Shared Nearest Neighbor Clustering in a Locality Sensitive Hashing Framework

Sawsan Kanj, Thomas Bruls, Stéphane Gazut

Shared Nearest Neighbor Clustering in a Locality Sensitive Hashing Framework

SAWSAN KANJ,[1–5] THOMAS BRÜLS,[1,3–5] and STÉPHANE GAZUT[2]

ABSTRACT

We present a new algorithm to cluster high-dimensional sequence data and its application to the field of metagenomics, which aims at reconstructing individual genomes from a mixture of genomes sampled from an environmental site, without any prior knowledge of reference data (genomes) or the shape of clusters. Such problems typically cannot be solved directly with classical approaches seeking to estimate the density of clusters, for example, using the shared nearest neighbors (SNN) rule, due to the prohibitive size of contemporary sequence datasets. We explore here a new approach based on combining the SNN rule with the concept of locality sensitive hashing (LSH). The proposed method, called LSH-SNN, works by randomly splitting the input data into smaller-sized subsets (buckets) and employing the SNN rule on each of these buckets. Links can be created among neighbors sharing a sufficient number of elements, hence allowing clusters to be grown from linked elements. LSH-SNN can scale up to larger datasets consisting of millions of sequences, while achieving high accuracy across a variety of sample sizes and complexities.

Keywords: density-based methods, metagenomic data, sequence clustering, locality sensitive hashing.

1. INTRODUCTION

CLUSTERING IS USUALLY DEFINED AS the task of unsupervised learning, where the class labels of the data items are unknown (Jain and Dubes, 1988; Kotsiantis and Pintelas, 2004; Berkhin, 2006; Hastie et al., 2009). Clustering methods aim at creating categories from the data in such a way that similar objects will be grouped together, whereas dissimilar objects will be separated into different groups, referred to as clusters. Important issues in clustering research focus on the effectiveness and scalability of the methods on data of varying complexities and arising from various domains (Yeung et al., 2001; Liao et al., 2004; Aggarwal, 2009).

Commonly used methods to cluster high-dimensional data are presented in Kriegel et al. (2009). *K*-means is one of the most widely used clustering method due to its low algorithmic complexity. However,

[1]CEA, Genoscope, Evry, France.
[2]CEA, LIST, Laboratoire d'Analyse de Données et Intelligence des Systèmes, Gif-sur-Yvette, France.
[3]Université d'Evry, Evry, France.
[4]CNRS-UMR 8030, Evry, France.
[5]Université Paris-Saclay, Evry, France.

it has been shown in Wu (2012) that K-means tends to produce clusters of relatively uniform sizes and globular shapes, even if the data structure is endowed with varying cluster sizes or different shapes. This bias is known as the uniform effect of the K-means. Moreover, the number of clusters K has to be specified *a priori*, which is not trivial when no prior knowledge is available. To address these problems, methods based on estimating the density and/or the similarity among instances have been introduced (Jarvis and Patrick, 1973; Ester et al., 1996).

In Ertöz et al. (2004), the authors presented an effective clustering method based on two key notions: the similarity between neighboring elements and the density around instances. This method, shared nearest neighbors (SNN), is a density-based clustering method and incorporates a suitable similarity measure to cluster data. After finding the nearest neighbors of each element and computing the similarity between pairs of points, SNN identifies core points, eliminates noisy elements, and builds clusters around the core elements. This method can yield improved performance compared with other clustering approaches with data of varying densities, and it can automatically infer the number of output clusters. However, this method has complexity $O(n^2)$, where n is the number of instances in the dataset, arising from the computation of the similarity matrix, which can be prohibitive when dealing with very large amounts of data.

One interesting concept to reduce the burden of computing the similarity matrix is locality sensitive hashing (LSH). This concept was initially introduced to find approximate near neighbor information in high-dimensional space (Indyk and Motwani, 1998; Har-Peled et al., 2012; Wang et al., 2014a). The key idea is to hash elements into different buckets; then for a query instance $\mathbf{x}$, to use instances stored in buckets containing $\mathbf{x}$ as candidates for near neighbors. This approximation reduces the query time complexity to $O(\log n)$ instead of $O(n)$ ($O(n)$ is the complexity for searching nearest neighbors for one instance). Therefore, the similarity matrix computation time can be reduced to $O(n \log n)$.

