally Adaptive Cluster–based methods [12], genetic algorithm-based methods [60], Relabeling and Voting [3], locally weighted co-association matrix-based methods [22], fast propagation of clusterwise similarities–based methods [23], noise evidence removing–based methods [61], and representative co-association matrix–based methods [31]. More recently, some graph representation learning techniques [52] have also been utilized for enhanced consensus clustering.

Related packages. In the past few years, researchers in both machine learning and statistical software communities have implemented a number of consensus clustering software packages, including ClusterEnsemble [50], CLUE [20, 21], LinkCLUE [24], SC^{2}ATmd [42], ConsensusCluster [47], OpenEnsembles [46], and DiceR [9]. Among them, ClusterEnsemble [50] is the seminal work that converts the basic partitions into a representation of a hypergraph and solves the consensus clustering problem over the hypergraph with three efficient heuristics, i.e., the **HyperGraph Partitioning Algorithm (HGPA)**, **Cluster-based Similarity Partitioning Algorithm (CSPA)**, and **Meta-CLustering Algorithm (MCLA)**. CLUE [21], proposed by Hornik [20], is an extensible framework of consensus clustering and is available on the CRAN package repository. As described in the original JSS paper [20], CLUE proposes data structures to represent collections

of partitions or collections of hierarchies and provides different built-in methods to minimize the dissimilarity between the consensus and basic partitions or hierarchies. LinkCLUE [24] is a variant based on co-association matrices, which aims to modify the co-association matrix for providing a better similarity matrix in later agglomerative hierarchical clustering. Three matrices, i.e., the SimRank-based Similarity matrix, Connected-triple-based Similarity matrix, and Approximate SimRank-based Similarity matrix, are designed to replace the co-association matrix. SC^2ATmd [42] employs a resampling scheme to obtain several sub-datasets, runs a clustering algorithm to get the incomplete basic partitions, and finally builds the co-association matrix based on these incomplete basic partitions. ConsensusCluster [47] first generates a collection of basic partitions with various clustering methods to bootstrap over samples and features and then takes an average-linkage hierarchical clustering algorithm to obtain the single best consensus partition. OpenEnsembles [46] uses open source Python projects to implement majority vote, mixture models, and two additional co-association matrix–based methods. DiceR [9] achieves the final consensus clustering based on the R statistical language via four algorithms, including the link-based cluster ensembles, K-modes, majority voting, and cluster-based similarity partitioning algorithm. Although extensive attempts have already been devoted to the implementation of consensus clustering, approaches adopted in the above existing software packages suffer from the following two critical drawbacks:

- Existing packages that implement utility function-based methods are either designed purposefully for one specific utility function or designed for different utility functions in a non-unified way. This may hinder their applicability in different real-world applications. For example, CLUE [21] uses different heuristics such as `SE`, `GV1`, and `DWH` to solve the optimization problem with objective functions derived by different utility functions. Although this may provide some flexibility, the performance of different heuristics may vary dramatically, making it difficult for users to select a good heuristic.
- Existing packages, especially the co-association matrix-based packages, have less consideration of computational efficiency, and their implemented algorithms are of high computational complexity. This would indeed hinder their applicability to handle large-scale real-world data. For example, many of them, e.g., LinkCLUE, SC^2ATmd, and ConsensusCluster, employ hierarchical clustering algorithms for the final consensus clustering, which have a non-linear (at least $O(n^2)$) [45] time complexity in the number of instances n. The time complexity of the CSPA employed in ClusterEnsemble and DiceR is also quadratic in n.

Compared to the built-in methods of existing packages, the KCC method [58] has the following crucial features. (a) *Interpretability* and *robustness*: the explicit utility functions and novel objective functions in KCC have been theoretically and empirically verified to produce interpretable and robust clustering results. Wu et al. [57] demonstrates that KCC achieves state-of-the-art performance regarding clustering quality on extensive real-world datasets compared to two competitors, i.e., graph partitioning-based algorithms, and hierarchical clustering-based algorithms. (b) *Flexibility* and *generalizability*: KCC obtains excellent flexibility by allowing the choice of different utility functions for clustering multiple types of data in a unified framework, which is extremely important for real-life applications in different domains. Several variants of consensus clustering algorithms, including the **Spectral Ensemble Clustering (SEC)** [32, 35], **Infinite Ensemble Clustering (IEC)** [33, 34], and Greedy optimization of k-means-based Consensus Clustering [30], are based on the framework of KCC. This demonstrates the generalization ability of KCC's framework. (c) *Efficiency*: by virtue of the k-means like iterative process, KCC is extremely fast compared with its excellent competitors, e.g., the method based on a co-association matrix [17], and shows potential for big data clustering. The KCC algorithm has a time complexity of $O(InrK)$ with I being

the number of iterations, r being the number of BPs, and K being the number of clusters. Since in practice I, r, K are often set to 20, 100, and a relatively small number compared to N, respectively, KCC's time complexity can be nearly linear to n [58]. The above merits of KCC, though being very impressive, are in sharp contrast to the fact that a standard open source KCC software package for use in academia and industry is not available elsewhere. This indeed motivates us to provide a high-quality KCC implementation.

To this end, this article presents a KCC package (publicly available on Gitee[1]) that implements the KCC method [58] in MATLAB for solving the consensus clustering problem. In this package, we realize the whole framework of KCC, including the generation of basic partitions via random sampling of clustering numbers or data features, the selection of different utility functions originating from various k-means distances [57], and the k-means like iterative process of knowledge fusion of BPs. Moreover, considering the longstanding difficulty of cluster validity, multiple external and internal evaluation measures that assess the clustering quality are also included in this package for the purpose of self-containment.

The KCC package's features in software design and implementation are mainly discussed in Section 3 and are summarized as follows:

- The execution of the package can be achieved without the need of compilation nor the need of loading any commercial toolbox. The users can install the package by just unpacking the software archive, which creates a directory with all data and code files in it.
- All functions follow strict naming conventions for ease of use, review, and extension. For example, names with the prefix **`distance_`** indicate distance functions, and names with the prefix **`BasicCluster_`** indicate functions for generating BPs.
- The functions for consensus clustering are designed to be foolproof. For example, different utility functions can be used by calling a single consensus function **`KCC`**, which has a parameter to indicate the choice of a utility function. The preprocessing and consensus functions are combined to form a single **`RunKCC`** function for reducing the efforts of calling each function separately. Practical strategies such as running a number of times repeatedly to obtain the best results are also integrated in the **`RunKCC`** function.
- The package utilizes subtle MATLAB data variables to decrease the storage space. For example, the sparse representation of the binary dataset, i.e., Ki, $sumKi$, and $binIDX$, can save much memory space in the k-means heuristic.
- The package is easy to use with rich illustrative examples, e.g., examples of using different utility functions on extensive datasets, and performing consensus clustering on incomplete basic partitions. The names of the scripts for illustrating these examples start with **`demo`**.
- The package can easily be extended to include new utility functions or new consensus clustering algorithms, e.g., SEC and IEC, under the framework of KCC.

The rest of this article is arranged as follows. Some essential consensus clustering concepts and the previously proposed KCC method are briefly reviewed in Section 2 to make this article self-contained. The most important part of this article, i.e., the implementation aspects in KCC, is described in Section 3. Section 4 presents extensive experimental results. Section 5 concludes the article.

2 CONSENSUS CLUSTERING AND k-MEANS-BASED CONSENSUS CLUSTERING

This section reviews some mathematical concepts of consensus clustering and the KCC method that is proposed in Reference [58].

[1] https://gitee.com/linhaobuaa/KCC

2.1 Mathematical Background of Consensus Clustering

2.1.1 Problem Definition. Assume that there are a set of n instances $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ that belong to K groups $\mathcal{C} = \{C_1, \ldots, C_k\}$, with the constraint that $C_k \bigcap C_{k'} = \emptyset$, $\forall k \neq k'$, and $\bigcup_{k=1}^{K} C_k = \mathcal{X}$. Based on $\mathcal{X}$, a collection of r BPs, i.e., $\Pi = \{\pi_1, \pi_2, \ldots, \pi_r\}$, are generated. For each BP π_i, $\mathcal{X}$ is partitioned into K_i crisp clusters, and each instance is assigned to a cluster label index that ranges in $[1, K_i]$. A consensus clustering algorithm aims at finding an optimal partition π to fuse the information from the BPs Π for better clustering quality. We call π a consensus parition. As mentioned, ensemble clustering methods could be grouped into two main sets, i.e., the approaches based on a utility function and approaches based on a co-association matrix. In the first category, the optimal π is assumed to generate the maximum utility value, which has the following formulation:

$$\max_{\pi} \sum_{i=1}^{r} w_i U(\pi, \pi_i), \tag{1}$$

where U denotes a utility function that calculates the similarity between a consensus partition and a basic partition, and $w_i \in [0, 1]$ denotes a weight assigned to the basic partition π_i with a constraint of $\sum_{i=1}^{r} w_i = 1$. Usually all partitions can be of equal weight, but if some partitions are more important than others, then they can be assigned with a larger weight [50]. Compared to the co-association matrix-based method, we are more interested in the utility function-based method, because utility values are interpretable and robust to noise. However, a tradeoff between the computational efficiency and clustering performance is often the difficulty for this line of methods. As such, it remains an open question on how to design a good utility function to achieve both excellent clustering performance and high computational efficiency.

2.1.2 Basic Partition Generation Strategy. Consensus clustering algorithms requires the BPs rather than the raw data as the input. Therefore, it is important to design BP generation strategies. In the following, we introduce three widely used BP generation strategies, which focus on exploiting the diversity of BPs,

- *Random Parameter Selection.* Predefined parameters are usually needed for most base clustering algorithms. For instance, both k-means and K-medoids require the cluster number. For generating diverse BPs, **Random Parameter Selection (RPS)** samples values randomly from a certain range of alternatives as the parameters of a base clustering.
- *Random Feature Selection.* For generating diverse BPs, **Random Feature Selection (RFS)** employs the same base clustering algorithm with fixed parameters but performs clustering on different subspaces of $\mathcal{X}$. This is achieved by sampling random partial features from the whole feature space.
- *Combination of Different Algorithms.* This strategy produces diverse BPs by employing different base clustering algorithms over the same dataset.

2.1.3 Utility Function for Consensus Clustering. Recall that a traditional clustering algorithm usually quantifies the similarity in an instance level, e.g., by calculating the distance between an instance and a centroid. Instead, for consensus clustering, the similarity is measured at the partition level by the utility function; more specifically, the similarity between the consensus partition and basic partition is calculated. A contingency table as shown in Table 1 is often needed for comparing the two partitions.

