IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 31, 2017, accepted March 15, 2017, date of publication April 4, 2017, date of current version May 17, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2690987

NC-Link: A New Linkage Method for Efficient Hierarchical Clustering of Large-Scale Data

YONGKWEON JEON[1], JAEYOON YOO[1], JONGSUN LEE[1], AND SUNGROH YOON[1,2], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, Seoul National University, Seoul 08826, South Korea
[2]Department of Neurology and Neurological Sciences, Stanford University, Stanford, CA 94305 USA

Corresponding author: Sungroh Yoon (sryoon@snu.ac.kr)

This work was supported in part by the Brain Korea 21 Plus Project in 2017, in part by a project from the Center for Research and Development of Police Science and Technology and the Korean National Police Agency through the Ministry of Science, ICT and Future Planning (MSIP) of Korea under Grant PAC000001, in part by the Bio & Medical Technology Development Program of the National Research Foundation of Korea through MSIP under Grant 2015M3A9A7029735, in part by the Institute for Information & communications Technology Promotion through MSIP under Grant 2014-0-00147 (Basic Software Research in Human-level Lifelong Machine Learning), and in part by the SBS Foundation.

ABSTRACT In various disciplines, hierarchical clustering (HC) has been an effective tool for data analysis due to its ability to summarize hierarchical structures of data in an intuitive and interpretable manner. A run of HC requires multiple iterations, each of which needs to compute and update the pairwise distances between all intermediate clusters. This makes the exact algorithm for HC inevitably suffer from quadratic time and space complexities. To address large-scale data, various approximate/parallel algorithms have been proposed to reduce the computational cost of HC. However, such algorithms still rely on conventional linkage methods (such as single, centroid, average, complete, or Ward's) for defining pairwise distances, mostly focusing on the approximation/parallelization of linkage computations. Given that the choice of linkage profoundly affects not only the quality but also the efficiency of HC, we propose a new linkage method named *NC-link* and design an exact algorithm for NC-link-based HC. To guarantee the exactness, the proposed algorithm maintains the quadratic nature in time complexity but exhibits only linear space complexity, thereby allowing us to address million-object data on a personal computer. To underpin the extensibility of our approach, we show that the algorithmic nature of NC-link enables single instruction multiple data (SIMD) parallelization and subquadratic-time approximation of HC. To verify our proposal, we thoroughly tested it with a number of large-scale real and synthetic data sets. In terms of efficiency, NC-link allowed us to perform HC substantially more space efficiently or faster than conventional methods: compared with average and complete linkages, using NC-link incurred only 0.7%–1.75% of the memory usage, and the NC-link-based implementation delivered speedups of approximately 3.5 times over the centroid and Ward's linkages. With regard to clustering quality, the proposed method was able to retrieve hierarchical structures from input data as faithfully as in the popular average and centroid linkage methods. We anticipate that the existing approximation/parallel algorithms will be able to benefit from adopting NC-link as their linkage method for obtaining better clustering results and reduced time and space demands.

INDEX TERMS Clustering algorithm, data mining, machine learning.

I. INTRODUCTION

Clustering, or cluster analysis, refers to a computational technique that can group similar objects and separate dissimilar objects. Clustering has been very popular in many fields as a tool for discovering new knowledge hidden in collected data. Types of existing clustering methods include the following [1], [2]: *Partitioning* algorithms divide input data into non-overlapping partitions and typically require the user to specify the number of clusters. *Hierarchical* methods discover hierarchical structures appearing in input data by gradually merging or splitting intermediate clusters. *Density-based* techniques search regions of high population density and can find arbitrarily shaped clusters. *Grid-based* approaches examine a number of cells forming a grid and perform clustering using the grid. *Model-based* algorithms build a model for each cluster and aim at finding the best fit for the model. *Two-way clustering* (or *biclustering*) considers both the rows and columns of a matrix to find meaningful submatrices. *Subspace clustering* focuses only on parts of the input data space and is useful for analyzing high-dimensional

data. Further details on clustering methods can be found in [1]–[10].

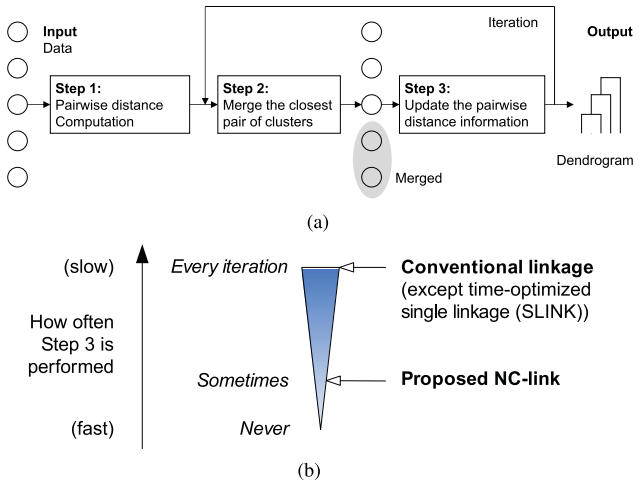


FIGURE 1. **Hierarchical clustering. (a) Overview (see Section IV-B for details). (b) The frequency of cluster distance updates (Step 3) critically affects the efficiency of hierarchical clustering.**

Among the numerous available clustering techniques, *hierarchical clustering* (HC; [2]–[4], [6]) is one of the most popular methods due to its ability to summarize hierarchical structures of data in an intuitive and interpretable manner. Starting with singleton clusters, HC repeatedly searches the two closest groups and then merges them until every object is grouped into one cluster, as informally shown in Fig. 1(a).[1] In the context of HC, a *linkage method* should be defined to calculate the distance between two clusters (see Fig. 4). The trace of the entire clustering process is usually recorded into a graphical representation called a *dendrogram* [2], [3], from which information on individual clusters can be derived. A dendrogram consists of many ⊓-shaped lines connecting objects in a hierarchical binary tree, as shown in Fig. 2(b). The leaves of a dendrogram correspond to individual objects, and a merge of two clusters is represented by an internal node, which is denoted by a dot in Fig. 2(b). The height of an internal node indicates the distance/dissimilarity at which the corresponding merge occurs. To complete the clustering of n objects, $n-1$ merges should occur, and the resulting dendrogram will have $n-1$ internal nodes. Note that flat clusters can be derived from a dendrogram by cutting it at a certain height [3].

As sketched above, a run of HC requires multiple iterations, each of which needs to compute and update the pairwise distances between all intermediate clusters. This makes the exact algorithm for HC require quadratic time and space complexity. To reduce the cost of HC, various approximate and/or parallel algorithms thus have been proposed [10]–[14]. However, these algorithms still rely on conventional linkage methods (such as single, centroid, average, complete, or Ward's)

[1]This bottom-up method is called *agglomerative* HC. We do not consider its top-down alternative called *devisive* HC in this paper.

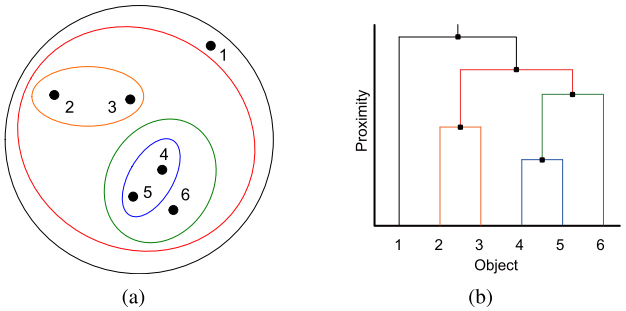


FIGURE 2. **An example. (a) A hypothetical hierarchy of clusters existing among six objects. Any HC algorithm needs five merges to cluster these six objects. (b) The corresponding dendrogram, where each merge is represented by a dot.**

for defining pairwise distances and mostly focus on the approximation and/or parallelization of linkage computation. With regard to enhancing clustering quality, multi-phase clustering algorithms have been designed on top of HC, including BIRCH [15], ROCK [16], and Chameleon [17]. However, the focus of these methods is on the integration of HC with other clustering methods for discovering complex clusters rather than on the improvement of the quality of HC itself.

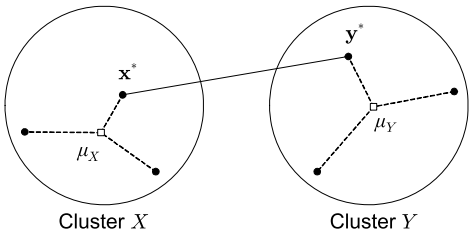


FIGURE 3. **In NC-link, the distance between two clusters is defined as the distance between two points, each of which is the closest to the centroid of each cluster. x* ≜ NC-point(X) and y* ≜ NC-point(Y).**

The choice of the linkage method profoundly affects not only the quality but also the efficiency of HC. To reach the root of the problem and derive a better solution therefrom, we propose in this paper a new linkage method termed *NC-link* (NC stands for nearest to centroid) and present an efficient implementation of the HC that uses NC-link as its linkage method. In NC-link, the distance between two clusters is defined as the distance between two data points, each of which is the closest to the centroid of each cluster (see Fig. 3 and Definitions 1–2 in Section III-A for more details). This linkage definition allows the proposed NC-link-based algorithm[2] to employ some key techniques for reducing space usage and response time. For instance, the distance update action is not needed after every merge when NC-link is used, as informally shown in Fig. 1(b); more details will follow shortly. To guarantee the exactness, the proposed algorithm maintains the quadratic nature in the time complexity, although it exhibits only a linear space complexity, which

[2]Where no confusion arises, we also refer to the proposed NC-link-based HC algorithm as NC-link for convenience.

is crucial for ensuring scalability. To underpin our method's extensibility to approximation and parallelization, we will show that the algorithmic nature of NC-link enables SIMD parallelization and subquadratic-time approximation for HC.