We propose here to retain the basic principle of LSH by randomly splitting the dataset into a number of smaller-sized subsets, using a family of hashing functions, so that similar elements will be hashed together with high probability. We then look for nearest neighbors of each element in its bucket, and construct links among elements sharing a significant number of neighbors to generate clusters. The proposed method, called LSH-SNN, has the advantage of reducing the complexity for computing the similarity matrix, while maintaining the same level of clustering accuracy.

In this study, we have evaluated the performance of the LSH-SNN method on metagenomic datasets of various sizes and complexities. We have also compared the results with another density-based clustering algorithm and the K-means method implemented in a popular sequence clustering software called MetaCluster (Yang et al., 2010; Wang et al., 2014b). Many computational tools have been proposed in the literature to analyze metagenomic sequences that are generated from microbial communities. These tools can be grouped into two main categories: (1) supervised and (2) unsupervised methods. Supervised methods, often relying on sequence similarities and alignments of DNA fragments to reference sequences of known taxonomic origins, include tools such as Kraken (Wood and Salzberg, 2014), Clark (Ounit et al., 2015), and Diamond (Buchfink et al., 2015). Unsupervised methods group metagenomic fragments based on intrinsic features, such as the statistics of l-mer frequencies extracted from fragments. Unsupervised methods, such as MetaCluster (Yang et al., 2010), MaxBin (Wu et al., 2014), and MetaProb (Girotto et al., 2016), became attractive due to the lack of reference genomes for the bulk of micro-organisms identified by culture-independent approaches.

The rest of this article is organized as follows: Section 2 surveys related work on clustering; Section 3 recalls some background on local sensitive hashing and the SNN methods; and Section 4 introduces our method based on the combination of local sensitive hashing and SNN. Experimental results on synthetic and real datasets are illustrated in Sections 5 and 6, respectively, whereas Section 7 concludes the article.

2. RELATED WORK

Clustering methods look for similarities within a set of instances without any need for prior data labeling. Numerous methods have been proposed in literature to deal with clustering tasks. Existing algorithms can be grouped into five categories as proposed in Berkhin (2006) and Kotsiantis and Pintelas (2004): partitioning methods, hierarchical methods, density-based methods, grid-based methods, and model-based methods. Hereafter, we will describe the main characteristics of these methods.

Partitioning methods construct K partitions of the data by grouping instances around the gravity center of each cluster. They can be divided into two main groups: the centroid methods such as K-means (Hartigan and Wong, 1979), and the medoids ones (Kaufman and Rousseeuw, 1987) such as the K-modes (Huang, 1998) and the K-prototypes algorithms (Huang, 1997). Partitioning methods are simple to implement; however, the number of clusters K should be specified.

Hierarchical methods build a tree hierarchy, known as dendrogram, to form clusters in two different manners: agglomerative (bottom-up) and divisive (top-down). The former starts with singleton clusters and recursively merges them in a bottom-up strategy, whereas the latter breaks the dataset into smaller clusters in a top-down strategy. They use various local criteria to join or split clusters. Hierarchical methods have the advantage of handling any form of similarity without requiring the number of clusters to be known in advance. However, to construct a dendrogram, they suffer from time and space complexities, which are quadratic with respect to the number of clusters. Hierarchical clustering includes methods such as: BIRCH (Zhang et al., 1996), CURE (Guha et al., 1998), and CHAMELEON (Karypis et al., 1999).

Density-based methods generate clusters based on the density of instances in a region. These methods are related to different concepts defining a point's nearest neighbors, such as density, connectivity, and boundary. Density-based methods can find arbitrary-shaped clusters; however, they output border instances, which may be unclustered and considered as outliers. Existing methods include DBSCAN (Ester et al., 1996), OPTICS (Ankerst et al., 1999), DENCLUE (Hinneburg and Keim, 1998), Jarvis-Patrick (JP) (Jarvis and Patrick, 1973), and SNN (Ertöz et al., 2003) algorithms. SNN will be described in further detail in Section 3.2.

Grid-based methods quantize the space into a finite number of cells that form a grid structure. Clustering is then performed on the grid cells, instead of the database itself (Liao et al., 2004). The main advantage of these methods is their fast processing time; however, they output clusters with either vertical or horizontal boundaries (no diagonal boundary can be detected). This category includes STING (Wang et al., 1997), WaveCluster (Sheikholeslami et al., 1998), and CLIQUE (Agrawal et al., 1998) algorithms, among others.