In Table 1, we have two partitions π and π_i, of which the first one is the consensus partition, and the other one is the ith basic partition derived by some algorithm. Let $n_{kj}^{(i)}$ be the number of instances that are simultaneously assigned to C_k in the consensus partition π and $C_j^{(i)}$ in the

Table 1. Contingency Matrix

		π_i				
		$C_1^{(i)}$	$C_2^{(i)}$	$\cdots$	$C_{K_i}^{(i)}$	$\sum$
π	C_1	$n_{11}^{(i)}$	$n_{12}^{(i)}$	$\cdots$	$n_{1K_i}^{(i)}$	$n_{1+}^{(i)}$
	C_2	$n_{21}^{(i)}$	$n_{22}^{(i)}$	$\cdots$	$n_{2K_i}^{(i)}$	$n_{2+}^{(i)}$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
	C_K	$n_{K1}^{(i)}$	$n_{K2}^{(i)}$	$\cdots$	$n_{KK_i}^{(i)}$	$n_{K+}^{(i)}$
	$\sum$	$n_{+1}^{(i)}$	$n_{+2}^{(i)}$	$\cdots$	$n_{+K_i}^{(i)}$	n

basic partition π_i; $n_{k+}^{(i)} = \sum_{j=1}^{K_i} n_{kj}^{(i)}$; $n_{+j}^{(i)} = \sum_{k=1}^{K} n_{kj}^{(i)}$, $1 \le k \le K, 1 \le j \le K_i$. The Category Utility Function [38] can then be derived from the elements of the contingency table. This utility function has the following form:

$$U_c(\pi, \pi_i) = \sum_{k=1}^{K} p_{k+}^{(i)} \sum_{j=1}^{K_i} \left(\frac{p_{kj}^{(i)}}{p_{k+}^{(i)}} \right)^2 - \sum_{j=1}^{K_i} \left(p_{+j}^{(i)} \right)^2, \tag{2}$$

where $p_{kj}^{(i)} = n_{kj}^{(i)}/n$ denotes one instance's joint probability of being simultaneously assigned to C_k in π and $C_j^{(i)}$ in π_i; $p_{k+}^{(i)} = n_{k+}^{(i)}/n$ is the portion of C_k in π, and $p_{+j}^{(i)} = n_{+j}^{(i)}/n$ is the portion of $C_j^{(i)}$ in π_i. Since the contingency table is a tool to compare two partitions, any utility function can theoretically be designed based on it.

2.2 k-means-based Consensus Clustering

In this subsection, we do not intend to go through all derivation details of the KCC method but only remark several key motivations and techniques of KCC that are closely related to its implementation described in Section 3; we refer to the article [58] for the full details of the KCC method.

2.2.1 General Ideas. The utility function-based consensus clustering methods face several computational challenges. First, traditional methods usually suffer from using inefficient metaheuristics, e.g., the genetic algorithm and simulated annealing, to solve the consensus clustering problem. An interesting perspective to tackle this issue emerges when Topchy et al. [53] indicates the goal of consensus clustering, i.e., the utility maximization, is proved to be equivalent to the square-error criterion minimization, and can be achieved by a classical k-means clustering process. As a simple partition-based clustering method, k-means aims at finding K crisp clusters. Mathematically, k-means clustering can be regarded as optimizing an objective function as follows:

$$\min \sum_{k=1}^{K} \sum_{x \in C_k} f(x, m_k), \tag{3}$$

where m_k denotes C_k's cluster centroid and f is a function for computing the distance between an instance and a centroid, also known as a *point-to-centroid distance function*. To be specific, Topchy et al. [53] demonstrates that if a Category Utility Function is used as the utility function in Equation (1), and a squared Euclidean distance function is adopted as f in Equation (3), then the maximization of the objective function in consensus clustering is equivalently transformed into the minimization of the intra-class variance criterion in k-means clustering. In other words, a

consensus clustering problem with the Category Utility Function could be converted to a k-means clustering problem with the squared Euclidean distance. As such, consensus clustering could be solved in a two-phase k-means heuristic, i.e., an iterative process of cluster assignment and centroid calculation. This process is of high efficiency with a time complexity of $O(InrK)$. KCC follows this intuition to efficiently solve the consensus clustering problem.

The second issue lies in that few existing methods are able to provide a unified framework to flexibly choose different utility functions while this ability is very important in real-world applications. Flexible choice of utility functions is important in practice, because a certain utility function may perform excellently on one dataset but perform poorly on another one. For example, another KCC method [53] restricts their utility function to be the form of the special Category Utility Function and has weak generalizability on different datasets. Meanwhile, solving the consensus clustering problem with a chosen utility function usually requires a corresponding heuristic. Since different heuristic has different computational complexity, we may not guarantee that the problem can always be solved in an efficient way. For example, the state-of-the-art package CLUE [21] provides different utility functions but uses different heuristic, e.g., `SE`, `GV1`, and `DWH`, to deal with the different utility function. As such, it is often hard for users to choose a good utility function and a satisfied heuristic simultaneously in practice.

In sharp contrast, KCC achieves to convert consensus clustering to k-means clustering for the efficiency concern and provide choices of different utility functions in a unified way for the flexibility concern, simultaneously. In the following, we introduce multiple key techniques in KCC to achieve the goal.

2.2.2 Binary Dataset for KCC. Recall that consensus clustering could be intuitively viewed as finding an ensemble partition π from a set of BPs Π, so that the instances within a cluster of π are more similar to each other, which is very alike to conventional clustering excepting that consensus clustering exploits information in Π [53]. Therefore, it is essential to firstly represent the objects based on Π. Specifically, based on the set of r basic partitions Π, KCC defines a binary dataset $\mathcal{X}^{(b)}$ as follows:

$$\mathcal{X}^{(b)} = \left\langle x_1^{(b)}, \ldots, x_l^{(b)}, \ldots, x_n^{(b)} \right\rangle, \tag{4}$$

$$x_l^{(b)} = \left\langle x_{l,1}^{(b)}, \ldots, x_{l,i}^{(b)}, \ldots, x_{l,r}^{(b)} \right\rangle, \tag{5}$$

$$x_{l,i}^{(b)} = \left\langle x_{l,i,1}^{(b)}, \ldots, x_{l,i,j}^{(b)}, \ldots, x_{l,i,K_i}^{(b)} \right\rangle, \tag{6}$$

$$x_{l,i,j}^{(b)} = \begin{cases} 1, & \text{if } L_{\pi_i}(x_l) = j \\ 0, & \text{otherwise} \end{cases}, \tag{7}$$

$$\text{s.t.} \sum_{j=1}^{K_i} x_{l,i,j}^{(b)} = 1, \ \forall\, l \in \{1, 2, \ldots, n\}, i \in \{1, 2, \ldots, r\}, \tag{8}$$

where "$\langle\ \rangle$" indicates a transversal vector, K_i is the number of clusters in π_i, and $L_{\pi_i}(x_l)$ is one of the K_i labels in $\{1, 2, \ldots, K_i\}$ that π_i maps x_l to.

We can see that $\mathcal{X}^{(b)}$ is a binary representation of objects in the space of BPs. This binary representation is one key technique to the KCC framework, since it helps to calculate centroids for k-means, and derive generalized utility functions for KCC, which are introduced in the following.

2.2.3 Centroids and Generalized Distance Functions. The basic idea of KCC is to assume that the consensus partition π with K clusters is obtained by employing the k-means algorithm over the above binary representation $\mathcal{X}^{(b)}$. Wu et al. [58] demonstrates that the kth cluster's centroid in

π, denoted as m_k, can be derived based on the binary representation $\mathcal{X}^{(b)}$ and contingency matrix in Table 1. The centroid m_k can be represented as follows:

$$m_k = \langle m_{k,1}, \ldots, m_{k,i}, \ldots, m_{k,r} \rangle, \tag{9}$$

$$m_{k,i} = \langle m_{k,i,1}, \ldots, m_{k,i,j}, \ldots, m_{k,i,K_i} \rangle, \tag{10}$$

$$m_{k,i,j} = \frac{\sum_{x_l \in C_k} x_{l,i,j}^{(b)}}{|C_k|} = \frac{p_{kj}^{(i)}}{p_{k+}^{(i)}}. \tag{11}$$

Equation (11) builds a connection between consensus clustering and k-means clustering: The centroids of k-means could be acquired from the contingency matrix, on which the consensus clustering's utility function could also be designed. In other words, the binary representation and contingency matrix may help to connect the distance function with the utility function.

As inspired by the above connection, we remark on KCC's intuition to generalize utility functions for consensus clustering as follows: If we can define a generalized point-to-centroid distance function for k-means, and somehow map it to a utility function via the binary representation and contingency matrix, then we can achieve the generalization of the utility function.

A family of generalized distance functions, i.e., Bregman divergence [7], has been proved to preserve the simplicity and scalability of k-means and thus fits the k-means optimization framework [4]. A Bregman divergence $f : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$ is defined as follows:

$$f(x, y) = \phi(x) - \phi(y) - (x - y)^\top \nabla\phi(y), \tag{12}$$

where $\phi : \mathbb{R}^d \mapsto \mathbb{R}$ is a differentiable strictly convex function. This generalized form is intuitive, since several widely used point-to-centroid distance functions could be easily derived based on it, e.g., the squared Euclidean distance can be generated by setting $\phi(x)$ to $||x||_2^2$. It is worth noting that we can further relax the strictness property of ϕ's convexity to form a convex yet not necessarily strictly convex ϕ [59], which leads to the same mathematical form of f as the Bregman divergence in Equation (12). Following this intuition, we build KCC based on the generalized point-to-centroid distance function of k-means, which is f with convex ϕ as shown in Equation (12).

2.2.4 KCC Utility Functions. Before going into more details, we first introduce generalized utility functions of the KCC framework, which are also called *KCC utility functions*. More formally, a KCC utility function can be defined in the following way.

Definition 2.1 (KCC Utility Function). A utility function U is called a KCC utility function if $\forall\ \Pi = \{\pi_1, \ldots, \pi_r\}$ and $K \geq 2$; there is a function f defined in Equation (12) with a differentiable convex function ϕ, so that the following equation holds:

$$\max_{\pi} \sum_{i=1}^{r} w_i U(\pi, \pi_i) \Leftrightarrow \min_{\pi} \sum_{k=1}^{K} \sum_{x_l \in C_k} f\left(x_l^{(b)}, m_k\right). \tag{13}$$

The above definition is nothing but to establish the property that the KCC utility function should be beneficial to convert consensus clustering into k-means clustering with a generalized distance function.

As outlined in Section 2.2.3, to obtain the generalized utility function, we need to further consider the way to map the generalized distance function f to a utility function U. Since the convex function ϕ determines the form of the distance function f, building the mapping is equivalent to building a connection between ϕ and U. To achieve this, Wu et al. [58] derives the following theorem based on Definition 2.1, the binary representation, and the contingency matrix.

Table 2. Sample KCC Utility Functions

	$\mu(m_{k,i})$	$U_\mu(\pi, \pi_i)$	$f(x_l^{(b)}, m_k)$
U_c	$\|m_{k,i}\|_2^2 - \|P^{(i)}\|_2^2$	$\sum_{k=1}^{K} p_{k+}^{(i)} \|P_k^{(i)}\|_2^2 - \|P^{(i)}\|_2^2$	$\sum_{i=1}^{r} w_i \|x_{l,i}^{(b)} - m_{k,i}\|_2^2$
U_H	$(-H(m_{k,i})) - (-H(P^{(i)}))$	$\sum_{k=1}^{K} p_{k+}^{(i)} (-H(P_k^{(i)})) - (-H(P^{(i)}))$	$\sum_{i=1}^{r} w_i D(x_{l,i}^{(b)} \| m_{k,i})$
$U_{\cos}$	$\|m_{k,i}\|_2 - \|P^{(i)}\|_2$	$\sum_{k=1}^{K} p_{k+}^{(i)} \|P_k^{(i)}\|_2 - \|P^{(i)}\|_2$	$\sum_{i=1}^{r} w_i (1 - \cos(x_{l,i}^{(b)}, m_{k,i}))$
U_{L_p}	$\|m_{k,i}\|_p - \|P^{(i)}\|_p$	$\sum_{k=1}^{K} p_{k+}^{(i)} \|P_k^{(i)}\|_p - \|P^{(i)}\|_p$	$\sum_{i=1}^{r} w_i (1 - \frac{\sum_{j=1}^{K_i} x_{l,ij}^{(b)} (m_{k,ij})^{p-1}}{\|m_{k,i}\|_p^{p-1}})$

Note: $\|x\|_p$ – L_p norm of x; H, Shannon entropy; D, KL-divergence; cos, cosine similarity.