To verify our proposal, we thoroughly tested it with a number of large-scale real and synthetic data sets. According to our experiments, the proposed algorithm ran as fast as the average and complete linkage methods, therein only using 0.7–1.75% of the memory required by these methods. In addition, our method achieved up to 3.54 and 3.36 times speed up over the centroid and Ward's linkage methods, respectively, despite using a similar amount of memory. Furthermore, the in-memory processing of multiple millions of objects on a personal computer was possible by NC-link. In addition to the evaluation of the speed and scalability of NC-link, we also confirmed its clustering quality through a quantitative evaluation of clustering results by the cophenetic correlation coefficient [3], the Silhouette Index [18], and the measure of concordance (MoC) [19]. The results confirm that NC-link can retrieve the hierarchical structure as effectively as can the popular average and centroid linkages.

The remainder of this paper is organized as follows: Section II reviews the fundamentals of HC, including the conventional linkage methods. Section III presents our NC-link algorithm at length. The complexity analysis of the proposed method is given in Section IV, along with in-depth comparisons with existing techniques. Section V demonstrates the effectiveness of our NC-link methodology through experimental results, followed by a discussion in Section VI. Section VII concludes the paper.

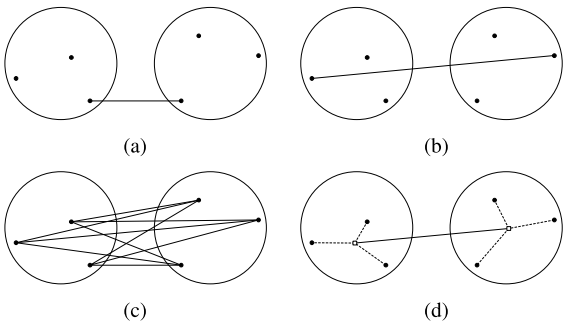


FIGURE 4. Conventional linkage methods. (a) Single. (b) Complete. (c) Average (UPGMA). (d) Centroid (UPGMC).

II. BACKGROUND

In HC, the closest pair of clusters is merged iteratively, and the measure of the distance between two clusters should thus be defined. Fig. 4 compares four common methods for this. One of the earliest measures proposed was the nearest-neighbor or single linkage method, which defines the cluster distance as the distance between the two closest objects, one from each cluster, as presented in Fig. 4(a). Fig. 4(b) shows the farthest-neighbor or complete linkage method, in which the cluster distance is defined as the distance between the two most distant objects. The average linkage or *unweighted pair-group method using arithmetic averages* (UPGMA) shown in Fig. 4(c) utilizes the group average, that is, the average pairwise distance among all pairs of points in the different clusters. In the centroid linkage or *unweighted pair-group method using centroids* (UPGMC), the cluster distance is defined as the distance between the centroids of two clusters, as depicted in Fig. 4(d). Other linkage methods not presented in Fig. 4 include the median linkage or *weighted pair-group method using centroids* (WPGMC), which uses the weighted centroid of clusters, and Ward's linkage method, which defines the distance between two clusters as the increase in the error sum of squares resulting from merging them [3].

The linkage method used by an HC algorithm not only determines the clustering result produced by the algorithm but also critically affects the time and space complexity of the algorithm. After two clusters are merged into a new cluster, the distances between this new cluster and the other unchanged clusters need to be updated before the next merge. Since this update occurs in every iteration of the conventional HC, the update time accounts for a major portion of the entire running time. If single linkage is used, the update can be omitted entirely by pre-calculating the distance between every pair of objects. Sibson exploited this property and proposed an algorithm called *SLINK* [20] that is time optimal for single-linkage-based HC. However, single linkage has an obvious disadvantage called *the chaining effect* [21], [22], which results in long strings of objects being assigned to the same cluster. Thus, despite its efficiency, the single-linkage method is of limited value for clustering [4]. All other linkage methods mentioned above require distance updates at every iteration and are generally less efficient than the single-linkage-based approach, although they are typically more robust.

The worst-case time complexity of a naïve implementation of the conventional HC algorithm would be $O(n^3)$ since there are $O(n)$ iterations, each of which involves $O(n^2)$ operations to find the closest pair in the distance matrix. The naïve method can be improved with priority queues by reducing the search space to detect the closest pair. Retrieving only the fronts of the queues takes $O(n)$, resulting in a time complexity of $O(n^2 \log n)$. The $\log n$ is the operation cost of a priority queue [23]. It has been proven that the worst-case time complexity of the agglomerative HC algorithm is $O(n^2)$ for the single, centroid, complete, average, and Ward's linkage methods [20], [23], [24]. See Section IV-B for more details.

III. PROPOSED NC-LINK METHOD

A. PRELIMINARIES

We assume that the input data are given as a real matrix $D \in \mathbb{R}^{n \times m}$, where n and m are the numbers of objects and dimensions (or features) of the input data, respectively. A cluster is defined as an m-dimensional submatrix of D and is denoted by $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ for $1 \leq i \leq k$. The centroid of X is denoted by $\mu_X \in \mathbb{R}^m$. As a distance measure, we use the Euclidean distance or the

L_2 norm denoted by $||\cdot||$ in this paper; depending on the application, a different type of distance may be employed. In NC-link, we define the distance between two clusters using the concept of the NC-point of a cluster:

Definition 1: For cluster X, its element nearest to the centroid of X is called NC-point(X). Specifically, NC-point(X) $= \underset{\mathbf{x}\in X}{\text{argmin}}||\mathbf{x} - \mu_X||$.

Definition 2: For two clusters X and Y, the distance between X and Y is defined as $||\mathbf{x}^* - \mathbf{y}^*||$, where $\mathbf{x}^* =$ NC-point(X) and $\mathbf{y}^* =$ NC-point(Y).

Algorithm 1 NC-Link

input: Data matrix $D \in \mathbb{R}^{n\times m}$ ▷ n objects; m dimensions
input: Parameter δ ($0 < \delta \ll n$) ▷ Control space usage
output: Cluster information represented by a dendrogram

1: **procedure** NC-link(D, δ)
2: Initialize GQ and LQ (priority queues) and T (a dendrogram);
3: FillQueues(D, δ, GQ, LQ); ▷ Algorithm 2
4: **while** TRUE **do**
5: $(v, w) \leftarrow GQ$.Dequeue();
6: **if** each of v and w is the NC-point of a certain cluster **then**
7: Merge the two clusters v and w;
8: Add a node to T reflecting this merge;
9: **break if** T has $n - 1$ nonleaf nodes;
10: $r \leftarrow$ the NC-point of the merged cluster;
11: **if** $((r \neq v) \wedge (r \neq w))$ **then**
12: UpdateQueue(D, r, δ, GQ, LQ); ▷ Algorithm 3
13: **else if** $r == v$ **then**
14: **if** $LQ[v]$.IsEmpty() **then**
15: UpdateQueue(D, v, δ, GQ, LQ); ▷ Algorithm 3
16: **else**
17: GQ.Enqueue($LQ[v]$.Dequeue());
18: **else if** v is a NC-point **then**
19: **if** $LQ[v]$.IsEmpty() **then**
20: UpdateQueue(D, v, δ, GQ, LQ); ▷ Algorithm 3
21: **else**
22: GQ.Enqueue($LQ[v]$.Dequeue());
23: **return** T;

B. ALGORITHM DESCRIPTION

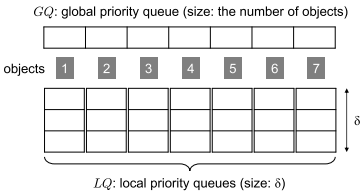
The pseudocode of NC-link is given in Algorithms 1–3. The input consists of a data matrix D, and $\delta \in \mathbb{Z}$, a user-defined parameter specifying the number of the nearest neighbors of each object that the algorithm will store during its run. The δ parameter thus controls the size of the space that the algorithm utilizes ($0 < \delta \ll n$). Of note is that the value of δ is irrelevant to the accuracy of the algorithm, meaning that invoking Algorithm 1 with different values of δ would produce the same set of clusters; δ only affects the execution time and space usage of the algorithm.

Algorithm 2 FillQueues

1: **procedure** FillQueues(D, δ, GQ, LQ)
2: **for** each object v in D **do**
3: **for** each object $w(\neq v)$ in D **do**
4: **if** w is one of the δ-nearest neighbors of v **then**
5: $LQ[v]$.Enqueue((v, w), $||v - w||$);
6: **for** each object v in D **do**
7: GQ.Enqueue($LQ[v]$.Dequeue());

Algorithm 3 UpdateQueue

1: **procedure** UpdateQueue(D, r, δ, GQ, LQ)
2: $LQ[r]$.Clear();
3: **for** each existing cluster K **do**
4: $k \leftarrow$ NC-point(K);
5: **if** k ($\neq r$) is one of the δ-nearest neighbors of r **then**
6: $LQ[r]$.Enqueue((r, k), $||r - k||$);
7: GQ.Enqueue($LQ[r]$.Dequeue());



COST OF OPERATION FOR A BINARY HEAP

Operation	Worst-case complexity
enqueue	$O(\log n)$
dequeue	$O(\log n)$
makeHeap	$O(n)$

FIGURE 5. An illustration of the concept of the global queue (*GQ*) and the local queues (*LQs*). Assume that the number of objects is 7, with the user-specified parameter $\delta = 3$. In the proposed algorithm, there exists a unique *GQ*, whereas one distinct *LQ* is allocated to each object (with the maximum queue size of δ elements). The δ-nearest neighbors (δ-NN) of each object are found and stored in the *LQ* dedicated to that object. The first entries of all pairs in an *LQ* are filled with the owner object of that *LQ*. After that, the information on the closest neighbor of each object is moved to *GQ*. Throughout the algorithm, *GQ* holds the pairs including each NC-Point as its first (and unique) entry. Each of the priority queues is implemented with a binary heap, and the computational cost of each heap operation is listed in the table shown above.