In model-based clustering, it is assumed that the data are generated from K probability distributions and the goal is to find the distribution parameters (Yeung et al., 2001). Model-based methods are characterized by a small number of parameters; however, the computational burden can become significant if the number of distributions is large. Moreover, it is difficult to estimate the number of clusters. Many model-based clustering methods are described in the literature, such as Expectation-Maximization (Dempster, Laird and Rubin, 1977), Self Organized Maps net (Kohonen, 2001), and AutoClass (Cheeseman and Stutz, 1996).

None of the categories mentioned earlier can directly cluster large amounts of instances of arbitrary shapes and at the same time automatically detect the appropriate number of clusters. The SNN algorithm from the density-based clustering category can deal with local density variations and automatically find clusters of different shapes. However, adapting this technique with massive data requires extensive storage and time costs, especially for the computation of the similarity matrix.

3. BACKGROUND

In this section, we briefly review some background on locality sensitive hashing (Subsection 3.1) and SNN algorithms (Subsection 3.2).

3.1. Locality sensitive hashing

LSH was first introduced in Indyk and Motwani (1998) as a classical geometric lemma on random projections, to quickly find similar items in large datasets. One or many families of hash functions map similar inputs to the same hash code. This hashing technique produces a splitting of the input space into many subspaces, called bins or buckets, with a high probability that instances originally close in their input space will be in the same bin or in adjacent bins within the LSH framework.

To alleviate the curse of dimensionality, each hash function projects the data to a lower-dimensional space ($h : \mathbb{R}^d \rightarrow \mathbb{Z}$). Different techniques have been presented in the literature to generate

hash functions. These techniques can be categorized into two families: min-hash (Broder, 1997) and random projections (sim-hash). In document classification, min-hash is typically used when looking for textually similar documents by processing items and generating integers from strings of characters (Leskovec et al., 2014). Random projections, on the other hand, are obtained via simple probability distributions such as p-stable distribution (Datar et al., 2004) and sign-random-projection (Charikar, 2002).

Dealing with large datasets, LSH is typically combined with nearest neighbors techniques (Buhler, 2001) and used for data clustering (Broder, 1997; Boydell et al., 2013). To perform k-nearest neighbors, buckets [and sometimes their adjacent buckets (Lv et al., 2007)] containing the query element are checked and all the existing instances are ranked according to their distances to the query element. To cluster high-dimensional data (Haveliwala et al., 2000; Koga et al., 2004), similar elements contained in the same bucket can be joined to output clusters in a hierarchical way (Rasheed et al., 2012).

3.2. Shared nearest neighbor

SNN is a density-based clustering approach that was successfully applied for finding groups of documents sharing a strong, coherent topic or theme (Gowda and Krishna, 1978; Ertöz et al., 2004; Steinbach et al., 2004; Berkhin, 2006; Moëllic et al., 2008; Patidar et al., 2012). SNN handles clusters of widely differing sizes, densities, shapes, and in the presence of large amounts of noise and outliers. To exploit space density of the data, SNN relies on the concept of similarity based on the SNN approach. The similarity matrix is sparsified by keeping only the k-most nearest neighbors (*knn*). The SNN graph is then constructed by creating links between pairs of instances having each other in their respective *knn* lists. The weight of the link can be calculated either as the number of shared neighbors between two *knn* lists or using the ordering of these shared neighbors.

The algorithm determines the type of each instance (core, border, or noisy) by calculating its connectivity; that is, the number of links coming out of this instance, which will be compared with *noisy* and *topic* thresholds. Noisy instances are discarded and will never be used in the clustering process. Core instances form final clusters with their connected elements. The algorithm's behavior is controlled by four parameters: the number of nearest neighbors (noted as *knn* hereafter), the *topic* threshold, and two other thresholds governing the addition of elements to clusters. Depending on the parameter settings, many of the border instances can remain unlabeled because they are unconnected to core elements.

4. OUR PROPOSAL: THE LSH-SNN ALGORITHM

We describe the key idea of our algorithm, called LSH-SNN, in Section 4.1. We then describe how to tune the different parameters in Section 4.2.

4.1. Method description

SNN is a relatively effective unsupervised method to automatically find clusters of different shapes and densities. For n data items and *knn* nearest neighbors, the computational complexity of SNN is $O(n^2)$, whereas its space complexity is $O(knn * n)$. Hence, SNN can face important scalability issues when the number of instances n becomes large.