Theorem 2.2. *U is a KCC utility function if and only if $\forall\ \Pi = \{\pi_1, \ldots, \pi_r\}$ and $K \geq 2$; there is a set of continuously differentiable convex functions denoted as $\mu = \{\mu_1, \ldots, \mu_r\}$ so that*

$$U(\pi, \pi_i) = \sum_{k=1}^{K} p_{k+}^{(i)} \mu_i \left(P_k^{(i)}\right), \forall\ i \in \{1, 2, \ldots, r\}, \tag{14}$$

where

$$P_k^{(i)} = \left\langle \frac{p_{k1}^{(i)}}{p_{k+}^{(i)}}, \ldots, \frac{p_{kj}^{(i)}}{p_{k+}^{(i)}}, \ldots, \frac{p_{kK_i}^{(i)}}{p_{k+}^{(i)}} \right\rangle. \tag{15}$$

The convex function ϕ for the corresponding k-means clustering is given by

$$\phi(m_k) = \sum_{i=1}^{r} w_i \nu_i(m_{k,i}), \forall\ k \in \{1, 2, \ldots, K\}, \tag{16}$$

where

$$\nu_i(x) = a\mu_i(x) + c_i, \forall\ i \in \{1, 2, \ldots, r\}, a \in \mathbb{R}_{++}, c_i \in \mathbb{R}. \tag{17}$$

The utility function U in Equation (14) is indeed a generalized utility function. Theorem 2.2 intuitively establishes the specific connection between the convex function ϕ and the utility function U: Both of them are dependent on the set of r convex functions, i.e., $\mu = \{\mu_1, \ldots, \mu_r\}$. In other words, if the μ is specified as a certain set of functions, then the utility function U and its corresponding k-means distance function f can be determined accordingly. As such, the theorem guarantees that KCC can always be solved by using any arbitrary utility function and the efficient k-means heuristic, as long as each μ_i, $\forall i \in \{1, \ldots, r\}$, follows the continuously differentiable convex property. Hereinafter, the KCC utility function constructed by the μ_i in Equation (14) is denoted as U_μ. In Table 2, we give examples of KCC utility functions stemming from different form of the convex function μ_i and their corresponding distance function f. Note that $P^{(i)} \doteq \langle p_{+1}^{(i)}, \ldots, p_{+j}^{(i)}, \ldots, p_{+K_i}^{(i)} \rangle$, $\forall i \in \{1, \ldots, r\}$. Except for the well-known Category Utility Function U_c [38], we are not aware of the other three utility functions mentioned in prior literature.

Moreover, Theorem 2.2 leads to the process of conducting KCC, of which the pseudocode is shown in Algorithm 1. That is, we should first design a set of convex functions $\mu = \{\mu_1, \ldots, \mu_r\}$ and then calculate the utility function and consensus function with Equations (14) and (1), respectively; after setting the values of a and c_i in Equation (17) with default settings [58], i.e., $a = 1$, $c_i = 0$, $\forall i$, we can obtain ϕ and f with Equations (16) and (12), respectively; and, finally, the consensus partition can be found with the two-phase k-means heuristic as proposed in Section 2.2.1.

Another issue with the utility function-based consensus clustering methods is how to make the utility function interpretable. Recall that the utility function is normally viewed as a similarity measure between a basic partition and a consensus partition. Directly computing utility functions may lead to positive or negative values of utilities. If we somehow force the utility function $U(\pi, \pi_i)$

ALGORITHM 1: KCC algorithm.

Input: basic partition set Π, BP weight set w, a set of convex functions $\mu = \{\mu_1, \ldots, \mu_r\}$, number of clusters K

Output: consensus partition π, optimal consensus-function value Γ

1: $\forall i \in \{1, \ldots, r\}$, let $v_i \equiv \mu_i$, get ϕ by Equation (16), and then f by Equation (12);
2: Construct the binary representation $\mathcal{X}^{(b)}$ from Π;
3: Call k-means to cluster $\mathcal{X}^{(b)}$ into K clusters and get π;
4: Compute Γ by Equations (1) and (14);
5: **return** π and Γ.

to be non-negative, then it may provide interpretability to some extent, i.e., $U(\pi, \pi_i)$ could be regarded as a *utility gain* of the consensus partition π obtained from the ith basic partition π_i. Following this intuition, Wu et al. [58] further introduces two forms, i.e., a standard form and normalized form, for each utility function to guarantee its non-negativity.

Standard Form. Assume that we have obtained a utility value $U_\mu(\pi, \pi_i)$ based on the set of convex functions $\mu = \{\mu_1, \ldots, \mu_r\}$ by Equation (14). We derive a *standard form*, i.e., U_{μ_s}, of the KCC utility function in the following way:

$$U_{\mu_s}(\pi, \pi_i) = U_\mu(\pi, \pi_i) - \mu(P^{(i)}), \tag{18}$$

where $P^{(i)} = \langle p_{+1}^{(i)}, \ldots, p_{+K_i}^{(i)} \rangle$, $\forall i \in \{1, \ldots, r\}$ and the standard form, i.e., $U_{\mu_s}(\pi, \pi_i)$, can be interpreted as the *utility gain* of the consensus partition π obtained from the ith basic partition π_i. Note that Table 2 presents the standard forms of multiple KCC utility functions.

Normalized Form. Based on the *standard form* U_{μ_s}, we further derive a *normalized form* U_{μ_n} in the following way:

$$U_{\mu_n}(\pi, \pi_i) = \frac{U_{\mu_s}(\pi, \pi_i)}{|\mu(P^{(i)})|} = \frac{U_\mu(\pi, \pi_i) - \mu(P^{(i)})}{|\mu(P^{(i)})|}. \tag{19}$$

The *normalized form* $U_{\mu_n}(\pi, \pi_i)$ can be interpreted as a *utility gain ratio*.

2.2.5 Handling Incomplete Basic Partitions. Recall that the basic partitions are implicitly assumed to be complete for all data instances. In other words, each of the basic partitions is generated by conducting clustering on the same complete dataset $\mathcal{X}$. However, it is common that there are potential data collection or transformation failure issues in real-world applications, and this results in a scenario that only subsets of $\mathcal{X}$ can be observed when conducting consensus clustering. How can the consensus clustering problem be solved if only subsets of $\mathcal{X}$ are offered? More formally, assume that the ith basic partition π_i is generated on a subset of $\mathcal{X}$, i.e., $\mathcal{X}_i \subseteq \mathcal{X}$ with the constraint of $\bigcup_{i=1}^{r} \mathcal{X}_i = \mathcal{X}$. We call π_i an **incomplete basic partition (IBP)**. The problem is therefore defined as to solve consensus clustering given the set of r IBPs.

We remark that the aforementioned KCC framework can solve this problem with only slight modifications on the centroid update and distance calculation of the k-means heuristic. More specifically, we still follow the intuitions of Definition 2.1 and Theorem 2.2 to derive the KCC utility function but adjust the distance calculation and centroid update for handling IBPs as follows:

$$f\left(x_l^{(b)}, m_k\right) = \sum_{i=1}^{r} \mathbb{I}(x_l \in \mathcal{X}_i) f'\left(x_{l,i}^{(b)}, m_{k,i}\right), \tag{20}$$

and $m_k = \langle m_{k,1}, \ldots, m_{k,i}, \ldots, m_{k,r} \rangle$ with

$$m_{k,i} = \frac{\sum_{x_l \in C_k} \mathbb{I}(x_l \in \mathcal{X}_i) x_{l,i}^{(b)}}{|C_k \cap \mathcal{X}_i|}, \tag{21}$$

where $\mathbb{I}(x_l \in \mathcal{X}_i) = 1$ if $x_l \in \mathcal{X}_i$ and 0 otherwise; f' is a k-means distance.

The indicator term $\mathbb{I}(x_l \in \mathcal{X}_i)$ in the above equations is very intuitive: For the lth object in the dataset, if its clustering label information is missed in the basic partition π_i, i.e., $\mathbb{I}(x_l \in \mathcal{X}_i) = 0$, then its binary representation $x_{l,i}^{(b)}$ would contribute neither to the centroid computation of the kth cluster in π_i nor to the point-to-centroid distance computation.

Remark. Under the framework of KCC, KCC utility functions should help to convert consensus clustering into k-means clustering with generalized distance functions. If a utility function follows the property of a KCC utility function, then the consensus clustering problem that aims at finding the maximum value of the utility function could be transformed into an optimization problem with a k-means heuristic. The k-means heuristic refers to a two-phase process for conducting k-means clustering, i.e., an iterative process of cluster assignment and centroid calculation. In the transformation, the binary dataset plays an important role. Therefore, in the following section, we will introduce one important aspect in implementing the KCC algorithm, i.e., the implementation of the binary dataset.

3 IMPLEMENTATION ASPECTS

Although Wu et al. [58] has already presented the whole KCC framework as summarized in Section 2, aspects in implementing the KCC algorithm have seldomly been addressed in the existing literature. In this section, we propose the efforts in systematically implementing the KCC algorithm, building a sparse representation of the binary dataset and using the sparse representation to efficiently calculate distance and centroid in the k-means heuristic.

3.1 Systematic Implementation of KCC

We have implemented the KCC algorithm systematically in a MATLAB package with functional programming. Figure 1 gives an overview of the package's structure, which illustrates its main functions, including the basic partition generation functions, consensus clustering preprocessing function, consensus functions, and clustering quality evaluation functions. The full details of the package's available functions can be found in Section 3 of the user manual that accompanies the software. The package was developed and tested with MATLAB R2022a.

The core function in the package is the consensus function **`KCC`**, which provides the functionality for the consensus clustering to produce consensus partition results, i.e., the cluster labels for all data objects. In practical applications, the usage of different utility functions in alternative packages such as CLUE may result in different optimization problems, which usually need to be solved with different heuristics, e.g., `SE`, `GV1`, and `DWH` in CLUE. In contrast, the KCC package addresses this issue by implementing a unified consensus function, i.e., **`KCC`**, to solve the consensus clustering. In other words, it allows for the flexible choices of different utility functions but optimizing them using the same k-means clustering heuristic. Specifically, according to the process of k-means, the function **`KCC`** is mainly built upon the centroid initialization functions, point-to-centroid distance functions, and cluster centroid update functions. The function **`KCC`** achieves the flexiblity by defining a user input parameter `U` that indicates the choice of a utility function and calling different point-to-centroid distance functions according to the choice. For example, if a user chooses the utility function U_c, then a Euclidean distance function called **`distance_euc`** is applied for computation; if a user chooses the utility function U_{cos}, then a cosine distance function called **`distance_cos`** is applied for computation.

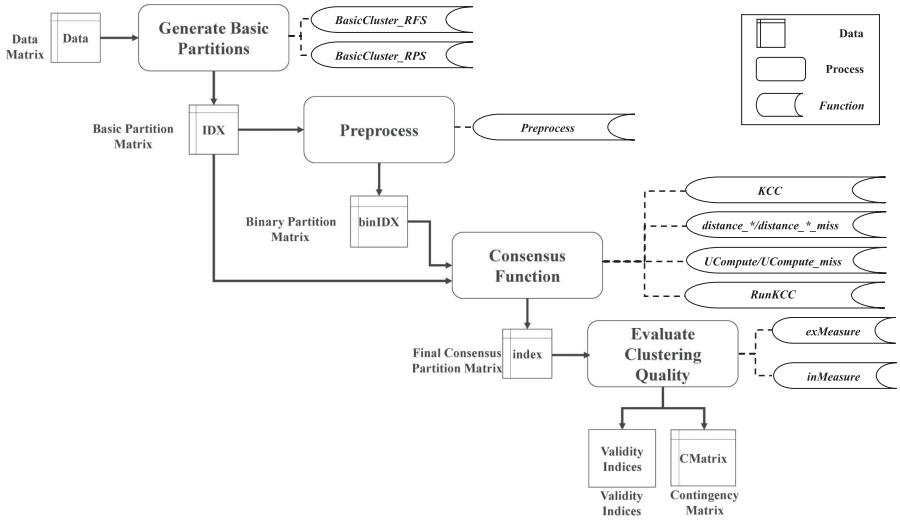


Fig. 1. Workflow of the KCC package.