Line 2: The algorithm initializes its internal data structures. A dendrogram denoted by T is maintained to record the clustering procedure, the centroid of each of the merged clusters, and their membership. In addition, the algorithm utilizes two types of priority queues [25]: a global queue (GQ) and a local queue (LQ). When the algorithm runs, it creates and maintains only one GQ, whereas each object in the input data is associated with its own instance of LQ that can hold up to δ elements. The roles of GQ and LQ will be explained shortly; refer to the caption for Fig. 5 for an example with additional details. In this paper, we implement each instance of GQ and LQ using a binary heap [25] and assume that the following operations are supported:

- Enqueue((v, w), $||v - w||$): For two distinct objects $v, w \in D$, this function inserts the pair (v, w) into the

priority queue, with $||v-w||$ as its priority. The elements in the priority queue are stored in ascending order with respect to the pairwise distance.
- Dequeue(): This function returns the pair at the front of the priority queue (*i.e.*, the pair with the smallest distance in the queue) and removes the pair from the priority queue.
- IsEmpty(): This is to check whether the priority queue is empty.
- Clear(): This function removes all the pairs from the priority queue.

Line 3 (Algorithm 2): This line invokes Algorithm 2 to fill up the priority queues with the initial pairwise distance information. Specifically, for each object in the input data, the algorithm first searches for the δ-nearest neighbors (δ-NN) of the object and inserts them into the LQ dedicated to the object (lines 2–5 of Algorithm 2). After the search is completed for all the objects, the "closest pair" of each object (*i.e.*, the pair formed by an object and its nearest neighbor) is dequeued from the LQ of that object and then enqueued to the GQ (lines 6–7 of Algorithm 2). Although the distance between every pair of objects in D is computed, only $(\delta \cdot n)$ pairs of objects are inserted into the priority queues in the end. The effect of δ on the running time will be discussed in Section V-B. Note that each object can be considered as a singleton cluster, and every object initially becomes the NC-point of the singleton cluster.

Line 4: The main loop starts and is repeated until all $(n-1)$ merges required for completing the HC of n objects have occurred.

Line 5: The pair at the front of the GQ is popped out. This pair is denoted by (v, w) and represents the most closely located pair among all the pairs currently residing in the GQ.

Line 6: The algorithm checks if v is the NC-point of its cluster. After the same test on w, the algorithm proceeds to the next line if each of v and w is the NC-point of a cluster. Otherwise, the algorithm proceeds to line 18.

Lines 7–9: The two clusters represented by v and w are merged into a new cluster, and the centroid of the newly created cluster is computed, which can be done in constant time by using the two existing clusters' centroids and their membership. A new node corresponding to this merge is inserted into the dendrogram T along with the centroid and the membership information. Having $(n-1)$ nonleaf nodes in T indicates the completion of clustering. If that is the case, the main loop finishes to terminate the algorithm in line 23; otherwise, it proceeds to the next step.

Line 10: The NC-point of the newly created cluster is computed and denoted by r.

Lines 11–12: When r is neither v nor w, $LQ[r]$ (*i.e.*, the local priority queue dedicated to r) may have outdated values, and the algorithm needs to clear and update $LQ[r]$. Algorithm 3 performs the update of $LQ[r]$ as follows: All pairs in $LQ[r]$ are dropped (line 2 of Algorithm 3). The distance between r and the NC-point of each of the existing clusters is computed, and only δ-NN of r are inserted into $LQ[r]$ (lines 3–6 of Algorithm 3). This is to ensure that the space usage is linear in n. Then, the closest pair of the object r is dequeued from the $LQ[r]$ and then enqueued to the GQ (line 7 of Algorithm 3).

Lines 13–17: If r is v, the pair whose first entry is v should be moved to GQ. $LQ[v]$ needs to be updated by invoking Algorithm 3 if it is empty; otherwise, the front element of $LQ[v]$ is removed and inserted into the GQ. Note that the pair (v, w) popped out from the GQ indicates that the pair is moved to the GQ via the LQ of v (which is the first entry of the pair). Therefore, if v is an NC-point, the pair whose first entry is v have to be moved to the GQ to ensure the correctness of the algorithm. It is unnecessary to check if r is w since the algorithm guarantees that the pair whose first entry is w is already stored in the GQ.

Lines 18–22: This is for the situation in which v is an NC-point but where w is not. These lines execute the code to maintain the pair whose first entry is v, an NC-point in the GQ as in lines 14–17.

Line 23: The algorithm terminates, returning the completed T.

C. CORRECTNESS OF THE PROPOSED ALGORITHM

The proposed algorithm maintains correctness by guaranteeing that the pair including each NC-point as its first entry must exist uniquely in the GQ. Should there be no pair whose first entry is either v or w in the GQ, where v and w are NC-points, then the pair (v, w) or (w, v) could not be computed in the algorithm, and the correctness would be compromised.

If an object r, which was not an NC-point previously, becomes an NC-point of the newly created cluster, then the nearest neighbors information of the other clusters can be changed. However, this does not affect the correctness of our algorithm since this information is inserted into $LQ[r]$ as it is updated, and the order in the GQ is adjusted by the algorithm.

IV. ANALYSIS

The worst-case time and space complexities of the proposed method as described in Algorithms 1–3 are $O(n^2)$ and $O(n)$, respectively, where n is the number of objects in the input. In this section, we highlight some of the key design principles and present the details of deriving the worst-case complexity of our NC-link algorithm; then, we compare it with conventional HC methods. In particular, we theoretically prove an important property of NC-link, which states that the probability of having multiple NC-points in a cluster becomes practically zero if the cluster has more than two members. This property is crucial to keep the worst-case time complexity of the NC-link algorithm quadratic in n. Finally, we define the performance metrics used for evaluating experimental results in Section V.

A. DESIGN PRINCIPLES OF NC-LINK ALGORITHM

As previously explained, NC-link does not need to update the distances between a newly merged cluster and all the other clusters after every merge, unlike most conventional

linkage methods. This can greatly accelerate the HC procedure, as will be demonstrated in Section V.

Another key principle of designing our algorithm was to keep the space usage linear in n to ensure its scalability to massive data. Some implementations of conventional HC precalculate the distances between every pair of objects and store the information in memory to achieve fast lookup; however, this approach results in limited scalability due to its quadratic space requirements. For in-memory processing of large-scale data, it is crucial for a mining algorithm to have linear space complexity. The proposed algorithm achieves this by selectively storing distance information: Only the distances of at most $(\delta \cdot n)$ pairs are stored in the priority queue, as explained in Section III-B. Note that δ ($\ll n$) is a user-specified parameter and is independent of n.

In a similar manner, we could implement a conventional HC method using the priority queues and maintaining the linear space. In this case, however, the computational cost would increase to $O(n^3)$ (*e.g.*, for the average or complete linkage), or the acceleration effect will be marginal (*e.g.*, for the centroid linkage).

B. COMPLEXITY ANALYSIS AND COMPARISON

As stated in Section II, a naïve implementation of the HC algorithm would incur an $O(n^3)$ worst-case time complexity. However, various techniques have been proposed to reduce the complexity to $O(n^2)$, as will be explained below. To facilitate further explanations, we first review the core operations involved in HC. As seen in Fig. 1(a), a typical HC algorithm consists of three steps:

- **Step 1:** The algorithm computes the pairwise distances between singleton clusters (*i.e.*, data objects). Many algorithms maintain a priority queue to store the distance information and to support rapid lookup operations performed later.
- **Step 2 (iterative):** The algorithm finds the closest pair of clusters from the list of intermediate clusters that have been found so far. The pair of clusters is merged into a new cluster, and depending on the linkage method used, some statistic (*e.g.*, for Ward's linkage, the sum of squared deviations from point to centroids) is calculated, or the representative point (*e.g.*, the centroid) of the new cluster is appointed.
- **Step 3 (iterative):** Merging two clusters in Step 2 may change the distance between some clusters. The algorithm thus updates the pairwise distances between the affected clusters, including the newly created cluster.

Note that Step 1 is performed only once, whereas Steps 2 and 3 are repeated until the clustering is completed. Certain types of linkage methods allow us to optimize the HC algorithm so that it can omit Steps 1 and 3 (*e.g.*, single and Ward's linkages; see Table 1). Recall that n and m denote the number of objects and the number of dimensions in the input data.

We start with the complexity analysis of the centroid linkage method, given its similarity to NC-link (in a sense, NC-link can be considered an enhanced version of the centroid linkage by reducing the frequency of updates of the distance information during HC). For the centroid linkage, Step 1 requires $O(n^2\ m)$ time for computing pairwise distances, and constructing a priority queue with the NN pair of each singleton demands additional $O(n)$ time (assuming the use of a binary heap and the makeHeap operation shown in Fig. 5). In Step 2, the pair of closest clusters is retrieved by the dequeue operations and can be merged if the pair is valid. The overall time complexity of Step 2 then can be denoted as $O((n-1)(\alpha+1)\log n)$, where the first factor $(n-1)$ represents the number of merges, and the remaining part indicates the maximum number of dequeue operations. In this setting, α denotes the maximum number of NN pairs invalidated by merging two clusters [23], and such pairs need to be updated to find new NNs. The upper bound of α is $2(3^m - 2)$, which depends only on m (the number of dimensions), not n (the number of objects) [23]. In Step 3, the nearest neighbors of at most $(\alpha+1)$ clusters need to be updated at every merge. In the end, the overall worst-case time complexity of the centroid linkage method becomes $O((n-1)(\alpha+1)nm)$.

In NC-link, the δ-nearest neighbors of each singleton are stored in its own priority queue during the process of computing the pairwise distance in Step 1. This takes $O(n^2\ m + n^2 \log \delta)$, where the second term $n^2 \log \delta$ comes from the priority queue operations used to find δ-NN (Algorithm 2). The cost of Step 2 is $O(\alpha n \log\ (\delta n)) + O(n^2 m)$. The first term is similar to that in the centroid linkage (the cost for local queue management is newly added). The need to find NC-points for $(n-1)$ clusters incurs the additional $O(n^2 m)$ term. The worst-case time complexity for performing Step 3 is $O((n-1)(\alpha+1)nm \log \delta)$, which is on the same order as the cost of Step 3 in the centroid linkage. In the best case, however, we can reduce the cost of Step 3 in NC-link to $O(1)$ by storing the δ-nearest neighbors of each cluster, whereas the best-case complexity of the centroid linkage for Stage 3 is $\Omega(n^2\ m)$.