Our goal is to adapt a suitable framework for clustering large number of instances by using the SNN principle. It is motivated by large metagenomic datasets incurring high computational costs. To help reduce these costs, we consider the rationale of locality sensitive hashing as a framework for the development of the SNN method. In this framework, the n data instances are randomly partitioned into a number of smaller-sized subsets called buckets, and for each data point, we locate its approximate nearest neighbors inside its bucket. This approach, called LSH-SNN, has an advantage over SNN since it restricts the calculation of distances for each single item inside its bucket, whereas SNN needs to calculate n distance measures for each point before selecting the list of nearest neighbors.

LSH-SNN begins with the extraction of features from the sequence fragments by computing the frequencies of all possible l-mers (substrings of length l) in each of them (Section 4.1.1). Nearest neighbors of

all sequences are then computed by applying the LSH technique, which involves splitting sequences into different buckets in such a way that similar sequences end up in the same bucket with a higher probability (Section 4.1.2). Elements stored in buckets containing a given sequence $\mathbf{x}$ are retrieved and ranked according to their distances to $\mathbf{x}$ to compute its nearest neighbors list. Shared neighbors are linked according to the SNN rule, and connected sequences form output clusters (Section 4.1.3). Finally, in the case of unclustered fragments, a last step is performed to assign them to the cluster that is the most similar in terms of l-mer distribution (Section 4.1.4).

4.1.1. l-mer *frequency calculation.* Sequence similarities are typically identified by comparing occurrence patterns of relatively short DNA substrings of length l between the sequences (Yang et al., 2010; Tanaseichuk et al., 2012).

Two broad scenarios can be used to assess l-mer-based similarities: Abundance-based methods make use of relatively large l values ($l \geq 20$) to ensure the uniqueness of most l-mers (Tanaseichuk et al., 2012), whereas composition-based methods rely on smaller l values. Since DNA is a combination of four different types of nucleotides (A,T,G,C), there are at most 4^l l-mer combinations forming the feature vector. The frequency of each l-mer combination is normalized by dividing the number of occurrences by the fragment length.

Because of nucleotide base complementarities, the size of the feature vector can be reduced by half, that is, for a DNA sequence $\mathbf{x}$ of length s, the feature vector is given by: $\mathbf{x} = (\omega_1, \omega_2, \ldots, \omega_m)$, where $\omega_i = \frac{f_i}{s}$, f_i is the frequency occurrence of a given l-mer combination and m represents the size of the vector (or number of descriptive attributes), $m = 4^l/2$ if l is odd, and $(4^l + 4^{l/2})/2$ if even.

4.1.2. Locality sensitive hashing. For convenience, we briefly recall some notations. Let $\mathbb{X}$ be the collection of n sequences of m-dimensional features, and let $\mathbf{x} \in \mathbb{X}$ denote an input sequence. Let k be the number of projections. For each $i \in [k]$, $h_i(\mathbf{x})$ is given by: $h_i(\mathbf{x}) = \text{sign}(\mathbf{x}.v_i)$, where v_i is a vector whose components are randomly generated from a Gaussian distribution, for example, $\mathcal{N}(0, 1)$. This scalar projection gives one hash value for $\mathbf{x}$. The hash code for $\mathbf{x}$ is then obtained by a concatenation of the k hash values, $g(x) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_k(\mathbf{x}))$. LSH prepares r copies of $g(.)$ to improve the hashing discriminative power (Dasgupta et al., 2011, Wang et al., 2014a) (to avoid confusion, k [in lower case] is the number of sampled bits, whereas K [in upper case] is the number of output clusters).

The feature vectors are first normalized with zero mean and unit variance. Each input sequence $\mathbf{x}$ is then indexed by a hash code $g(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_k(\mathbf{x}))$ defining its bucket identity, and the hash code of all sequences in $\mathbb{X}$ constitutes a hash table. This projection produces a new k-dimensional space ($k << m$). Since the number of elements per bucket is typically much smaller than n, we need to ensure that similar sequences share the same bucket with a higher probability while minimizing random effects. To achieve this, r hash functions $g_1, g_2, \ldots, g_r$ are sampled independently, each generating a distinct hash table. For each sequence $\mathbf{x}$, we then identify $g_1(\mathbf{x}), g_2(\mathbf{x}), \ldots, g_r(\mathbf{x})$ indexing the r buckets where $\mathbf{x}$ is mapped in each projection.

Note that any hash function could be applied in this step (Paulevé et al., 2010) when distances are measured as angles between pairs of points. In this work, we demonstrate results based on random hash functions that are generated from a Gaussian distribution.