Moreover, from the algorithm input's perspective, the input dataset may contain IBPs. As indicated in Section 2.2.5, the KCC algorithm may use slightly different manners in handling complete and incomplete BPs. Hence, we need to consider two scenarios, i.e., datasets with complete and incomplete BPs. Meanwhile, from the algorithm output's perspective, the value of the utility function as in Equation (14), i.e., the consensus clustering's utility value, may be of interest to a user if the user regards it as a metric in evaluating the clustering results. A notable aspect is that not outputing it does not influence the monitoring of convergence in the iteration of KCC algorithm (see Algorithm 1) but can accelerate the KCC's computation. As such, we should also consider two scenarios, i.e., whether or not the algorithm outputs the utility value. In the real implementation, we divide the KCC's usage in the following four application scenarios based on whether there exist IBPs in the input data matrix and whether the algorithm outputs the utility value: (1) the output of utility value is enabled, and there exist IBPs in the input; (2) the output of utility value is enabled, and there does not exist IBPs in the input; (3) the output of utility value is disabled, and there exist IBPs in the input; and (4) the output of utility value is disabled, and there does not exist IBPs in the input. For different application scenarios, the function **`KCC`** uses different combinations of the centroid initialization functions, point-to-centroid distance functions, and cluster centroid update functions. For example, the centroid initialization functions, i.e., **`sCentroid`** and **`sCentroid_miss`**, are designed to initialize the cluster centroids with complete BPs and with IBPs, respectively. The distance functions such as **`distance_euc`** and **`distance_euc_miss`** are designed to calculate point-to-centroid distance with complete BPs and with IBPs, respectively. The cluster centroid update functions, i.e., **`gCentroid`** and **`gCentroid_miss`**, are designed to update cluster centroids with complete BPs and with IBPs, respectively. Thanks to the structure of the **`KCC`** function, implementations of new KCC utility functions with complete/incomplete basic partitions can be easily incorporated into the package by adding new functions like **`distance_X`** and **`distance_X_miss`**, respectively.

As indicated in the overview of the KCC package in Figure 1, there are multiple supporting functions for the function **`KCC`**. The functions **`BasicCluster_RFS`** and **`BasicCluster_RPS`** provide the functionality for generating the basic partitions. They implement the RFS strategy and RPS strategy, respectively. More basic partition generation functions can also be incorporated into the package by adding a new function like **`BasicCluster_X`**. The function **`Preprocess`** provides the

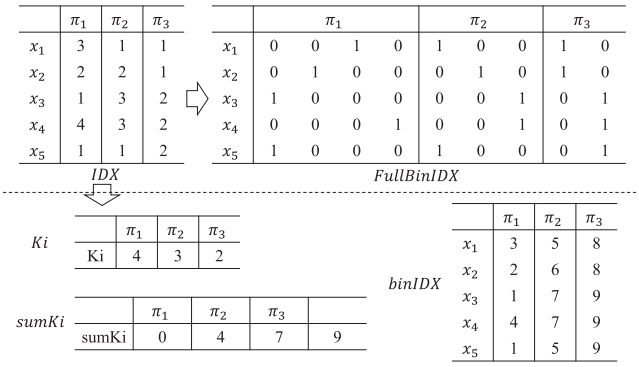


Fig. 2. An illustrative example of the binary matrix and sparse representation.

functionality for producing the input directly used by the function **KCC**. It utilizes the techniques as described in Sections 3.2 and 3.3 to help to save memory and accelerate computations. The **RunKCC** function provides the functionality for combining the two functions **Preprocess** and **KCC**. It performs basic partition preprocessing and consensus clustering in one step. Moreover, it obtains the best consensus result by running the KCC algorithm in multiple repeated experiments.

Evaluating the clustering solutions in the consensus partition is also an important concern in the implemented package. The functions **inMeasure** and **exMeasure** provide the functionality for measuring the quality of a clustering when the ground-truth cluster label information is unavailable and available, respectively. In practice, the ground-truth labels are usually annotated by human experts [18]. The function **inMeasure** evaluates a clustering by examining how well the clusters are separated and how compact the clusters are. It calculates three internal validity indices, i.e., the Distortion Score [6], Silhouette Coefficient [27], and Calinski and Harabasz index [8]. The function **exMeasure** evaluates a clustering by comparing the clustering solution with the ground-truth labels. It calculates five external validity indices, including the classification accuracy [41] (CA), normalized mutual information [10] (NMI), normalized Rand statistic [44] (R_n), normalized van Dongen criterion [13] (VD_n), and normalized Variation of Information [10] (VI_n).

3.2 Sparse Representation of $\mathcal{X}^{(b)}$

As discussed in Section 2.2.2, the binary dataset $\mathcal{X}^{(b)}$ plays a key role in deriving the k-means heuristic, KCC utility functions, and input for the final ensemble clustering. It can be seen that $\mathcal{X}^{(b)}$ is typically a three-dimensional variable-length vector, since the length of its third dimension, i.e., K_i, is usually varied for different partition index i. In other words, considering the generality in real-world applications, different basic partition π_i may have different number of clusters K_i. Since programming languages cannot easily deal with such a variable-length vector, it is natural to flatten the last two dimensions of $\mathcal{X}^{(b)}$ to reduce $\mathcal{X}^{(b)}$ into a two-dimensional, i.e., $(n \times \sum_{i=1}^{r} K_i)$-dimensional, fix-length matrix *FullBinIDX* (see the upper right of the Figure 2). We call the *FullBinIDX* the binary matrix or the full matrix interchangeably hereinafter.

However, directly representing the binary dataset as a two-dimensional MATLAB matrix, i.e., *FullBinIDX*, is still problematic, since *FullBinIDX* is very sparse with many zeros and only a few ones, and it would consume a huge amount of memory space. Sparse representation has been a long-standing technique to deal with such memory intensive computations in implementation. For example, in MATLAB the built-in function **sparse** can convert a full matrix into a sparse form by squeezing out any zero elements. When the full matrix contains many zero elements, this

ALGORITHM 2: Algorithm for generating sparse representation of $\mathcal{X}^{(b)}$.

Input: basic partition result matrix IDX
Output: matrix Ki, index matrix $sumKi$, index matrix $binIDX$

```
Calculate number of data objects n, and number of basic partitions r from IDX;
Initialize Ki as a 1 × r matrix with all elements setting to 0s;
Initialize sumKi as a 1 × (r + 1) matrix with all elements setting to 0s;
Initialize binIDX as a n × r matrix with all elements setting to 0s;
for each p ∈ {1, 2, ..., r} do
    Set Ki_{1,p} ← max(IDX_{·,p}). // Computes the maximum value for each column in IDX;
    Set sumKi_{1,p+1} ← sumKi_{1,p} + Ki_{1,p};
    for each q ∈ {1, 2, ..., n} do
        Set binIDX_{q,p} ← sumKi_{1,p} + IDX_{q,p};
    end for
end for
return Ki, sumKi, and binIDX.
```

conversion technique can save much memory space. As such, it is essential to adopt a sparse representation technique to address this issue.

In the KCC package, we propose a sparse technique to represent the binary dataset with three matrices, i.e., $Ki \in \mathbb{R}^{1\times r}$, $sumKi \in \mathbb{R}^{1\times(r+1)}$, and $binIDX \in \mathbb{R}^{n\times r}$, where n is the number of data objects and r is the number of input basic partitions. This sparse representation can be regarded as a substitute for the full matrix $FullBinIDX$. More specifically, the matrix Ki contains the number of clusters for each of the basic partitions. The matrix $sumKi$ contains the starting column index for each of the basic partitions in $FullBinIDX$, while the last element of $sumKi$ indicates the ending column index for the last basic partition. The matrix $binIDX$ indicates the offset column position of the ones element for each data object in $FullBinIDX$. For facilitating understanding, the bottom of the Figure 2 gives an illustrative example of Ki, $sumKi$, and $binIDX$. The function **`Preprocess`** in the package produces Ki, $sumKi$, and $binIDX$. The pseudocode for generating the sparse representation is shown in Algorithm 2.

This representation technique can not only keep the essential basic partition information of $FullBinIDX$ but also avoids the need to store extra zeros. Particularly, by using the three matrices, we can recover the full matrix $FullBinIDX$; in other words, the three matrices can be seen as a sparse form of $FullBinIDX$. With this representation, we reduce the space complexity of storing $FullBinIDX$, i.e., $O(n\sum_{i=1}^{r} K_i)$, to the space complexity of storing Ki, $sumKi$, and $binIDX$, i.e., $O(nr)$. This saves a large amount of memory storage when the sum of the number of clusters in all basic partitions, i.e., $\sum_{i=1}^{r} K_i$, is large.

3.3 The Usage of Sparse Representation in Distance and Centroid Calculation

As indicated in Algorithm 1, $\mathcal{X}^{(b)}$ is the input of the k-means clustering for the final consensus clustering. Therefore, the next key question would be how the sparse representation of $\mathcal{X}^{(b)}$ can be used in the two-phase computation of k-means, i.e., the point-to-centroid distance and centroid computation. Since the KCC algorithm supports multiple utility functions, we take one utility function, i.e., U_c, as an example and show the complete pesudo code of the corresponding k-means heuristic in Algorithm 3.

The general idea in Algorithm 3 is to divide the computations of the cluster centroids and point-to-centroid distances into multiple smaller computations. Concretely, in initializing the cluster

ALGORITHM 3: Algorithm for consensus clustering with U_c over sparse representation.