We can implement the average, complete, single, and Ward's linkage methods based on the nearest-neighbor chain (NN-chain) algorithm [14]. Its worst-case time and space complexities are both $O(n^2)$, except for the single and Ward's linkage methods, each of which has a worst-case space complexity of $O(n)$. Wards's linkage completes the computation of inter-cluster distances in constant time by utilizing the information on cluster centroids and their membership [10], [14], [23], [26]. The time-optimal version of the single linkage was reported in the SLINK [20] method, which has a worst-case time and space complexity of $O(n^2)$ and $O(n)$, respectively. More details of the NN-chain algorithm can be found in [14].

Table 1 summarizes the worst-case time complexity of various linkage methods. Note that the worst-case complexity of each step of the NC-link algorithm is expressed with more terms than those of the alternatives, and the overall complexities seem to present no differences between methods. However, this only shows the theoretical worst-case complexities,

TABLE 1. Comparison of best-known worst-case time complexities of different methods.

Linkage	Overall Complexity	Complexity for each step (with constants)		
		Step 1	Step 2 (over all iterations)	Step 3 (over all iterations)
NC-link	$O(n^2)$	$O(n^2(m+\log\delta))$	$O(\alpha n\log(\delta n))+O(n^2m)$	$O(\alpha n^2 m\log\delta)$
Centroid linkage [23]	$O(n^2)$	$O(n^2m)$	$O(\alpha n\log n)$	$O(\alpha n^2 m)$
SLINK (single linkage) [20]	$O(n^2)$	–	$O(3n^2m)$	–
NN-chain (Ward's linkage) [14]	$O(n^2)$	–	$O(3n^2m)$	–
NN-chain (other linkages) [14]	$O(n^2)$	$O(n^2m)$	$O(3n^2)$	$O(n^2)$

n : the number of objects in the input data; m : the number of dimensions in the input data; δ : the number of nearest neighbors of each cluster maintained in a priority queue; α : the number of pairs invalidated after a merge

and the proposed method shows competitive performance in practice, as will be shown in Section V.

C. PROOF OF THE UNIQUENESS OF NC-POINT

Proving the uniqueness of NC-point is important for keeping the worst-case time complexity of the NC-link algorithm quadratic in n. If there are multiple NC-points for every intermediate cluster created during a run of the NC-link algorithm, it will have a cubic worst-case time complexity since each iteration now needs $O(n^2)$ operations for updating the cluster distances, not $O(n)$ as previously explained.

For a cluster with two members, it is axiomatic that the NC-point of this cluster is not unique since both of the objects are placed at the same distance from the centroid. However, for a cluster with more than two objects, we show in this section that it is implausible for the cluster to have multiple NC-points in practice.

According to measure theory [27], events that are impossible have zero probability, but the converse is not true in general. Specifically, a zero-probability event can indeed happen. This is because it is possible to have a non-empty set with zero volume, the probability being a measure of the volume. In the present context, we can paraphrase this theoretical statement as follows: It is *possible* for a cluster to have multiple NC-points, *e.g.*, if all the members of the cluster are symmetrically placed on the surface of a sphere whose center is located at the centroid of the members. However, encountering such a configuration of points from a non-trivial dataset is *implausible*, and we can claim that the cluster has a unique NC-point in practice. The following theorem formalizes this idea:

Theorem 1: For a cluster that consists of more than two real-valued vectors, the probability that the NC-point of the cluster is not unique is zero.

Proof: Consider a cluster $X=\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_k\}$, where $\mathbf{x}_i\in\mathbb{R}^m$ for $1\le i\le k$. The centroid of X is denoted by $\mu_X\in\mathbb{R}^m$. Let A denote the event whereby the cluster X has more than one NC-point. Specifically, if we let $\mathbf{x}^*=arg\,min_{\mathbf{x}\in X}||\mathbf{x}-\mu_X||$, then A indicates the event whereby $\mathbf{x}^*$ is not unique.

Let us introduce an indicator variable for the event A:

$$\mathbf{1}_A=\begin{cases}1 & \text{if event } A \text{ occurs;}\\ 0 & \text{otherwise.}\end{cases} \tag{1}$$

Then, the probability of the event A is given by

$$P(A)=\int_{\mathbf{x}_1}\cdots\int_{\mathbf{x}_k}\mathbf{1}_A P(\mathbf{x}_1,\ldots,\mathbf{x}_k)d\mathbf{x}_1\cdots d\mathbf{x}_k \tag{2}$$

$$=\int_{\mathbf{x}_1}P(\mathbf{x}_1)\int_{\mathbf{x}_2}P(\mathbf{x}_2|\mathbf{x}_1)\cdots \times\int_{\mathbf{x}_k}\mathbf{1}_A P(\mathbf{x}_k|\mathbf{x}_1,\ldots,\mathbf{x}_{k-1})d\mathbf{x}_k\cdots d\mathbf{x}_1 \tag{3}$$

It is clear that $P(A)$ will be zero if the innermost part of the integration (Eq. 3), namely,

$$\int_{\mathbf{x}_k}\mathbf{1}_A P(\mathbf{x}_k|\mathbf{x}_1,\ldots,\mathbf{x}_{k-1})d\mathbf{x}_k \tag{4}$$

is zero for any $\mathbf{x}_1,\ldots,\mathbf{x}_{k-1}$ since the following always holds:

$$0\le P(A)\le\max_{\mathbf{x}_1,\ldots,\mathbf{x}_{k-1}}\int_{\mathbf{x}_k}\mathbf{1}_A P(\mathbf{x}_k|\mathbf{x}_1,\ldots,\mathbf{x}_{k-1})d\mathbf{x}_k. \tag{5}$$

Consequently, to prove that $P(A)$ is zero, it suffices to show that the quantity in Eq. 4 equals zero. In this regard, suppose that we have fixed the $k-1$ points $\{\mathbf{x}_1,\ldots,\mathbf{x}_{k-1}\}\triangleq X'$ and that we are about to add the last k-th point $\mathbf{x}_k$ to complete the cluster X, as informally shown in Fig. 6. We can then interpret the integration in Eq. 4 as computing the probability of placing $\mathbf{x}_k$ such that the event A occurs.

Note that μ_X is affine with respect to $\mathbf{x}_k$:

$$\mu_X=\frac{\mathbf{x}_1+\cdots+\mathbf{x}_k}{k}=\frac{k-1}{k}\frac{\mathbf{x}_1+\cdots+\mathbf{x}_{k-1}}{k-1}+\frac{\mathbf{x}_k}{k} \tag{6}$$

$$=\frac{k-1}{k}\mu_{X'}+\frac{\mathbf{x}_k}{k} \tag{7}$$

where $\mu_{X'}$ is the centroid of the partial cluster X'. Based on this fact, the proof will focus on exploring the possible location of μ_X (in lieu of $\mathbf{x}_k$) in $\mathbb{R}^m$ and estimating the associated probability of the event A.

If the event A occurs, then the point represented by μ_X will have multiple nearest neighbors. The point $\mathbf{x}_k$ may or may not be one of these nearest neighbors. We can thus represent the probability of A by the sum of the probabilities of two disjoint events A_{in} and A_{out} as follows:

$$P(A)=P(A_{\text{in}})+P(A_{\text{out}}) \tag{8}$$

where the event A_{in} represents the case in which $\mathbf{x}_k$ is one of the nearest neighbors of μ_X and the event A_{out} represents the case in which $\mathbf{x}_k$ is not one of these nearest neighbors.

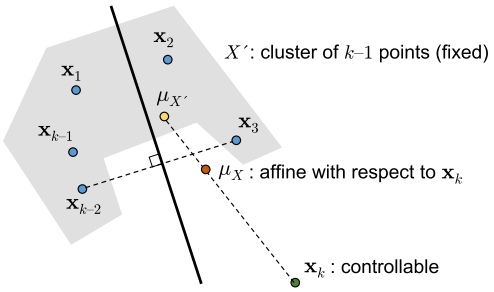


FIGURE 6. Illustration of the proof idea. For a cluster $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, the centroid of X is denoted by μ_X. Assume that X', a partial cluster of $(k-1)$ points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k-1}\}$, is given (fixed) and that we can only control the location of $\mathbf{x}_k$. Since μ_X is affine with respect to $\mathbf{x}_k$, as explained in the main text, we instead focus on exploring the possible locations of μ_X. In order for the event A whereby X has multiple NC-points to occur, μ_X must lie on one of the $\binom{k-1}{2}$ lines determined by $\mathbf{x}_i$ and $\mathbf{x}_j$ for $i \neq j, i \neq k, j \neq k$ (an example of such a line is shown in the figure) or on one of the $k-1$ circles of Apollonius defined by $\mu_{X'}$ and $\mathbf{x}_i$ for $1 \leq i < k$. The proof in the main text shows that the event A corresponds to a set that has measure zero, implying that X will have a unique NC-point in practice.

In the following, we prove that $P(A) = 0$ by showing that $P(A_{\text{in}}) = P(A_{\text{out}}) = 0$.

We first prove that $P(A_{\text{out}}) = 0$. Assume that the centroid μ_X has two nearest points, $\mathbf{x}_i$ and $\mathbf{x}_j$ (note that $i \neq j$, $i \neq k$, and $j \neq k$). Let $\Psi_{i,j}$ denote the hyperplane (in $\mathbb{R}^m$) composed of the points that have the same distance to $\mathbf{x}_i$ and $\mathbf{x}_j$, *i.e.*,

$$\Psi_{i,j} = \left\{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^m \text{ and } ||\mathbf{x} - \mathbf{x}_i|| = ||\mathbf{x} - \mathbf{x}_j||\right\}. \tag{9}$$

Then, the probability of A_{out} must be upper bounded by the probability of the union of the following events: $\mu_X \in \Psi_{i,j}$ (for $1 \leq i < k, 1 \leq j < k, i \neq j$). Specifically,

$$P(A_{\text{out}}) < \sum_{i,j=1, i\neq j}^{k-1} P(\mu_X \in \Psi_{i,j}). \tag{10}$$

Note in Eq. 9 that the hyperplane corresponds to a set that has measure zero [27] and zero probability. Since the summation in Eq. 10 is countable, the upper bound of $P(A_{\text{out}})$ given in Eq. 10 must be zero, and thus, $P(A_{\text{out}}) = 0$.