The projection of the data items into different buckets can be summarized as follows:

Algorithm 1: Hashing

Input: Set of DNA sequences $\mathbb{X}$ of size n, number of repetitions r
Output: Set of matrices of hash code $\mathbb{T}$ corresponding to $\mathbb{X}$:
1: $\mathbb{X} = (\mathbb{X} - \text{mean}(\mathbb{X}))/\text{std}(\mathbb{X})$;
2: $k = \log(n)$; ▷ k is the number of projections or axis
3: **for** $i = 1$ to r **do**
4: Create an m-by-k matrix A_i where rows are identical and generated from a Gaussian distribution ($\mathcal{N}(0, 1)$); ▷ m is the number of features
5: $\mathbb{T}_i = \mathbb{X}.A_i$;
6: **end for**

For hashing the dataset, the time complexity is $O(m \times k \times r)$ per sequence, since each sequence of dimension m will be processed by k hash functions repeated r times. Therefore, for n sequences, the time complexity of LSH is $O(n \times m \times k \times r)$. For a fixed l-mer (and hence of m) value, LSH has a complexity of $O(n \times \log(n) \times r)$ ($k = \log(n)$; see Section 4.2).

4.1.3. Shared nearest neighbors. Once the space has been partitioned $k \times r$ times, a simple k-nearest neighbor classifier may be considered to find the nearest neighbors of a sequence $\mathbf{x}$ inside its bucket (i.e., sharing the same hash code) for all partitions. The union of r subsets of nearest neighbors for a given sequence is treated as its neighborhood list. Since the nearest neighbor lists are generated from sparsely populated buckets, the computational cost and runtime can be greatly improved.

Once the sets of nearest neighbors have been defined, the SNN algorithm follows two steps: computing the strength of links and the labeling of sequences. A link is created between two sequences $\mathbf{x}_1$ and $\mathbf{x}_2$ if they have each other in their respective neighborhood lists, and it can be weighted according to the product of positions of shared instances between these two lists, namely:

$$link(\mathbf{x}_1, \mathbf{x}_2) = \sum (knn + 1 - p_1) * (knn + 1 - p_2), \tag{1}$$

where p_1 and p_2 are the positions of a shared neighbor in the lists of $\mathbf{x}_1$ and $\mathbf{x}_2$. The *knn* lists are then transformed into a graph where sequences (nodes) are connected via weighted links. For each sequence $\mathbf{x}$ in the graph, the sum of its total links ($conn_x$) is computed to enable the selection of a subset of representative sequences according to a connectivity-based criterion ($conn > topic\ threshold$).

Algorithm 2 inset summarizes the SNN method. To check the nearest neighbors of a sequence $\mathbf{x}$, n' distances need to be evaluated, where n' is the number of elements sharing the same hash code as $\mathbf{x}$. Since we have n sequences, the time complexity for computing the nearest neighbor elements is $O(n \times n' \times m + C(knn))$, where $C(knn)$ is a relatively small factor enabling the selection of *knn* near neighbors for each sequence (Jarvis and Patrick, 1973). To assess the link between two sequences having each other in their respective *knn* list, two columns of size *knn* are selected and evaluated. The cost of this process amounts to $O(n \times knn \times knn)$.

Therefore, the total complexity of the algorithm becomes:

$$\begin{aligned}
&Complexity \\
&= O(n \times k \times r + n \times m \times n' + n \times knn \times knn) \\
&= O(n \times k \times r + n \times n' + n \times knn \times knn) && \text{(for fixed } l\text{-mer value)} \\
&= O(n \times \log(n) \times r + n \times n' + n \times n') && knn = \sqrt{n'} \\
&= O(n \times \log(n) \times r + 2 \times n \times n') && n' \approx n/2^{\log(n)} \\
&\approx O(n \times \log(n) \times r)
\end{aligned}$$

This algorithm is able to handle clusters of different densities, but it can leave a large number of nonnoisy sequences unclustered. To alleviate this problem, we define an additional step to relabel unclustered sequences.

4.1.4. Relabeling. This relabeling step was developed to reduce the number of unclustered sequences. Briefly, it identifies a subset of frequencies that are characteristic of each cluster and contributing most to the classifier's accuracy, discarding less relevant features. Each unclustered sequence is then added to the cluster that is the most closely related with respect to this subset of frequencies.

Relabeling proceeds by computing the mean of each cluster and dividing it by the mean of the other clusters. Discriminant l-mer frequencies, that is, those that most differentiate this cluster from others, are selected (cf line 7 in Algorithm 3 inset). Then, for a given sequence, we compute its distance to the mean of each cluster by using the subset of discriminant frequencies and assign it to the nearest cluster. If two clusters are nearly equally close to a given sequence, we keep the latter unlabeled to avoid increasing the number of misclassified instances.