Input: matrix Ki, matrix $sumKi$, matrix $binIDX$, matrix IDX, the number of clusters K for the consensus clustering, the number of data objects n, number of basic partitions r, matrix $weight \in \mathbb{R}^{r\times 1}$ indicating the weights for all basic partitions

Output: consensus partition π

Form an index list li by sampling K numbers randomly from the $\{1, 2, \ldots, n\}$ without replacement.
Initialize the centroids of K clusters m as a $K \times sumKi_{1,r+1}$ matrix with all elements setting to 0s;
for each $k \in \{1, 2, \ldots, K\}$ **do**
  Set $m_{k,binIDX(li,.)} \leftarrow 1$; // set values at corresponding indexes of m to 1s
end for
Initialize the point-to-centroid distances D as a $n \times K$ matrix with all elements setting to 0s;
Initialize the cluster assignment matrix π as a $n \times 1$ matrix with all elements setting to 0s;
Initialize a temporary matrix m' as a $K \times r$ matrix with all elements setting to 0s;
Initialize a temporary matrix $ones$ as a $K \times r$ matrix with all elements setting to 1s;
repeat
  for each $p \in \{1, 2, \ldots, r\}$ **do**
    Set $m'_{.,p} \leftarrow \sum_{q=sumKi_{1,p}+1}^{sumKi_{1,p+1}} m^2_{.,q}$; // calculating the sum square of all components in the centroid matrix
  end for
  for each $i \in \{1, 2, \ldots, n\}$ **do**
    Set $D_{i,.} \leftarrow (m' - 2m_{.,binIDX_{i,.}} + ones) * weight$; // The symbol * denotes the matrix multiplication operator
  end for
  Update cluster assignment matrix $\pi \leftarrow \min(D)$. // Assign each data object i to the cluster within the minimum distance.
  Calculate the distribution for each of the basic partition based on π and IDX to form a histogram count matrix $counts \in \mathbb{R}^{\max(Ki)\times r}$.
  for each $k \in \{1, 2, \ldots, K\}$ **do**
    Calculate the num of data objects in the kth cluster of the consensus partition as $count_k$.
    for each $i \in \{1, 2, \ldots, r\}$ **do**
      Update the cluster centroid matrix $m_{k,sumKi:sumKi_{i+1}} \leftarrow counts_{1:Ki_{1,i},i}/count_k$, where $counts_{1:Ki_{1,i},i}$ corresponds to the column vector $\langle p^{(i)}_{k1}, p^{(i)}_{k2}, \ldots, p^{(i)}_{kKi} \rangle^\top$ as described in Equation (15).
    end for
  end for
until convergence;
return π.

centroid matrix $C \in \mathbb{R}^{K\times sumKi_{1,r+1}}$, instead of using the full matrix $FullBinIDX$, we divide the initialization process into three smaller subprocesses: (1) initializing a $K \times sumKi_{1,r+1}$ matrix with all elements setting to 0s, (2) randomly sampling an index list to indicate the row indexes of the data objects selected as the initialized centroids, and (3) constructing the final centroid matrix by setting the values at the positions indicated by the offset matrix $binIDX$ to 1s. An example illustrating this process is shown in Figure 3.

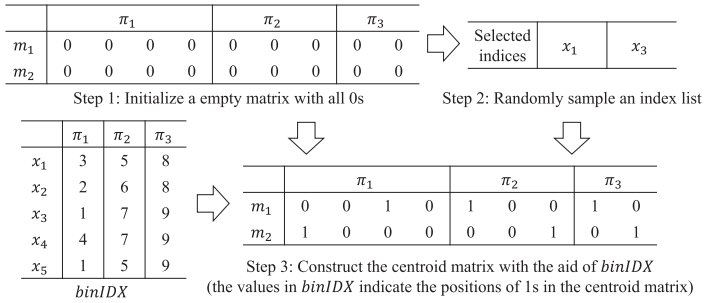


Fig. 3. An illustrative example of initializing the centroid matrix using $binIDX$ when $K = 2$.

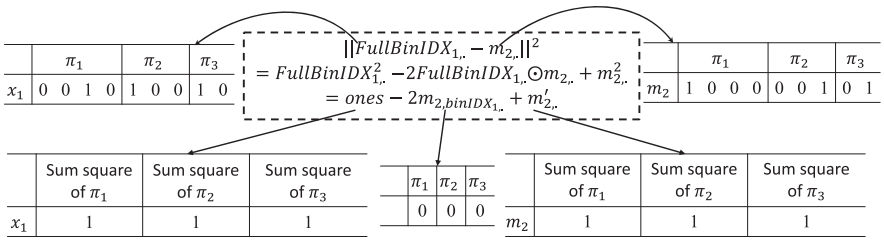


Fig. 4. An illustrative example of computing the distance from the data object $i = 1$ to the second cluster centroid at the first iteration using $binIDX$.

In the point-to-centroid distance computation, at each iteration, we divide the computation of the Euclidean distance from each data object i to the kth cluster centroid, i.e., $||FullBinIDX_{i,.} - m_{k,.}||^2$, into the sum of three smaller terms, including $FullBinIDX^2_{i,.}$, $-2FullBinIDX_{i,.} \odot m_{k,.}$, and $m^2_{k,.}$. Figure 4 illustrates an example of such a distance computation. Thanks to the division, the first term $FullBinIDX^2_{i,.}$ is always equal to a $1 \times r$ constant matrix with all values equal to 1, which could be omited in implementation. Moreover, the computationally expensive element-wise multiplication operation in the second term $-2FullBinIDX_{i,.} \odot m_{k,.}$ can be transformed into a simple matrix indexing operation, i.e., $-2m_{k,binIDX_{i,.}}$, in MATLAB which utilizes the sparse representation $binIDX$ rather than the full matrix $FullBinIDX$. In this way, the KCC package achieves a space complexity of $O(nr + nK + n)$ in the k-means heuristic with the sparse representation.

4 NUMERICAL EXPERIMENTS

In this section, we conduct multiple numerical experiments on 11 diverse datasets to empirically evaluate our proposed KCC package. First, we evaluate the performances of the package with different utility functions. We then report the performances of the KCC package compared to four alternative consensus clustering packages. Next, we investigate the impact of the number of clusters in the consensus partition, number of basic partitions, and different basic partition generation strategies. Finally, we present the performance of the KCC package on handling incomplete basic partitions. All experiments were conducted on a Linux Server with Intel Xeon E5-2687W v3 3.10GHz CPUs and 503G memory. Some of the numerical experiments are also provided as examples to illustrate the usage of the package in Section 2 of the user manual that accompanies the software.

Table 3. Statistics of Real-world Datasets

Datasets	Source	#Objects	#Attributes	#Classes	MinClassSize	MaxClassSize	CV	Density
breast_w	UCI	699	9	2	241	458	0.439	1.0000
ecoli	UCI	332	7	6	5	143	0.899	1.0000
iris	UCI	150	4	3	50	50	0.000	1.0000
pendigits	UCI	10,992	16	10	6,330	6,864	0.042	1.0000
satimage	UCI	4,435	36	6	415	1,072	0.425	1.0000
dermatology	UCI	358	33	6	20	111	0.509	1.0000
wine	UCI	178	13	3	48	71	0.194	1.0000
mm	TREC	2,521	126,373	2	1,133	1,388	0.143	0.0015
reviews	TREC	4,069	126,373	5	137	1,388	0.640	0.0015
la12	TREC	6,279	31,472	6	521	1,848	0.503	0.0048
sports	TREC	8,580	126,373	7	122	3,412	1.022	0.0010

4.1 Datasets from UCI and TREC Repositories

We use multiple datasets acquired from two large data repositories, i.e., the UCI[2] and **Text REtrieval Conference (TREC)**[3] repositories, in the experiments. Each dataset contains multiple attributes of the data objects, and their corresponding classes, which provide ground-truth labels for evaluating the clustering performance with external validity metrics. The *breast_w* [56] dataset is obtained from the clinical cases of Dr. Wolberg in the breast cancer databases of the University of Wisconsin Hospitals, Madison, and the classes correspond to the diagnosis result of the cases, i.e., benign and malignant. The *ecoli* [40] dataset contains protein information with known localization sites. The *iris* [15] dataset contains the attribute information of iris plants, and the classes correspond to the plants' types. The *pendigits* [2] dataset is derived from a handwritten digit database with the numbers 0 to 9 being the class labels. The *satimage* [51] dataset consists of the multispectral values of pixels in 3×3 neighborhoods in a satellite image, and the classes correspond to the central pixel in each neighborhood. The *dermatology* [11] dataset contains 33 attributes indicating clinical and histopathological features for diagnosis of the type of Eryhemato-Squamous Disease. The *wine* [1] dataset contains chemical features of wines, and the classes correspond to the origin of wines. The *mm*, *reviews*, *la12*, and *sports* datasets are all text datasets collected from the TREC data repository [19]. For example, the *reviews* dataset is derived from *San Jose Mercury* newspaper articles, and contains text documents about food, movies, music, radio, and restaurants. The *la12* dataset is obtained from articles of the *Los Angeles Times*, and its classes include entertainment, financial, foreign, metro, national, and sports desks. Some dataset statistics are given in Table 3, where *CV* denotes the variation coefficient statistic that measures the cluster imbalance of ground-truth data and *Density* denotes the portion of nonzero elements. For preprocessing these text documents, readers can refer to the technical report [26] of the clustering toolkit CLUTO[4] for more details.

4.2 Comparison of Clustering Quality and Efficiency with Different Utility Functions

Here we evaluate the clustering quality and efficiency of the KCC package with 10 different utility functions on the 11 real-world datasets. For basic partition generation, we use RPS as the strategy.

For clustering quality evaluation, we only show the results of one metric, i.e., *CA*, in Table 4 due to the page limit. The full results on all five external metrics including *CA*, *NMI*, R_n, VI_n,

[2] https://archive.ics.uci.edu/ml/
[3] http://glaros.dtc.umn.edu/gkhome/fetch/sw/cluto/datasets.tar.gz
[4] http://glaros.dtc.umn.edu/gkhome/home-of-cluto

Table 4. Clustering Quality of KCC with Different Utility Functions in Terms of *CA*

	U_c	U_H	U_{cos}	U_{L5}	U_{L8}	NU_c	NU_H	NU_{cos}	NU_{L5}	NU_{L8}
breast_w	0.6403	0.9624	0.7172	0.6820	0.6896	0.6260	**0.9639**	0.6838	0.6820	0.6820
ecoli	0.5639	0.5789	0.5651	0.5756	0.5590	0.5831	0.5578	**0.5880**	0.5605	0.5657
iris	0.8940	0.8973	**0.9000**	**0.9000**	0.8940	0.8933	0.8907	**0.9000**	**0.9000**	0.8880
pendigits	0.6526	0.6584	**0.6837**	0.6422	0.6520	0.5947	0.6443	0.6833	0.6600	0.6533
satimage	0.5829	0.6571	0.6149	0.6060	0.6290	0.5229	0.6425	**0.6594**	0.6006	0.5597
dermatology	0.2913	**0.3128**	0.2729	0.2701	0.2723	0.3075	0.2975	0.2802	0.2760	0.2899
wine	0.5135	**0.5247**	0.5180	0.5112	0.5112	0.5208	0.5079	0.5219	0.5157	0.5152
mm	0.9337	0.9497	0.9532	0.9336	**0.9551**	0.7797	0.9495	0.9415	0.9439	0.9093
reviews	0.6153	0.6593	0.6313	0.6407	0.6569	0.6086	**0.6698**	0.6030	0.6353	0.6198
la12	0.4626	0.4912	**0.5064**	0.4755	0.4690	0.4349	0.4890	0.4853	0.4491	0.4476
sports	0.4497	**0.4858**	0.4715	0.4642	0.4502	0.4592	0.4584	0.4764	0.4693	0.4461
score	0.9182	**0.9898**	0.9462	0.9283	0.9314	0.8910	0.9707	0.9477	0.9265	0.9135

The best results are highlighted with bold.