Next, we show that $P(A_{\text{in}}) = 0$. In this case, $\mathbf{x}_k$ is assumed to be an NC-point of the cluster X. In order for the event A_{in} to occur, there must exist $\mathbf{x}_i \in X$ such that

$$||\mu_X - \mathbf{x}_k|| = ||\mu_X - \mathbf{x}_i|| \tag{11}$$

for $1 \leq i < k$. Given that

$$\mu_X - \mu_{X'} = \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_k}{k} - \frac{\mathbf{x}_1 + \cdots + \mathbf{x}_{k-1}}{k-1} \tag{12}$$

$$= \frac{1}{k-1}(\mathbf{x}_k - \mu_X), \tag{13}$$

we can rewrite Eq. 11 as

$$(k-1)||\mu_X - \mu_{X'}|| = ||\mu_X - \mathbf{x}_i|| \tag{14}$$

and define Φ_i, a set of the points in $\mathbb{R}^m$ that are an identical distance to $\mathbf{x}_i \in X'$ and the fixed point $\mu_{X'}$, as

$$\Phi_i = \left\{\mathbf{x} \mid \mathbf{x} \in \mathbb{R}^m \text{ and } ||\mathbf{x} - \mu_{X'}|| = \frac{||\mathbf{x} - \mathbf{x}_i||}{k-1}\right\} \tag{15}$$

for $1 \leq i \leq k-1$. Then, the probability of A_{in} must be upper bounded by the probability of the union of the events $\mu_X \in \Phi_i$, *i.e.*,

$$P(A_{\text{in}}) < \sum_{i=1}^{k-1} P(\mu_X \in \Phi_i). \tag{16}$$

Notice that the set Φ_i describes an Apollonius circle [28] in $\mathbb{R}^m$ since the ratio $||\mathbf{x} - \mu_{X'}||/||\mathbf{x} - \mathbf{x}_i||$ is fixed. The set Φ_i is thus a sub-hyperplane in $\mathbb{R}^m$ and has measure zero. Since the summation in Eq. 16 is countable, the upper bound of $P(A_{\text{in}})$ must be zero; we can thus conclude that $P(A_{\text{in}}) = 0$.

Since both of the probabilities $P(A_{\text{in}})$ and $P(A_{\text{out}})$ are zero, we have proved that $P(A) = 0$. ■

D. PERFORMANCE METRICS

To facilitate our analysis of the experimental results in Section V, we define two performance metrics, both of which are counters that count how many times we need to update the distance information during HC. More specifically, in NC-link, these metrics count how many times we perform δ-NN search. Depending on the reason for initiating a search, we categorize these update counters as follows (see Section VI-A for more discussion on these performance metrics):

- U_{new} : The number of δ-NN searches that occur since the NC-point of a newly created cluster is neither of the NC-points of the two merged clusters.
- U_{empty} The number of δ-NN searches started because the queue is empty.

Note that the events that increase any of these counters are disjoint, and we define the total count U_{total} as follows:

$$U_{\text{total}} = U_{\text{new}} + U_{\text{empty}}. \tag{17}$$

The value of U_{total} for a run of HC depends on the value of δ, and adjusting δ can achieve a significant gain in efficiency, as demonstrated in Fig. 7 and Fig. 8.

V. EXPERIMENTAL RESULTS

A. SETUP

We tested the proposed method with the data sets listed in Table 2. Although NC-link is not limited to certain data types, we mostly used biological data for our experiments. HC is one of the most popular clustering methods in biomedicine. Biological data sets are typically not sparse, and algorithms that are optimized for sparse data should show poor performance. In addition, there are several biological data repositories in which a large amount of large-scale data is freely available.

The data sets used in our experiments can be classified as follows: First, we downloaded 20 gene expression data sets (DT1–20) from the *Stanford Microarray Database* (SMD) [29]. On average, these data sets have 40,957 objects with 34 dimensions. Second, we utilized the three-dimensional protein structural data (DT21) produced by [30]. This data set is a collection of FEATURE vectors,

TABLE 2. Datasets used for our experiments.

ID	Description	# objects	# dim.
DT1	Ebola-infected non-human primates	36,794	65
DT2	Pediatric acute lymphoblastic leukemia	39,328	32
DT3	Erythropoietin use in kidney transplants	36,789	13
DT4	Human universal reference RNA	42,845	10
DT5	Epithelial-mesenchymal interaction	41,421	95
DT6	Kawasaki disease samples	36,794	63
DT7	Diced siRNAs for RNAi (p38a MAPK)	41,421	54
DT8	Immune thrombocytopenic purpura	41,421	53
DT9	Homozygous deletions/pancreatic cancer	41,421	48
DT10	Interleukin 6 in myocardial dysfunction	36,410	42
DT11	Anatomical compartments of the eye	42,762	34
DT12	Malignant potential/serous carcinomas	42,845	29
DT13	Germ cell tumors	42,845	23
DT14	Toxoplasma bradyzoites	41,421	20
DT15	HeLa S3, thymidine-nocodazole block	42,746	19
DT16	CNAs in 89 breast tumors	41,805	18
DT17	Prostate cancer genes	41,421	18
DT18	Anti-inflammatory effect of colchicine	41,421	16
DT19	Intrinsic androgen-dependent genes	41,421	14
DT20	mRNA targets for PUF-family proteins	41,805	12
DT21	Protein microenvironment	100,000†	264
DT22	Pfam, a collection of protein families	1,000,000	265

† The original data set has 1,992,567 vectors; 100,000 vectors were randomly chosen.

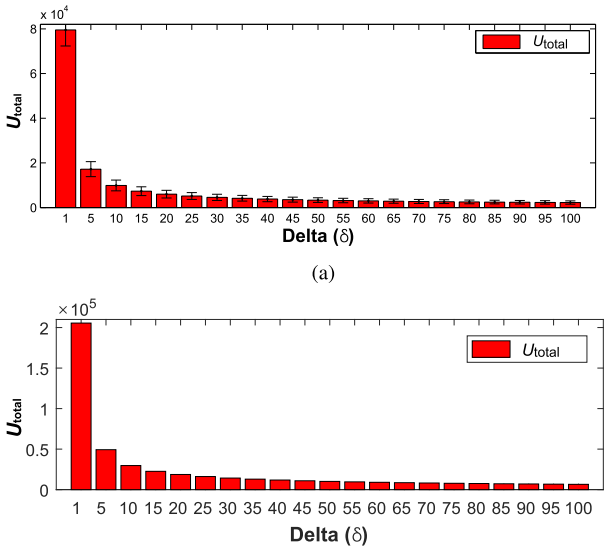


FIGURE 7. The effect of δ on U_{total}. (a) DT1–20 (avg. obj. = 40957; avg. dim. = 34). (b) DT21 (100, 000 × 264).

which can characterize local environments around protein residues by counting physicochemical properties within concentric shells around a center. Third, we used the Pfam repository [31] to generate a million-object dataset (DT22) to be used for scalability tests. Finally, although not shown in Table 2, we synthesized some matrices of random integers to use to generate baseline results in some of the experiments (see Section V-E.1).

We implemented the proposed NC-link method in C++, compiled it using Intel ICC (version 16.0), and conducted experiments on a 2 GHz Linux machine (Intel E5-2650, CentOS 6.6) with 128 GB of main memory without any

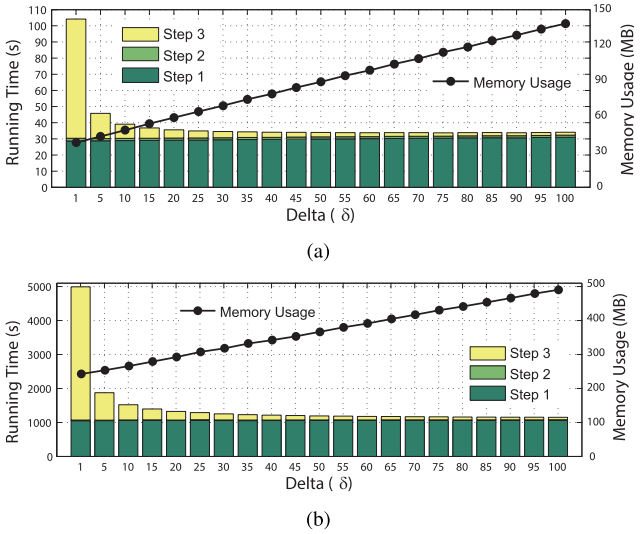


FIGURE 8. The effect of δ on running time and memory usage. (a) DT1–20 (avg. obj. = 40957; avg. dim. = 34). (b) DT21 (100, 000 × 264).

parallelization but with the user-mandated vectorization supported by the Intel architecture. For comparison, we used Fastcluster [26], which includes NN-chain-based implementations of the average, complete, and Ward's linkage methods. For the single linkage method, we utilized the SLINK program included in Cluster 3.0 [32]. The centroid linkage was implemented using the NC-link implementation. To ensure fair comparison, we revised all the above alternative methods so that they can also benefit from the user-mandated parallelization in the same manner as NC-link.

B. EFFECTS OF THE δ PARAMETER

As described in Section III, the NC-link algorithm takes a user-defined parameter denoted by δ. This parameter specifies how much space should be allocated for clustering—out of $\binom{n}{2}$ possibilities, only the distances of the $(\delta \cdot n)$ closest pairs are stored. Recall that the value of δ is independent of accuracy and only affects the running time and space requirements of the algorithm. Here, we analyze the effect of δ.