Algorithm 2: SNN computation

Input: Set of DNA sequences $\mathbb{X}$ of size n, set of matrices of hash code $\mathbb{T}$ corresponding to r projections of $\mathbb{X}$, the number of nearest neighbors *knn*, *strong*, *topic*, and *merge* thresholds
Output: Number of clusters K, predicted set of labels $\hat{Y}$ corresponding to $\mathbb{X}$:

for $i = 1$ to n **do**
  for $j = 1$ to r **do**
    Select Z_j the bucket indexing by the same hash code of $\mathbf{x}_i$, $\mathbb{T}_{ij}$;
    Compute the distances of $\mathbf{x}_i$ to each element in Z_j;
    Select the *knn* nearest neighbors elements and add them to $N_{\mathbf{x}_i}$;
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to $r \times knn$ **do**
    Select an element $\mathbf{x}_j$ from $N_{\mathbf{x}_i}$;
    if $\mathbf{x}_i \in N_{\mathbf{x}_j}$ **then**
      for each element in $N_{\mathbf{x}_i} \cap N_{\mathbf{x}_j}$ **do**
        Compute $links(i, j)$ using 1
      end for
    end if
    if $links(i, j) \geq strong$ **then**
      Increment $conn_{\mathbf{x}_i}$ and $conn_{\mathbf{x}_j}$;
    end if
  end for
end for
Set K to zero; ▷ number of clusters is equal to zero;
for $i = 1$ to n **do**
  if $conn_{\mathbf{x}_i} \geq topic$ **then**
    if $\hat{y}_i$ is not labeled **then** ▷ we have a representative sequence that could create a new cluster
      Increment K;
      Set $\hat{y}_i$ to K;
    end if
    for $j = 1$ to $r \times knn$ **do**
      if $links(i, j) \geq merge$ **then**
        Set $\widehat{y_{N_{\mathbf{x}i,j}}}$ to $\hat{y}_i$;
      end if
    end for
  end if
end for

The implementation of this part of the algorithm is presented as a pseudo code in Algorithm 3 inset.

Relabeling requires $n \times K$ operations, hence a complexity of $O(n \times K)$. The overall complexity of the algorithm depends on the complexity of the hashing functions and the SNN classifier. The total complexity is $\approx O(n \times \log(n) \times r)$.

4.2. Parameters in LSH-SNN

This section discusses the configuration of the LSH and SNN parameters, which impact the method's performance in terms of both runtime and clustering quality. Parameters were determined by grid search and focused on optimizing the V-measure (see Section 5.3).

LSH has two parameters, k and r: (1) The number k of bits sampled determines the number of instances inside the buckets, which, on average, is expected to be equal to $n' = \frac{n}{2^k}$. The total number of buckets is limited to $\max(n, 2^k)$. If k takes a small value with respect to the number of sequences, then we would end up with a large number of sequences per bucket (n') and the time consumption of the SNN phase will be very high. On the other extreme, if $k = n$, we would get on average one sequence per bucket and there will be no *knn* lists to be constructed. In this study, we set k to $\log(n)$. (2) The number of projections r is the second parameter. For $r = 1$, we take the risk that two close elements end up in distinct buckets because of the random nature of the hashing. By increasing the number of projections, we increase the probability that these two elements are mapped to the same bucket. The number of projections r should, thus, be increased

Algorithm 3: Relabeling of unclustered sequences

Input: Set of DNA sequences $\mathbb{X}$ of size n, number of projections r, number of clusters K, predicted set of labels $\hat{Y}$ corresponding to $\mathbb{X}$
Output: Predicted set of labels $\hat{Y}$:
1: **for** $i = 1$ to K **do**
2: Compute the mean of cluster i (m_i);
3: Compute the standard deviation of cluster i (sd_i);
4: **for** $j = 1$ to K **do**
5: Compute the mean of cluster j (m_j);
6: **for** $ind = 1$ to m **do**
7: **then if** $(((m_i[ind]/m_j[ind] \geq K) \vee (m_i[ind]/m_j[ind] \leq 1/K)) \wedge (sd_i[ind]$ is small))
8: ind is a significant frequency;
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: **for** $i = 1$ to n **do**
14: **if** $\hat{y}_i$ is not labeled **then**
15: Compute the distance of $\mathbf{x}_i$ to each cluster by using only the significant frequencies;
16: Label $\mathbf{x}_i$ with the cluster having the nearest distance to $\mathbf{x}_i$;
17: **end if**
18: **end for**

to stabilize the results. On the other hand, for large values of r, distant elements may be mapped to the same bucket, provoking the $r \times knn$ lists to grow and ultimately leading to the same drawbacks as the initial SNN method. In this study, r was empirically set between 300 and 1200, depending on the size of the dataset.