Table 5. Full Execution Time of the KCC Package with Different Utility Functions

	U_c	U_H	U_{cos}	U_{L5}	U_{L8}	NU_c	NU_H	NU_{cos}	NU_{L5}	NU_{L8}
breast_w	0.83	1.95	0.72	0.88	0.76	0.82	0.99	**0.64**	0.76	0.86
ecoli	1.61	3.50	1.24	1.67	1.51	**1.16**	2.06	1.23	1.50	1.40
iris	2.00	7.06	1.61	1.51	1.51	1.49	3.39	1.48	**1.46**	1.48
pendigits	**10.55**	12.87	11.45	13.84	11.87	11.39	13.00	11.19	14.01	13.12
satimage	1.25	1.80	**1.19**	1.42	1.73	1.22	1.62	1.22	1.58	1.96
dermatology	**1.11**	3.59	1.47	1.37	1.15	1.24	1.90	1.20	1.23	1.29
wine	1.86	6.55	2.30	1.84	1.61	1.80	3.28	1.78	1.56	**1.53**
mm	622.61	771.60	733.84	**571.96**	677.55	660.21	739.98	774.26	656.56	588.42
reviews	733.01	1209.34	769.65	693.93	647.68	977.17	1017.01	767.63	756.06	**629.17**
la12	255.44	302.76	283.09	258.23	252.70	266.36	**249.24**	256.89	267.51	252.64
sports	1043.65	1701.98	954.64	1008.88	929.79	989.38	1032.93	**899.55**	952.23	980.68
score	0.55	0.98	0.57	0.57	0.56	0.56	0.69	**0.54**	0.58	0.57

We collect the raw execution time in milliseconds, and normalize it by the number of data objects of the corresponding dataset. The best results are highlighted with bold.

and VD_n can be found in Section 2 of the user manual that accompanies the software. The results indicate that 7 of 10 utility functions achieve the best clustering performance over at least 1 dataset. This suggests providing flexible utility functions can be crucial to accurate ensemble clustering in real-world applications. In practice, we can hardly know which utility function should be used in consensus clustering. We recommend to follow the advice of Wu et al. [58] to rate utility functions over a testbed and select the utility function that achieves the best rating. More specifically, a final score is defined to assess the overall performance of a utility function on a set of datasets. The score is calculated as $score(U_i) = \frac{1}{11}\sum_j \frac{V(U_i,D_j)}{\max_i V(U_i,D_j)}$, where $V(U_i, D_j)$ denotes the clustering validity score obtained by applying a utility function U_i on a dataset D_j. We observe that U_H obtains the best score on all five validity metrics and is closely followed by NU_H. As such, we take U_H as the default choice for the KCC package unless otherwise specified.

For efficiency evaluation, we show the full execution time of the KCC package with different utility functions and datasets in Table 5. In the computation of execution time, we consider the whole process of using the package, including loading data, generating basic partitions, conducting consensus function, and evaluating the clustering quality. As can be seen, NU_{cos} achieves the best score in terms of efficiency on the 11 datasets.

Table 6. Clustering Quality of KCC and Alternative Packages in Terms of CA

	KCC	ClusterEnsemble	CLUE	LinkCluE	OpenEnsembles
breast_w	**0.9624**	0.9528	0.6579	0.6544	0.9585
ecoli	0.5789	0.5361	0.5557	**0.6892**	0.4767
iris	0.8973	**0.9733**	0.8587	0.8120	0.8867
pendigits	0.6584	**0.6760**	0.6345	0.2940	—
satimage	0.6571	0.6631	0.5499	0.2421	**0.6812**
dermatology	0.3128	0.2877	0.2665	**0.3883**	0.2709
wine	0.5247	0.5056	**0.5270**	0.3815	0.5000
mm	**0.9497**	0.9357	0.5945	0.5510	0.5502
reviews	**0.6593**	0.5766	0.4596	0.3414	0.4896
la12	0.4912	**0.5023**	0.3376	0.2943	0.3464
sports	**0.4858**	0.4243	—	0.3971	—
score	**0.9527**	0.9250	0.7800	0.6845	0.8063

The symbol "—" indicates that it fails to produce results. The best results are highlighted with bold.

4.3 Comparison of Clustering Quality and Efficiency with Alternative Packages

We also compare the clustering quality and efficiency of the KCC package with four alternative consensus clustering packages, including ClusterEnsemble [50], CLUE [21], LinkCluE [24], and OpenEnsembles [46], on the 11 real-world datasets. ClusterEnsemble converts the basic partition results into a representation of hypergraph and solves the consensus clustering problem over the hypergraph with three heuristics, i.e., the HGPA, CSPA, and MCLA. For evaluation, we chose the best clustering result of the three heuristics. CLUE implements multiple methods for minimizing the dissimilarity between the consensus partition and several basic partitions. For evaluation, we used the default choice of the CLUE package in obtaining the consensus partitions, i.e., a fixed-point algorithm for obtaining soft least squares Euclidean consensus partitions. LinkCluE is a variant of co-association matrix-based methods and replaces the co-association matrix with a better similarity matrix by using three link-based measures, i.e., the connected-triple-based similarity, SimRank-based similarity, and Approximate SimRank-based Similarity. For evaluation, we used the connected-triple-based similarity with single link as the baseline. OpenEnsembles is a Python package, which implements majority vote, mixture models, and two additional co-association matrix-based methods for consensus clustering. For evaluation, we used the majority vote method as the baseline.

In Tables 6–10, we report the five external validity metrics as the measurements of the clustering quality for the five packages. From the results, we can see that the KCC package achieves superior scores compared to the alternative packages in terms of CA, NMI, and R_n. For VI_n and VD_n, the ClusterEnsemble package obtains the highest scores, but the difference between KCC and ClusterEnsemble is not large. This indicates that our proposed KCC is at least comparable to alternative packages in terms of clustering quality. Notably, some packages have already been computationally intractable on the sample datasets, such as CLUE on the *sports* dataset and OpenEnsembles on the *pendigits* and *sports* datasets. They either fail to produce results within a week or run out of memory on the Linux server.

For fair comparison of efficiency, we only compare the KCC package with two alternative packages that are also implemented in MATLAB i.e., the ClusterEnsemble and LinkCluE packages. For the KCC package, we adopt the widely used utility function U_c in efficiency comparison. We report the full execution time and peak memory usage of these three packages in Tables 11 and 12, respectively. Note that we only report the results of peak memory usage on the datasets from the

Table 7. Clustering Quality of KCC and Alternative Packages in Terms of NMI

	KCC	ClusterEnsemble	CLUE	LinkCluE	OpenEnsembles
breast_w	**0.7558**	0.7153	0.2022	0.0009	0.7361
ecoli	0.5941	0.5898	0.5811	**0.6200**	0.5947
iris	0.7937	**0.9011**	0.7407	0.7184	0.7419
pendigits	**0.6775**	0.6708	0.5991	0.4341	—
satimage	0.5747	0.5337	0.4467	0.0178	**0.6138**
dermatology	0.1417	0.1353	0.0954	**0.3728**	0.1050
wine	**0.1697**	0.1613	0.1622	0.0899	0.1338
mm	**0.7249**	0.6711	0.0279	0.0065	0.0005
reviews	**0.5347**	0.4441	0.3535	0.0112	0.2480
la12	0.3371	**0.3390**	0.1964	0.0161	0.1452
sports	**0.4641**	0.3901	—	0.0125	—
score	**0.9227**	0.8789	0.6129	0.3729	0.6355

The symbol "—" indicates that it fails to produce results. The best results are highlighted with bold.

Table 8. Clustering Quality of KCC and Alternative Packages in Terms of R_n

	KCC	ClusterEnsemble	CLUE	LinkCluE	OpenEnsembles
breast_w	**0.8537**	0.8176	0.0932	-0.0001	0.8391
ecoli	0.4230	0.4075	0.3932	0.5265	**0.5286**
iris	0.7445	**0.9222**	0.6875	0.6193	0.7163
pendigits	**0.5236**	0.5147	0.4761	0.0858	—
satimage	0.5018	0.4479	0.3377	0.0001	**0.5345**
dermatology	0.0553	0.0602	0.0275	**0.1563**	0.0279
wine	**0.1497**	0.1454	0.1490	-0.0010	0.1254
mm	**0.8092**	0.7594	0.0353	0.0002	-0.0001
reviews	**0.5193**	0.3833	0.1635	-0.0001	0.0612
la12	**0.2671**	0.2670	0.0453	0.0000	0.0206
sports	**0.3193**	0.2657	—	-0.0003	—
score	**0.9000**	0.8559	0.4839	0.2567	0.5523

The symbol "—" indicates that it fails to produce results. The best results are highlighted with bold.

UCI repository, because the profiling processes on the datasets of the TREC repository are computationally intractable, i.e., each algorithm does not produce results after running more than a week on the testing Linux server. We can see that the KCC package is much more efficient than the two alternative packages in terms of both execution time and peak memory usage.

4.4 Impact of the Number of Clusters in the Consensus Partition

The number of clusters in the consensus partition, i.e., K, is an important user-defined parameter for the KCC package. We conduct an analysis on how the three internal metrics, i.e., Distortion Score, Silhouette Coefficient, and Calinski and Harabasz index, vary with increasing K on the *iris* dataset. The results are shown in Figure 5. From the figure, we can see that on the *iris* dataset, all three internal metrics generally decrease with the increase of K. Moreover, we show the execution time of KCC with varying K on three different datasets in Figure 6. The results indicate that the execution time approximately increases linearly to the number of clusters on all three datasets.

A related important question in practice is how to determine the number of clusters for the consensus clustering. Based on the three internal metrics as in Figure 5, we implement three methods

Table 9. Clustering Quality of KCC and Alternative Packages in Terms of VI_n

	KCC	ClusterEnsemble	CLUE	LinkCluE	OpenEnsembles
breast_w	**0.2442**	0.2848	0.7979	0.9976	0.2639
ecoli	0.4079	0.4129	0.4213	**0.3969**	0.4064
iris	0.2064	**0.0989**	0.2595	0.2832	0.2581
pendigits	**0.3227**	0.3292	0.4010	0.6331	—
satimage	0.4253	0.4664	0.5534	0.9972	**0.3862**
dermatology	0.8584	0.8648	0.9046	**0.6349**	0.8950
wine	**0.8303**	0.8387	0.8378	0.9231	0.8662
mm	**0.2752**	0.3289	0.9721	0.9991	0.9993
reviews	**0.4660**	0.5568	0.6465	0.9982	0.7596
la12	0.6630	**0.6611**	0.8049	0.9973	0.8651
sports	**0.5392**	0.6132	—	0.9980	—
score	0.6067	**0.6028**	0.8240	0.9676	0.7871

The symbol "—" indicates that it fails to produce results. The best results are highlighted with bold.

Table 10. Clustering Quality of KCC and Alternative Packages in Terms of VD_n

	KCC	ClusterEnsemble	CLUE	LinkCluE	OpenEnsembles
breast_w	**0.1087**	0.1419	0.9087	0.9992	0.1221
ecoli	0.4323	0.4493	0.4367	0.3690	**0.3933**
iris	0.1675	**0.0404**	0.2194	0.2814	0.1799
pendigits	**0.3386**	0.3642	0.3843	0.6990	—
satimage	0.3829	0.4275	0.5478	0.9988	**0.3481**
dermatology	0.8877	0.8609	0.9352	**0.7170**	0.9301
wine	0.7399	0.7500	**0.7355**	0.9927	0.7586
mm	**0.1083**	0.1372	0.8782	0.9991	1.0000
reviews	**0.4042**	0.5075	0.7695	0.9989	0.8250
la12	0.6286	**0.6024**	0.8676	0.9991	0.9076
sports	**0.5335**	0.5789	—	0.9996	—
score	0.5368	**0.5216**	0.8017	0.9625	0.7198

The symbol "—" indicates that it fails to produce results. The best results are highlighted with bold.

to automatically select the number of clusters for consensus clustering. The first one is the Elbow method [5], which picks the elbow point of the Distortion score curve as the the best number of clusters. The second one is to choose the number of clusters that produces a clustering solution with the maximum value of average silhouette coefficient. The third one is to choose the number of clusters that produces a clustering solution with the maximum value of Calinski and Harabasz index. As indicated by the red line in Figure 5, all three methods consistently find the best number of clusters as $K = 2$ on the *iris* dataset.