Fig. 7 shows how U_{total} varies with different values of δ. For both types of datasets, U_{total} drastically decreased from $\delta = 1$ to $\delta = 5$. For larger values of δ, the decrease in U_{total} was negligible. This is a positive observation and implies that we do not need to use large values of δ (which would incur massive use of memory) to reduce U_{total}, which will eventually accelerate the whole run.

More specifically, for the DT1–20 datasets, the average U_{total} was approximately 79, 000 for $\delta = 1$ but decreased to approximately 19, 000 for $\delta = 10$ and to 2, 236 for $\delta = 100$. The value of U_{new} averaged over DT1–20 was 754.65 and did not vary with different values of δ, as expected from the fact that U_{new} is independent of the value of δ. We made similar observations from the DT21 data. Most of U_{total} was due to U_{empty} when $\delta = 1$, and U_{empty} quickly decreased for larger values of δ, which justifies our idea of utilizing

δ-NN information to reduce the number of updates on distance information during HC.

Note that the instance of NC-link with $\delta = 1$ corresponds to the case where the number of updates cannot be reduced because the extra nearest neighbors are not stored in the local queues (*e.g.*, HC using the centroid linkage). We will use this instance with $\delta = 1$ as the baseline for comparison in this section.

To confirm the effect of reducing U_{total} on the running time, we measured the running time and memory usage with the same values of δ, as in Fig. 7. We measured the running time of each of the three HC steps (see Section IV-B) separately so that we could analyze the effect of δ more closely.

As shown in Fig. 8, the running time decreased in the same manner as U_{total} as we increased δ. In particular, the running time of Step 3 was most significantly affected by δ. This is because the value of δ controls the frequency of the δ-NN search in Step 3, whereas Steps 1 and 2 are not significantly dependent on δ. As expected, the space demand grew linearly as we increased δ, but the slope of the line was moderate: For both data types (DT1-20 and DT21), amplifying δ 100 times (going from $\delta = 1$ to $\delta = 100$) only increased the memory usage by less than 3.5 times.

Based on the results shown in Fig. 8, we determined the values of δ that would be used for the other experiments. For DT1–20, $\delta = 75$ gave the shortest average running time, whereas $\delta = 10$ achieved the best balance[3] between time and space requirements. For DT21, $\delta = 100$ and $\delta = 20$ resulted in the best running time and balance, respectively. We thus used these values in the other experiments presented in this section.

C. COMPARISON WITH CENTROID LINKAGE

In Fig. 9, we compare NC-link with the centroid linkage in terms of the running time, memory usage, U_{new}, and U_{total} values measured over DT1-21. Given that NC-link can be considered as an improved version of the centroid, our aim was to identify where and how the performance gain of NC-link over the centroid was achieved.

In Step 1, NC-link ran slightly slower than the centroid linkage due to the need for storing more (δ vs one) elements in the queue (this extra cost is represented by the $(O \log \delta)$ term in Table 1). In Step 2, the running time of both methods was similar. In Step 3, we observed substantial decreases in running time for NC-link. Leveraged by the performance advantages in Step 3, NC-link significantly outperformed the centroid linkage in terms of overall running time. This expedited runtime can be accredited to the decreased U_{new} and U_{total} for NC-link. In particular, U_{new} was negligible for NC-link, whereas it accounted for a significant portion of U_{total} for the centroid.

The price for the speedup of NC-link over the centroid was the increased memory usage, but the level of the extra memory required was reasonable, especially for the high-dimensional DT21 data. Using 1.28–3.4 times more memory, NC-link reduced the running time by 2.05–3.54 times (overall) and by 6.66-42.08 times (Step 3).

[3]measured as $\frac{\text{speedup (decrease ratio in runtime)}}{\text{increase ratio in memory usage}}$ with respect to the $\delta = 1$ case

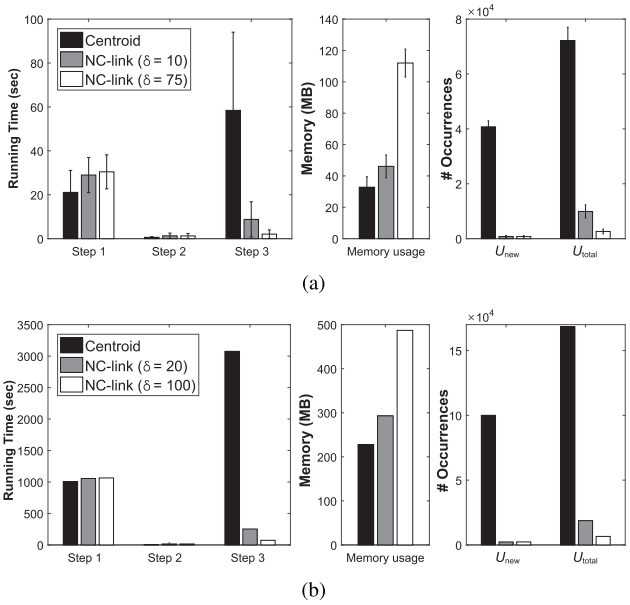


FIGURE 9. Comparison: NC-link and centroid linkage. (a) DT1–20 (avg. obj. = 40957; avg. dim. = 34). (b) DT21 (100, 000 × 264).

D. SPEED AND SCALABILITY

We compared the efficiency of NC-link with the alternative methods in terms of running time, memory usage, and time-space trade-off.

Fig. 10 shows the running time of different methods. For DT1-20, the NC-link instances with $\delta = 75$ were competitive, showing similar running time to the single linkage, which is known to be the fastest HC method. For DT21, four methods (NC-link, average, complete, and single linkage methods) outperformed the centroid and Ward's linkage. The result from DT22 shown in Fig. 10(c) is to demonstrate the ability of NC-link to handle million-scale data on a PC. For DT1-21, the running time of the average and complete linkage methods is similar to that of NC-link, but their space complexity is quadratic, as is the case for the time complexity (reducing the space complexity of these two linkages is possible, but the time complexity increases to cubic), limiting their scalability.

To further analyze this space-time trade-off, Fig. 11 shows the running time (the same information as shown in Fig. 10(b) and (c)) and the memory usage simultaneously. As previously stated, the average and complete linkages are fast but require a large amount of memory. The centroid and Ward's linkages are space efficient, but their running time is unsatisfactory.

In the same line of analysis, Fig. 12 shows the values of 1/(normalized time · normalized space) that can be used to quantify the time-space balance. We can more clearly observe the limitations of the average and complete linkages. Only the proposed NC-link and the single linkage gives competitive

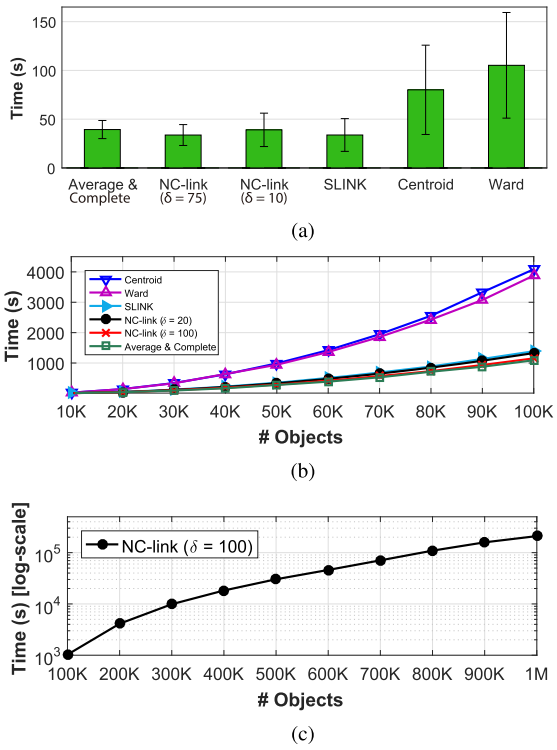


FIGURE 10. Running time comparison. (a) DT1–20 (avg. obj. = 40957; avg. dim. = 34). (b) DT21 (100, 000 × 264). (c) DT22 (1, 000, 000 × 265).

TABLE 3. Comparison of cophenetic correlation coefficients (CPCCs) [3] for different methods and data.

Data	**NC-link**	CE	A	CO	S	W
DT1–20†	**0.9373**	0.9427	0.9436	0.9259	0.9092	0.5183
DT21	**0.9711**	0.9796	0.9733	0.9477	0.9606	0.6896
Random†,♮	0.1484	0.2947	0.1701	0.0504	0.3231	0.0667

Abbreviations: A, Average linkage; CO, Complete linkage; CE, Centroid linkage; S, Single linkage; W, Ward's linkage
† Average value; ♮21 datasets randomly sampled from uniform distribution

results in terms of both time and space. However, as will be shown shortly (see Table III and Fig. 14), it is well known that the single linkage often suffers from the chaining problem, which often limits its practical applicability.

Taken together, our experimental results suggest that NC-link can outperform all the alternatives in terms of speed, memory usage, and quality of clustering results. To convince the readers of the third advantage of NC-link (clustering quality), we proceed to Section V-E.

E. EVALUATING QUALITY OF CLUSTERING RESULTS

To evaluate how well NC-link fits input data and retrieves the hierarchical structures therein in comparison to conventional methods, we carried out three types of experiments.