Four parameters determine the outcome of the SNN procedure: These are the *knn*, *topic*, *merge*, and *strong* thresholds. (1) The size of the nearest neighbor list *knn* ultimately depends on the number of elements per bucket. To construct *knn* lists containing a sufficiently large number of closely related sequences that are consistent with the shared neighbors criterion, we need to choose a relatively large number of nearest neighbors. However, increasing *knn* may add more distant sequences to the same cluster, thus increasing the computational cost. On the other hand, decreasing *knn* will result in many smaller-sized clusters. A simple and pragmatic approach is to set $knn = \sqrt{n'}$ or $knn = \log(n')$, where $n' = \frac{n}{2^k}$. In our experiments, we fixed *knn* to $\sqrt{(n')}$. (2) The *topic* threshold determines the proportion of highest *connectivity* links to be selected as representatives. This threshold ranged from 0.04 to 0.06 in our experiments. (3) The *merge* threshold represents the fraction of links to be used in the cluster merging process, and it was fixed to 0.02 in our experiments. (4) Finally, the *strong* threshold aims at reducing the number of unlabeled sequences (singleton clusters) and at picking representative elements. In our experiments, this parameter was set to 0.1.

5. EXPERIMENTS ON SEMI-SYNTHETIC DATASETS

In this section, we detail the experimental setup with the semi-synthetic datasets. We first describe the datasets (Section 5.1) and provide a short reminder about the methods that we compare our algorithm with (Section 5.2). We then detail the metrics used to evaluate the performance of our method (Section 5.3) and present and discuss the results in Section 5.4.

5.1. Datasets

We have used synthetic datasets of increasing sizes and complexities, composed of 600 base-pair length reads (DNA fragments) sampled from sequenced bacterial genomes that were assembled to create synthetic communities of known composition and abundance (Gkanogiannis et al., 2016), which were then used as ground truth to measure the clustering accuracy of the different algorithms evaluated. The mean coverage of the datasets was fixed to 1X, which means that, on average, a given position in the genome is covered by 1 read. The number of reads derived from each organism is equal to 5000.

To evaluate the performance of the various clustering modules on the benchmark datasets, we compared class memberships of elements (reads) in each dataset with the memberships induced by the clustering. Class memberships correspond to the genomes that the reads were sampled from, with the cardinality of the class set matching the samples' richness. For all the synthetic datasets, the read generation process was performed by using the Mason software (Holtgrewe, 2010) with default error model parameters for Illumina reads (Mason can insert position-specific sequence modifications according to empirically calibrated and sequencing platform-dependent error models).

5.2. Benchmark methods

The LSH-SNN method is compared with MetaCluster (Yang et al., 2010), a popular compositional binning software based on the K-means algorithm with the Spearman footrule distance, which operates on relative rankings of the *l*-mer frequencies (Yang et al., 2010).

The main advantage of this approach is its simplicity, which underlies its ability to handle datasets harboring a relatively large number of genomes. However, its behavior can be sensitive to the random choice of initial cluster centers, and it may fail to output clusters when the data present nonglobular shapes. Moreover, the number of clusters should be predetermined, which is not trivial in the absence of prior knowledge (Rokach and Maimon, 2005).

The complexity of the *K*-means method is $O(n \times m \times K \times t)$, where n is the number of instances, m is the dimension of data, K is the number of clusters, and t is the number of iterations for convergence. As *l*-mer is fixed for all experiments, the overall complexity of MetaCluster becomes $O(n \times K \times t)$.

We also compared our algorithm with the JP method (Jarvis and Patrick, 1973) coupled with the LSH indexing in a way similar to the LSH-SNN combination. JP also relies on the nearest neighbors similarity concept, but it simply merges elements in the same cluster if they share a sufficient number of neighbors, that is, the number of shared elements between two neighbors is greater than a user-predefined threshold kt, which we fixed to $(r * knn)^2/2$. The complexity of LSH-JP is $O(n \times k \times r + n \times n' + n \times knn \times knn) \approx O(n \times \log(n) \times r)$.