4.5 Impact of the Number of Basic Partitions

The number of basic partitions is another important parameter when using the KCC package in practice. To study the impact of this parameter, we first generate 1,000 BPs as the basic partition set Π and then do random sampling on Π to generate subsets with different number of basic partitions, i.e., Π^r with $r = 10, 20, \ldots, 90$. Given a specific r, sampling is repeated in 100 runs, and KCC is conducted on each independent sample for reporting clustering performance. Here we use R_n as the measure, and each result is the average R_n over 10 runs. The results are reported on three

Table 11. Full Execution Time of KCC and Two Alternative Packages

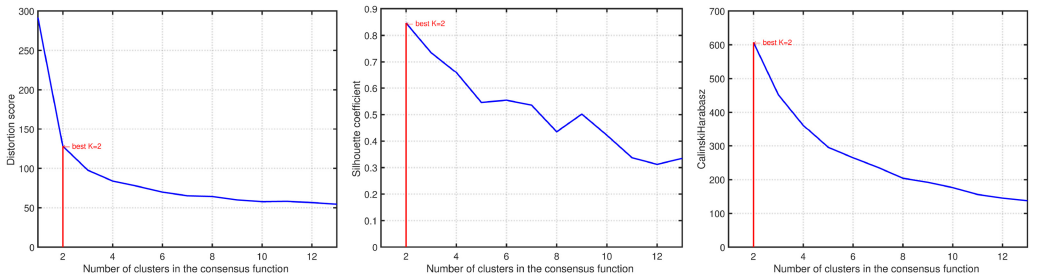
	KCC	LinkCluE	ClusterEnsemble
breast_w	0.83	15.10	8.58
ecoli	1.61	18.03	13.56
iris	2.00	23.98	22.26
pendigits	10.55	3620.00	103.87
satimage	1.25	59.94	7.59
dermatology	1.11	16.62	13.96
wine	1.86	20.32	19.16
mm	622.61	30291.15	1116.30
reviews	733.01	35340.38	1967.19
la12	255.44	14032.49	411.04
sports	1043.65	56274.34	2679.64
score	**0.04**	1.00	0.40

We collect the raw execution time in milliseconds, and normalize it by the number of data objects of the corresponding dataset. The best results are highlighted with bold.

Table 12. Peak Memory Usage of KCC and Two Alternative Packages (in Kilobytes)

	KCC	LinkCluE	ClusterEnsemble
breast_w	64	14204	15524
ecoli	52	8784	12628
iris	52	4380	5240
pendigits	127520	6663344	6821428
satimage	3476	153968	134872
dermatology	52	9908	12800
wine	52	5172	5700
score	**0.01**	0.87	0.98

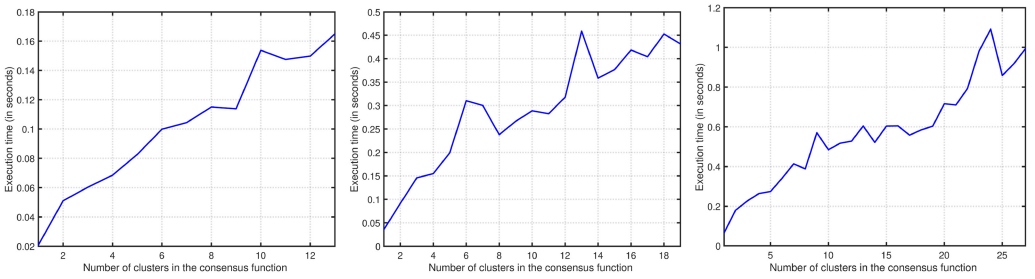
The best results are highlighted with bold.



(a) Distortion Score (b) Silhouette Coefficient (c) Calinski and Harabasz index

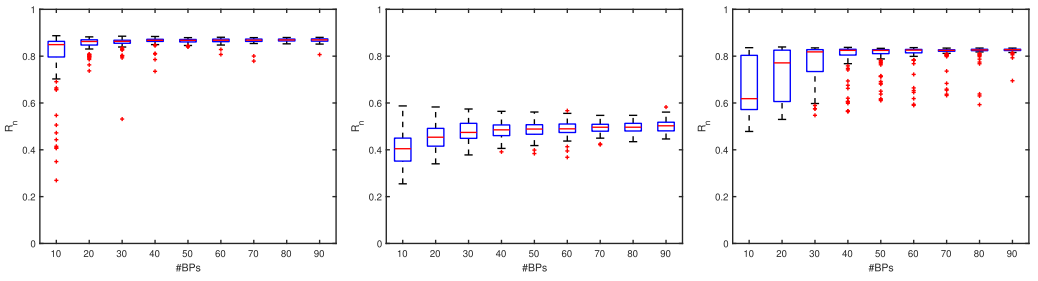
Fig. 5. Impact of the number of clusters in the consensus partition on the *iris* dataset.

example datasets, i.e., *breast_w*, *reviews*, and *mm*. The results in Figure 7 show that the deviations of the clustering performance tend to be reduced with the increase of r. This implies that a large number of BPs could increase the KCC's robustness. In practice, $r = 100$ is a reasonable choice for the number of BPs.



(a) *iris.* (b) *ecoli.* (c) *breast_w.*

Fig. 6. Execution time of KCC with varying number of clusters in the consensus partition on three datasets.



(a) breast_w (b) reviews (c) mm

Fig. 7. Impact of the number of basic partitions.

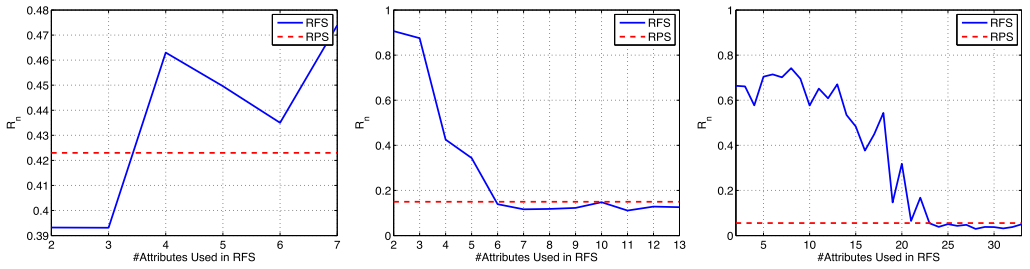
4.6 Impact of the Generation Strategy of Basic Partitions

We have used RPS as the BP generation strategy so far. Here in this subsection, we further study the performance of using RFS. For each dataset, we increase the number of attributes d used to generate BPs and showcase the clustering performance under different values of d. Experiments are conducted on three example datasets, i.e., *ecoli*, *wine*, and *dermatology*. The results are shown in Figure 8, where the red dashed line represents the benchmark clustering performance of using RPS. We can see that compared to RPS, RFS obtains significant performance gains on *wine* and *dermatology* with small values of d. This demonstrates that RFS is a useful substitute to RPS in some cases.

4.7 Performance on Incomplete Basic Partitions

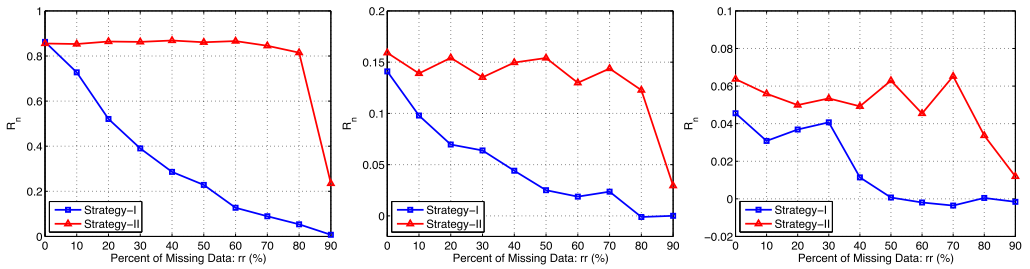
To validate KCC's effectiveness on handling IBPs, we illustrate two strategies for generating IBPs. In Strategy-I, we randomly eliminate objects from the original dataset to form a subset and conduct base clustering on the subset to produce IBPs. In Strategy-II, we conduct a base clustering on the complete dataset to form a complete BP and eliminate labels randomly from the BP to generate IBPs. The datasets *breast_w*, *wine*, and *dermatology* are used in this experiment with the default settings of KCC. The ratio rr denotes the removal portion, which ranges from 0% to 90%. As shown in Figure 9, IBPs with Strategy-II surprisingly bring barely adverse impact to the clustering performance except for $rr > 70\%$ on all three datasets. This experiment validates KCC's robustness in handling IBPs.

Remark. As validated by the above experiments, the KCC package is a simple yet effective and efficient package for solving consensus clustering problem with high robustness and generalization ability. First, the KCC package's execution time is of high efficiency. Due to the usage of the



(a) ecoli (b) wine (c) dermatology

Fig. 8. Clustering quality with RFS.



(a) breast_w (b) wine (c) dermatology

Fig. 9. Clustering quality on incomplete basic partitions.

k-means heuristic in solving the consensus clustering problem, the package has approximately linear complexity to the number of clusters in the consensus partition or number of data objects of the dataset. Second, the KCC package can fuse knowledge from a certain amount of basic partitions to reduce the performance variance in multiple independent runs. Moreover, the KCC package has the emergent property of potentially obtaining surprisingly high clustering quality even if only partial features are used to produce basic partitions. Last, the KCC package is capable of handling incomplete basic partitions and is robust to missing values in the raw data features or the basic partitions.

5 CONCLUSION

In this article, we presented a MATLAB package KCC, which implements the consensus clustering framework with flexible utility functions and a k-means heuristic. The current version of this package contains numerous useful functions, such as basic partition generation, preprocessing, consensus function, and clustering quality evaluation. The KCC package systematically implements the underlying KCC algorithm, with a focus on addressing the sparse implementation of the binary dataset, and distance/centroid computation using this sparse implementation. The efficiency and effectiveness of the KCC package were validated by the comparisons to multiple alternative packages on 11 real-life datasets. Impact factors of consensus clustering, such as the number of clusters for the consensus function, the number of BPs, the BPs' generation strategy, and the existence of IBPs, were further investigated to show the emergent properties of the package. The KCC package provides a data-driven statistical approach to consensus clustering and will have a significant impact due to its simplicity, effectiveness, and flexibility. In the future, we plan to investigate other sparse techniques to represent the binary dataset and further improve the implementation of the KCC package.

ACKNOWLEDGMENTS

The authors are grateful to the anonymous referees for their constructive comments on this article. The authors would also like to thank Dr. Guannan Liu for helpful discussions in the revision of the article.

REFERENCES

[1] Stefan Aeberhard, Danny Coomans, and Olivier de Vel. 1992. The classification performance of RDA. Technical Report, Department of Computer Science and Department of Mathematics and Statistics, James Cook University of North Queensland, 92–01.

[2] Fevzi Alimoglu and Ethem Alpaydin. 1996. Methods of combining multiple classifiers based on different representations for pen-based handwriting recognition. In *Proceedings of the 5th Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN'96)*.

[3] H. G. Ayad and M. S. Kamel. 2008. Cumulative voting consensus method for partitions with variable number of clusters. *IEEE Trans. Pattern Anal. Mach. Intell.* 30, 1 (Jan. 2008), 160–173. https://doi.org/10.1109/TPAMI.2007.1138

[4] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. 2005. Clustering with bregman divergences. *J. Mach. Learn. Res.* 6 (Oct. 2005), 1705–1749.

[5] Purnima Bholowalia and Arvind Kumar. 2014. EBK-means: A clustering technique based on elbow method and k-means in WSN. *Int. J. Comput. Appl.* 105, 9 (2014).

[6] Paul S. Bradley and Usama M. Fayyad. 1998. Refining initial points for k-means clustering. In *ICML*, Vol. 98. Citeseer, 91–99.