1) COPHENETIC CORRELATION COEFFICIENT

First, we used the cophenetic correlation coefficient (CPCC), a standard measure of how well a hierarchical cluster fits input data [3]. For two objects, their cophenetic distance is

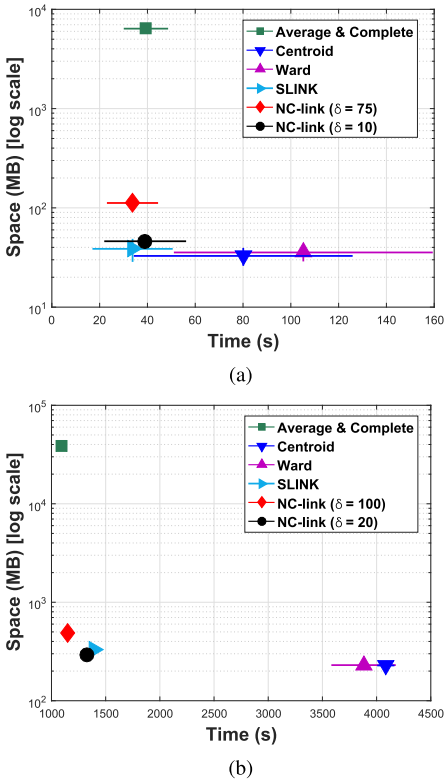


FIGURE 11. Simultaneous comparison of running time and memory usage. (a) DT1–20 (avg. obj. = 40957; avg. dim. = 34). (b) DT21 (100, 000 × 264).

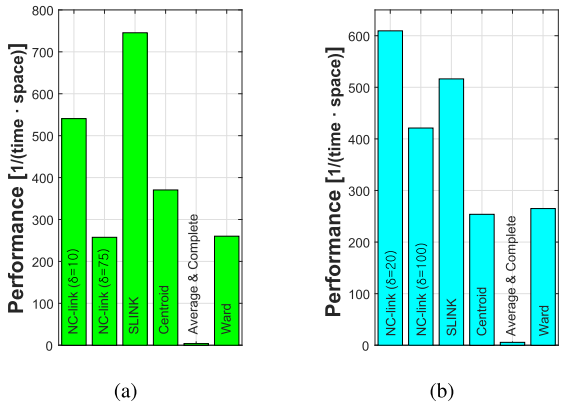


FIGURE 12. Quantitative comparison of the time-space trade off shown in Fig. 11 using a metric defined as 1/(normalizedtime · normalizedspace). (a) DT1–20. (b) DT21.

defined as the distance at which they are grouped together for the first time during HC. The CPCC then corresponds to the correlation between a cophenetic distance matrix and a pairwise distance matrix. The higher the quality of a clustering method, the closer to 1 the magnitude of this value.

Table 3 lists the CPCC values calculated from the clustering results obtained by running six different linkage methods on DT1–21 and 21 random datasets whose sizes match those of DT1–21 (the elements of the random matrices were randomly selected from a uniform distribution). As random data does not contain structure, the score obtained for the random

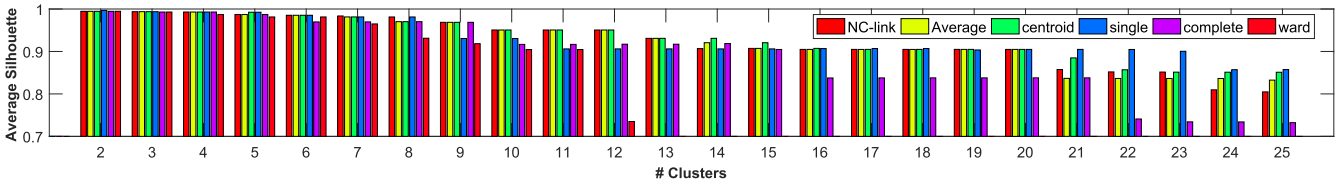


FIGURE 13. The average silhouette value of the flat clusters obtained from the dendrograms by decreasing cutoff heights (data: DT21).

data can serve as the baseline score to which the scores shown can be compared.

In this experiment, the values from NC-link were as high as those from the average and centroid linkage methods, indicating that NC-link fits the input data as faithfully as the average and centroid linkages do. It is intriguing to observe that the single linkage method gave the highest value for the random data, although it was not noticeably effective for the real data. The chaining effect (the main disadvantage of the single linkage) may have caused false patterns to be captured.

2) SILHOUETTE VALUE

Given a clustering result, we can use the silhouette method to measure how similar an object is to the objects in its own cluster (*i.e.*, cohesion) in comparison with those in other clusters (*i.e.*, separation) [18]. The silhouette value ranges between -1 and 1. One silhouette values is computed for each object, and if many objects in a clustering result present high values, then the clustering method used is considered appropriate. In practice, clusters with an average silhouette value of 0.71 or higher are considered successful in finding strong structures in the input data [33].

The dendrogram of the HC algorithm returns does not directly represent a group of clusters. To generate flat clusters from a dendrogram, we can conceptually cut it with a horizontal line [14, Fig. 1]. The height of the line determines the (dis)similarity threshold.

In our experiment, we gradually lowered the cutoff line from top to bottom, generating increasingly clusters, until the average silhouette value of the generated clusters remained above 0.71. In this way, we generated 46 (''strongly structured'') clusters from the dendrogram produced when running NC-link on DT21. In the same setting, the alternative methods produced fewer clusters: 42 (average), 28 (complete), 36 (single), 12 (Ward's), and 44 (centroid) clusters.

Fig. 13 shows how the average silhouette value of the flat clusters changes for decreasing cutoff heights (due to space limitations, we only show cases with 2- to 25-clusters). The NC-link method showed a similar tendency for the silhouette value under the average and centroid linkage methods. This suggests that using NC-link in lieu of the average or centroid linkage will not prevent us from discovering important patterns.

3) MEASURE OF CONCORDANCE (MoC)

The measure of concordance (MoC) [19] is used to compare two clustering results quantitatively. The value of MoC ranges between 0 and 1 (a value closer to 1 indicates two clustering results that are more similar). Based on the flat clusters produced from the experiments presented in Section V-E.2, we compared the clusters found by NC-link with those found by the alternative methods, therein measuring the MOC value for each pair of clustering results. Note that there are $\binom{6}{2} = 15$ combinations since we compared NC-link with five different methods.[4] This experiment should be complementary to the experiment in Section V-E.2 in that the latter is regarding the quality of individual methods, whereas the former is regarding the similarity of results between different methods.

Fig. 14 shows the result, where each box plot was generated from a pair of clustering results by varying the number of clusters from 2 to 50 (Fig. 14(a): DT1–20 and Fig. 14(b): DT21). In terms of the median MoC values, NC-link gave the most similar results to the centroid and average linkages in both of the plots. The similarity between the Ward's linkage and the other methods was the lowest. Although the similarity between the results from the single linkage and those from the other methods (except the Ward's linkage) remained reasonably high, outliers were consistently observed for the comparisons involving the single linkage, which is clearly seen in Fig. 14(b). This suggests the existence of odd-structured clusters in the results from the single linkage, probably due to the chaining problem that frequently occurs in the single linkage method.

VI. DISCUSSION

A. COMPARISON WITH CENTROID LINKAGE

Among the three steps of HC explained in Section IV-B, a key contribution of NC-link lies in making Step 3 efficient by introducing the concept of NC-point and managing δ-nearest neighbors of intermediate clusters. As shown through the various experimental results in Section V, this scheme allows the user to achieve a trade-off between time and memory, and typically, we can gain a significant reduction in running time with only a moderate increase in memory usage.

Note that $0 \leq U_{\text{new}} \leq n-1$ for NC-link, whereas $U_{\text{new}} = n-1$ for the centroid linkage. For the centroid linkage, the centroid of a new cluster is in general different from any of the centroids of the two clusters that have been merged, and there is no point in retaining more than one NN in the queue. This makes every merge trigger an NN search. By contrast, the NC-point of a new cluster often coincides with either

[4] As a baseline, we also tested a dummy clustering algorithm that randomly clusters data. The MoC values for this dummy method and all other methods were nearly zero, and we did not include this baseline result in the plot.

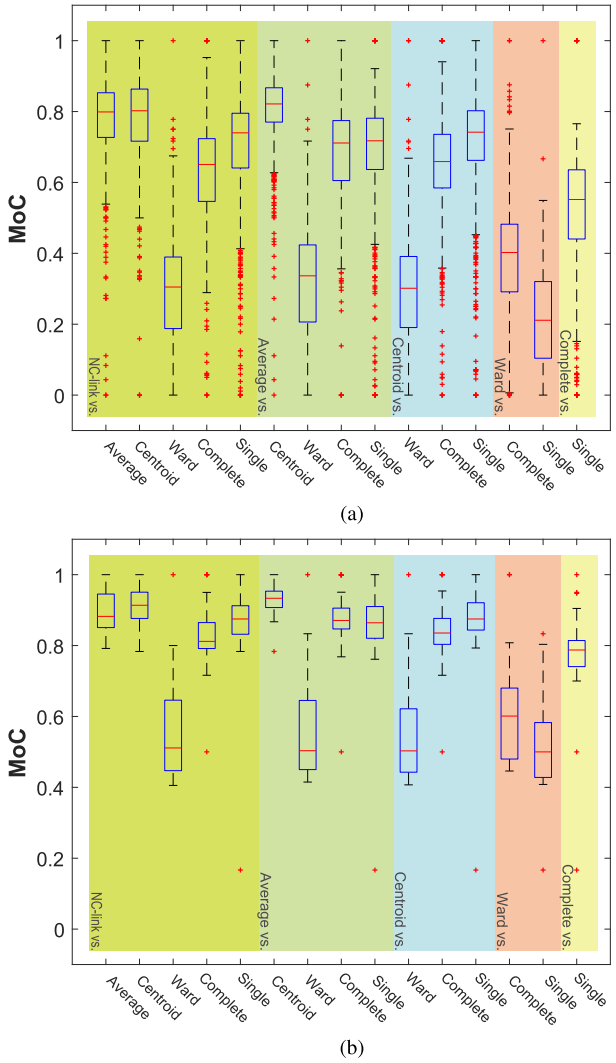


FIGURE 14. The measure of concordance (MoC). There are $\binom{6}{2} = 15$ combinations since we compared NC-link with five different methods. (a) DT1–20. (b) DT21.

NC-point of the two merged clusters, and storing multiple NNs in advance effectively reduces the number of NN searches.