[7] Lev M. Bregman. 1967. The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. *USSR Comput. Math. Math. Phys.* 7, 3 (1967), 200–217.

[8] Tadeusz Caliński and Jerzy Harabasz. 1974. A dendrite method for cluster analysis. *Commun. Stat. Theory Methods* 3, 1 (1974), 1–27.

[9] Derek S. Chiu and Aline Talhouk. 2018. diceR: An R package for class discovery using an ensemble driven approach. *BMC Bioinf.* 19, 1 (2018), 1–4.

[10] Thomas M. Cover and Joy A. Thomas. 2012. *Elements of Information Theory*. John Wiley & Sons.

[11] G. Demiroz, H. A. Govenir, and N. Ilter. 1998. Learning differential diagnosis of eryhemato-squamous diseases using voting feature intervals. *Artif. Intell. Med.* 13, 3 (1998), 147–165.

[12] Carlotta Domeniconi and Muna Al-Razgan. 2009. Weighted cluster ensembles: Methods and analysis. *ACM Trans. Knowl. Discov. Data* 2, 4, Article 17 (Jan. 2009), 40 pages. https://doi.org/10.1145/1460797.1460800

[13] Stijn Dongen. 2000. *Performance Criteria for Graph Clustering and Markov Cluster Experiments*. Technical Report. Amsterdam, The Netherlands, The Netherlands.

[14] 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 *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231.

[15] Ronald A. Fisher. 1936. The use of multiple measurements in taxonomic problems. *Ann. Eugen.* 7, 2 (1936), 179–188.

[16] Edward B. Fowlkes and Colin L. Mallows. 1983. A method for comparing two hierarchical clusterings. *J. Am. Stat. Assoc.* 78, 383 (1983), 553–569.

[17] A. L. N. Fred and A. K. Jain. 2005. Combining multiple clusterings using evidence accumulation. *IEEE Trans. Pattern Anal. Mach. Intell.* 27, 6 (Jun. 2005), 835–850. https://doi.org/10.1109/TPAMI.2005.113

[18] Jiawei Han, Micheline Kamber, and Jian Pei. 2012. 10 - cluster analysis: Basic concepts and methods. In *Data Mining (Third Edition)*, Jiawei Han, Micheline Kamber, and Jian Pei (Eds.). Morgan Kaufmann, Boston, 443–495. https://doi.org/10.1016/B978-0-12-381479-1.00010-1

[19] Donna Harman. 1998. The text retrieval conferences (TRECs): Providing a test-bed for information retrieval systems. *Bull. Am. Soc. Inf. Sci. Technol.* 24, 4 (1998), 11–13.

[20] Kurt Hornik. 2005. A CLUE for CLUster ensembles. *J. Stat. Softw.* 14, 12 (Sep. 2005). https://doi.org/10.18637/jss.v014.i12

[21] Kurt Hornik. 2020. *Clue: Cluster Ensembles*.

[22] Dong Huang, Chang-Dong Wang, and Jian-Huang Lai. 2017. Locally weighted ensemble clustering. *IEEE Trans. Cybernet.* 48, 5 (2017), 1460–1473.

[23] Dong Huang, Chang-Dong Wang, Hongxing Peng, Jianhuang Lai, and Chee-Keong Kwoh. 2018. Enhanced ensemble clustering via fast propagation of cluster-wise similarities. *IEEE Trans. Syst. Man Cybernet.: Syst.* (2018).

[24] Natthakan Iam-on and Simon Garrett. 2010. LinkCluE: A MATLAB package for link-based cluster ensembles. *J. Stat. Softw.* 36, 1 (2010), 1–36. https://doi.org/10.18637/jss.v036.i09

[25] Nick Jardine and Cornelis Joost van Rijsbergen. 1971. The use of hierarchic clustering in information retrieval. *Inf. Stor. Retr.* 7, 5 (1971), 217–240.
[26] George Karypis. 2002. *CLUTO-a Clustering Toolkit.* Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, MN.
[27] Leonard Kaufman and Peter J. Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons.
[28] Hongmin Li, Xiucai Ye, Akira Imakura, and Tetsuya Sakurai. 2020. Ensemble learning for spectral clustering. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'20)*. IEEE, 1094–1099.
[29] Tao Li, Chris Ding, and Michael I. Jordan. 2007. Solving consensus and semi-supervised clustering problems using nonnegative matrix factorization. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM'07)*. IEEE Computer Society, 577–582. https://doi.org/10.1109/ICDM.2007.98
[30] Xue Li and Hongfu Liu. 2018. Greedy optimization for K-means-based consensus clustering. *Tsinghua Sci. Technol.* 23, 2 (2018), 184–194.
[31] Yinian Liang, Zhigang Ren, Zongze Wu, Deyu Zeng, and Jianzhong Li. 2020. Scalable spectral ensemble clustering via building representative co-association matrix. *Neurocomputing* 390 (2020), 158–167.
[32] Hongfu Liu, Tongliang Liu, Junjie Wu, Dacheng Tao, and Yun Fu. 2015. Spectral ensemble clustering. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*. ACM, New York, NY, 715–724. https://doi.org/10.1145/2783258.2783287
[33] Hongfu Liu, Ming Shao, Sheng Li, and Yun Fu. 2016. Infinite ensemble for image clustering. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, New York, NY, 1745–1754. https://doi.org/10.1145/2939672.2939813
[34] Hongfu Liu, Ming Shao, Sheng Li, and Yun Fu. 2018. Infinite ensemble clustering. *Data Min. Knowl. Discov.* 32, 2 (2018), 385–416.
[35] Hongfu Liu, Junjie Wu, Tongliang Liu, Dacheng Tao, and Yun Fu. 2017. Spectral ensemble clustering via weighted k-means: Theoretical and practical evidence. *IEEE Trans. Knowl. Data Eng.* 29, 5 (2017), 1129–1143.
[36] Zhiwu Lu, Yuxin Peng, and Jianguo Xiao. 2008. From comparing clusterings to combining clusterings. In *Proceedings of the 23rd National Conference on Artificial Intelligence, Volume 2 (AAAI'08)*, Vol. 2. AAAI Press, 665–670.
[37] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, Berkeley, CA, 281–297.
[38] B. Mirkin. 2001. Reinterpreting the category utility function. *Mach. Learn.* 45, 2 (2001), 219–228. https://doi.org/10.1023/A:1010924920739
[39] Andrew W. Moore. 2001. Clustering with gaussian mixtures. School of Computer Science, Carnegie Mellon University.
[40] Kenta Nakai and Minoru Kanehisa. 1991. Expert system for predicting protein localization sites in gram-negative bacteria. *Proteins Struct. Funct. Bioinf.* 11, 2 (1991), 95–110.
[41] Nam Nguyen and Rich Caruana. 2007. Consensus clusterings. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM'07)*. IEEE Computer Society, 607–612. https://doi.org/10.1109/ICDM.2007.73
[42] Amy L. Olex and Jacquelyn S. Fetrow. 2011. SC2ATmd: A tool for integration of the figure of merit with cluster analysis for gene expression data. *Bioinformatics* 27, 9 (2011), 1330.
[43] Lance Parsons, Ehtesham Haque, and Huan Liu. 2004. Subspace clustering for high dimensional data: A review. *SIGKDD Explor. Newsl.* 6, 1 (2004), 90–105.
[44] William M. Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* 66, 336 (1971), 846–850.
[45] Lior Rokach and Oded Maimon. 2005. *Clustering Methods.* Springer US, Boston, MA, 321–352. https://doi.org/10.1007/0-387-25465-X_15
[46] Tom Ronan, Shawn Anastasio, Zhijie Qi, Pedro Henrique S. Vieira Tavares, Roman Sloutsky, and Kristen M. Naegle. 2018. OpenEnsembles: A python resource for ensemble clustering. *J. Mach. Learn. Res.* 19, 26 (2018), 1–6.
[47] Michael Seiler, C. Chris Huang, Sandor Szalma, and Gyan Bhanot. 2010. ConsensusCluster: A software tool for unsupervised cluster discovery in numerical data. *OMICS* 14, 1 (Feb. 2010), 109–113. https://doi.org/10.1089/omi.2009.0083
[48] Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. 2008. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the ACM Conference on Recommender Systems (RecSys'08)*. ACM, New York, NY, 259–266. https://doi.org/10.1145/1454008.1454048
[49] Jianbo Shi and J. Malik. 2000. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 8 (Aug. 2000), 888–905. https://doi.org/10.1109/34.868688
[50] Alexander Strehl and Joydeep Ghosh. 2003. Cluster ensembles—A knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.* 3 (Mar. 2003), 583–617. https://doi.org/10.1162/153244303321897735

[51] E. Ke Tang, Ponnuthurai N. Suganthan, Xin Yao, and A. Kai Qin. 2005. Linear dimensionality reduction using relevance weighted LDA. *Pattern Recogn.* 38, 4 (2005), 485–493.

[52] Zhiqiang Tao, Hongfu Liu, Jun Li, Zhaowen Wang, and Yun Fu. 2019. Adversarial graph embedding for ensemble clustering. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*. International Joint Conferences on Artificial Intelligence Organization, 3562–3568. https://doi.org/10.24963/ijcai.2019/494

[53] Alexander Topchy, Anil K. Jain, and William Punch. 2003. Combining multiple weak clusterings. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*. IEEE Computer Society, 331–. http://dl.acm.org/citation.cfm?id=951949.952159

[54] Alexander Topchy, Anil K. Jain, and William Punch. 2004. A mixture model for clustering ensembles. In *Proceedings of the 4th SIAM International Conference on Data Mining*. 379–390. https://doi.org/10.1137/1.9781611972740.35

[55] Sandro Vega-Pons, Jyrko Correa-Morris, and Jose Ruiz-Shulcloper. 2010. Weighted partition consensus via kernels. *Pattern Recogn.* 43, 8 (Aug. 2010), 2712–2724. https://doi.org/10.1016/j.patcog.2010.03.001

[56] William H. Wolberg and Olvi L. Mangasarian. 1990. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proc. Natl. Acad. Sci. U.S.A.* 87, 23 (1990), 9193–9196.

[57] Junjie Wu, Hongfu Liu, Hui Xiong, and Jie Cao. 2013. A theoretic framework of K-means-based consensus clustering. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13)*. AAAI Press, 1799–1805.

[58] J. Wu, H. Liu, H. Xiong, J. Cao, and J. Chen. 2015. K-means-based consensus clustering: A unified view. *IEEE T. Knowl. Data En.* 27, 1 (Jan. 2015), 155–169. https://doi.org/10.1109/TKDE.2014.2316512

[59] J. Wu, H. Xiong, C. Liu, and J. Chen. 2012. A generalization of distance functions for fuzzy c-means clustering with centroids of arithmetic means. *IEEE Trans. Fuzzy Syst.* 20, 3 (Jun. 2012), 557–571. https://doi.org/10.1109/TFUZZ.2011.2179659

[60] Hye-Sung Yoon, Sun-Young Ahn, Sang-Ho Lee, Sung-Bum Cho, and Ju Han Kim. 2006. *Heterogeneous Clustering Ensemble Method for Combining Different Cluster Results*. Springer, Berlin, 82–92. https://doi.org/10.1007/11691730_9

[61] Caiming Zhong, Lianyu Hu, Xiaodong Yue, Ting Luo, Qiang Fu, and Haiyong Xu. 2019. Ensemble clustering based on evidence extracted from the co-association matrix. *Pattern Recogn.* 92 (2019), 93–106.

Received 13 May 2021; revised 14 June 2023; accepted 7 August 2023