B. OPTIMIZING δ-NEAREST NEIGHBOR SEARCH

One of the key ideas behind the NC-link algorithm is the use of the δ-nearest neighbors of a cluster to reduce the number of clusters that requires updates of the distance information after a merge in each iteration. This search scheme corresponds to an instance of the k-nearest neighbor (k-NN) search problem, for which various effective solution methods exists [34]–[44].

In the present version of the NC-link algorithm, we use a simple linear search for finding the δ-NNs, which results in an $O(n^2)$ total time complexity for all iterations of HC. For moderate values of m (the number of dimensions), we will be able to accelerate the δ-NN search by employing one of the existing k-NN search methods. For example, we can construct a k–d tree of n elements in $O(n \log n)$ average time and perform a k-NN search in $O(k \log n)$ average time, which will lead to an improved $O(kn \log n)$ time complexity for all iterations. However, using a k-d tree becomes inefficient as the number of dimensions increases. As a general rule, we need to have $n \gg 2^m$ for a m-dimensional k-d tree to deliver better performance compared to exhaustive search [38].

For high-dimensional data, we may thus have to use other techniques, including [37]–[39]. For instance, we may devise an algorithm using a ball tree [37], which often shows satisfactory performance for high-dimensional data. The k-NN search operation on the ball tree normally runs in $O(m \log n)$, which will allow the NC-link algorithm to have subquadratic complexity. However, if the value of m becomes extremely large, the ball-tree-based algorithm will not maintain its efficiency, thus losing its performance edge over the linear search.

If we can sacrifice exactness for efficiency, we may employ approximation algorithms for k-NN search [40]–[44]. Replacing the current algorithm for δ-NN search with an approximation algorithm will be a part of our future work, and it remains to be seen how effective approximate nearest neighbors will be in the context of HC.

C. EXPEDITING HIERARCHICAL CLUSTERING

In various applications, HC presents advantages over alternative clustering techniques because of its ability to discover hierarchical patterns globally and summarize them in a format that is convenient for downstream analyses. However, such a property of HC poses a challenge for parallelization due to the difficulty in handling global dependence from one iteration to another [14]. Distributed implementations of HC algorithms are relatively rare, and most methods to accelerate HC thus focus on either devising approximate HC algorithms or parallelizing the pairwise distance calculation, which presents abundant opportunity for the single instruction multiple data (SIMD) type of parallelization.

As an alternative solution to the challenge of accelerating HC by parallelization, we propose to enhance the linkage method, which plays a key role in HC. All linkage methods have been around for a while, and to the best of the authors' knowledge, there has been no recent attempt to devise a new linkage method, particularly with the purpose of expediting HC. As demonstrated by our experimental results, the proposed NC-link suggests an effective alternative to the conventional linkages for accelerating HC.

VII. CONCLUSION

We have proposed a new linkage method called NC-link and an efficient implementation of HC based on this linkage. According to our experimental results, the proposed methodology was effective and often outperformed alternative methods in terms of the quality of the clustering results and the time and space efficiency. We believe that we can further enhance the efficiency of the proposed algorithm by expediting its components using optimized neighbor search schemes.

Given the effectiveness of our approach, we anticipate that it will make an attractive tool for researchers who want a scalable and robust solution to large-scale clustering.

REFERENCES

[1] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, no. 3, pp. 264–323, 1999.
[2] J. Han and M. Kamber, *Data Mining*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.
[3] P. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Boston, MA, USA: Addison-Wesley, 2005.
[4] D. Hand, H. Mannila, and P. Smyth, *Principles Data Mining*. Cambridge, MA, USA: Bradford Book, 2001.
[5] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco, CA, USA: Morgan Kaufmann, 2005.
[6] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press, 1999.
[7] S. C. Madeira and A. L. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. 1, no. 1, pp. 24–45, Jan./Mar. 2004.
[8] H.-P. Kriegel and P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discovery From Data*, vol. 3, no. 1, pp. 1–58, 2009.
[9] L. Liu and M. T. Zsu, *Encyclopedia of Database Systems*. San Francisco, CA, USA: Springer, 2009.
[10] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *Comput. J.*, vol. 26, no. 4, pp. 354–359, 1983.
[11] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Comput.*, vol. 21, no. 8, pp. 1313–1325, 1995.
[12] S. Rajasekaran, "Efficient parallel hierarchical clustering algorithms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 497–502, Jun. 2005.
[13] M. Kull and J. Vilo, "Fast approximate hierarchical clustering using similarity heuristics," *BioData mining*, vol. 1, no. 1, p. 1, 2008.
[14] Y. Jeon and S. Yoon, "Multi-threaded hierarchical clustering by parallel nearest-neighbor chaining," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 9, pp. 2534–2548, Sep. 2015.
[15] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," in *Proc. ACM SIGMOD*, 1996, pp. 103–114.
[16] S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," in *Proc. ICDE*, 1999, pp. 512–521.
[17] G. Karypis, E.-H. Han, and V. Kumar, "Chameleon: Hierarchical clustering using dynamic modeling," *Computer*, vol. 32, no. 8, pp. 68–75, Aug. 1999.
[18] P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, no. 1, pp. 53–65, 1987.
[19] D. Pfitzner, R. Leibbrandt, and D. Powers, "Characterization and evaluation of similarity measures for pairs of clusterings," *Knowl. Inf. Syst.*, vol. 19, no. 3, pp. 361–394, 2009.
[20] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *Comput. J.*, vol. 16, no. 1, pp. 30–34, 1973.
[21] M. Easter, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.
[22] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering points to identify the clustering structure," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 226–231.
[23] W. H. E. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *J. Classification*, vol. 1, no. 1, pp. 7–24, 1984.
[24] F. Murtagh, "Multidimensional clustering algorithms," *Compstat Lectures*, vol. 1. Vienna, Austria: Physika Verlag, 1985.
[25] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA, USA: MIT Press, 2001.
[26] D. Müllner, "fastcluster: Fast hierarchical, agglomerative clustering routines for r and python," *J. Statist. Softw.*, vol. 53, no. 9, pp. 1–18, 2013.
[27] T. Tao, *An Introduction to Measure Theory*. Providence, RI, USA: American Mathematical Society, 2011, vol. 126.
[28] D. Pedoe, *Geometry: A Comprehensive Course*. North Chelmsford, MA, USA: Courier Corp., 2013.
[29] G. Sherlock *et al.*, "The stanford microarray database," *Nucl. Acids Res.*, vol. 29, no. 1, pp. 152–155, 2001.
[30] S. Yoon, J. C. Ebert, E.-Y. Chung, G. D. Micheli, and R. B. Altman, "Clustering protein environments for function prediction: Finding PROSITE motifs in 3D," *BMC Bioinf.*, vol. 8, p. S10, May 2007.
[31] A. Bateman *et al.*, "The pfam protein families database," *Nucl. Acids Res.*, vol. 32, no. 1, pp. D138–D141, 2004.
[32] M. de Hoon, S. Imoto, J. Nolan, and S. Miyano, "Open source clustering software," *Bioinformatics*, vol. 20, no. 9, pp. 1453–1454, 2004.
[33] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, vol. 344. Hoboken, NJ, USA: Wiley, 2009,
[34] Y.-C. Liaw, M.-L. Leou, and C.-M. Wu, "Fast exact k nearest neighbors search using an orthogonal search tree," *Pattern Recognit.*, vol. 43, no. 6, pp. 2351–2358, Jun. 2010.
[35] J. McNames, "A fast nearest-neighbor algorithm based on a principal axis search tree," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 9, pp. 964–976, Sep. 2001.
[36] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, no. 7, pp. 750–753, Jul. 1975.
[37] T. Liu, A. W. Moore, and A. Gray, "New algorithms for efficient high-dimensional nonparametric classification," *J. Mach. Learn. Res.*, vol. 7, pp. 1135–1158, Jun. 2006.
[38] S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 9, pp. 989–1003, Sep. 1997.
[39] X. Wang, "A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality," in *Proc. IJCNN*, Jul. 2011, pp. 1293–1299.
[40] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 13th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.
[41] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," *SIAM J. Comput.*, vol. 30, no. 2, pp. 457–474, 2000.
[42] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci. (FOCS)*, Oct. 2006, pp. 459–468.
[43] C.-C. Chang and T.-C. Wu, "A hashing-oriented nearest neighbor searching scheme," *Pattern Recognit. Lett.*, vol. 14, no. 8, pp. 625–630, 1993.
[44] J. Gao, H. V. Jagadish, W. Lu, and B. C. Ooi, "DSH: Data sensitive hashing for high-dimensional k-nnsearch," in *Proc. ACM SIGMOD*, 2014, pp. 1127–1138.

YONGKWEON JEON received the B.S. and M.S. degrees in electrical engineering from Korea University, South Korea, in 2009 and 2013, respectively. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Seoul National University, South Korea. His research interests include the design of parallel machine learning algorithms and computer systems for intelligence.

JAEYOON YOO received the B.S. degree in electrical and computer engineering from Seoul National University, South Korea, in 2016, where he is currently pursuing the M.S. and Ph.D. degrees in electrical and computer engineering. His research interests include large-scale machine learning.

JONGSUN LEE received the B.S. degree in electrical and computer engineering from Seoul National University, South Korea, in 2015, where he is currently pursuing the Ph.D. degree in electrical and computer engineering. His research interests include time-series analysis, dynamic graph analysis, and clustering.

SUNGROH YOON (S'99–M'06–SM'11) received the B.S. degree in electrical engineering from Seoul National University, South Korea, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 2002 and 2006, respectively. From 2006 to 2007, he was with Intel Corporation, Santa Clara, CA. He held research positions with Stanford University and Synopsys, Inc., Mountain View, CA. He was an Assistant Professor with the School of Electrical Engineering, Korea University, from 2007 to 2012. Since 2016, he has been a Visiting Professor with the Department of Neurology and Neurological Sciences, Stanford University. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, Seoul National University. His research interests include data-driven artificial intelligence, deep learning, machine learning, and large-scale applications, including bioinformatics. He was a recipient of the 2013 IEEE/IEEK Joint Award for Young IT Engineers.