5.3. Performance evaluation

We considered the following metrics to evaluate the performance of our method: Homogeneity, Completeness, V-measure, F-measure, and the Adjusted Rand Index.

Homogeneity evaluates the class distribution within each cluster; it is highest when each cluster contains only elements from a single class. *Completeness*, on the other hand, measures the distribution of cluster assignments within each class; it is highest when elements of a given class are assigned to a single cluster. Let C be the number of distinct organisms in a dataset of n sequences and K be the number of output clusters. These two measures are defined as follows (Rosenberg and Hirschberg, 2007):

$$Homogeneity = 1 - \frac{\sum_{i=1}^{K}\sum_{j=1}^{C}\frac{n_{i,j}}{n}\log\frac{n_{i,j}}{\sum_{j=1}^{C}n_{i,j}}}{\sum_{j=1}^{C}\frac{\sum_{i=1}^{K}n_{i,j}}{n}\log\frac{\sum_{i=1}^{K}n_{i,j}}{n}}, \tag{2}$$

$$Completeness = 1 - \frac{\sum_{j=1}^{C}\sum_{i=1}^{K}\frac{n_{i,j}}{n}\log\frac{n_{i,j}}{\sum_{i=1}^{K}n_{i,j}}}{\sum_{i=1}^{K}\frac{\sum_{j=1}^{C}n_{i,j}}{n}\log\frac{\sum_{j=1}^{C}n_{i,j}}{n}}, \tag{3}$$

The V-measure is defined as the harmonic mean of *homogeneity* and *completeness*. It is calculated as:

$$V - measure = \frac{2 * homogeneity * completeness}{homogeneity + completeness}. \tag{4}$$

The clustering accuracy, *F-measure*, is defined as:

$$F - measure = \sum_{j=1}^{C}\frac{n_j}{n}\max_i F(i,j) \tag{5}$$

With

$$F(i,j) = \frac{2 \times \frac{n_{i,j}}{n_i} \times \frac{n_{i,j}}{n_j}}{\frac{n_{i,j}}{n_i} + \frac{n_{i,j}}{n_j}} \tag{6}$$

where n_j is the cardinality of cluster C_j.

Finally, the *Adjusted Rand Index* (Hubert and Arabie, 1985; Yeung and Ruzzo, 2001) computes a similarity measure between the observed and ideal clusterings as:

$$ARI = \frac{\sum_{i=1}^{K}\sum_{j=1}^{C}\binom{n_{ij}}{2} - \left[\sum_{i=1}^{K}\binom{n_i}{2}\sum_{j=1}^{C}\binom{n_j}{2}\right]/\binom{n}{2}}{\frac{1}{2}\left[\sum_{i=1}^{K}\binom{n_i}{2} + \sum_{j=1}^{C}\binom{n_j}{2}\right] - \left[\sum_{i=1}^{K}\binom{n_i}{2}\sum_{j=1}^{C}\binom{n_j}{2}\right]/\binom{n}{2}} \tag{7}$$

These measures all take values between 0 and 1, with higher values corresponding to better clustering accuracies.

5.4. Discussion

After tuning the parameters described in Section 4.2, we evaluated the performance of LSH-SNN as well as of two other clustering algorithms, LSH-JP and *K*-means (the MetaCluster implementation), on the different datasets (note that MetaCluster requires the number of component genomes to be provided as input for each dataset). Clustering accuracy was quantified by using the previously described metrics. The results are shown in Table 1 together with the rank (between parentheses) of each method, and they highlight (in bold) the best value for each evaluation criterion. The fraction of unclustered sequences (singleton elements) varied between 20% and 30% for LSH-SNN, 60% and 90% for LSH-JP, and between

Table 1. Performance on Synthetic Datasets

Datasets	*Metrics*	*LSH-SNN*	*LSH-JP*	*MetaCluster*
MC5	Homogeneity	0.502(2)	**0.614**(1)	0.302(3)
	Completeness	0.504(2)	0.499(3)	**0.618**(1)
	V-measure	0.503(2)	**0.551**(1)	0.406(3)
	F-measure	**0.642**(1)	0.469(3)	0.574(2)
	Adjusted Rand Index	**0.645**(1)	0.296(3)	0.405(2)
MC10	Homogeneity	**0.512**(1)	0.499(2)	0.415(3)
	Completeness	**0.721**(1)	0.704(2)	0.632(3)
	V-measure	**0.598**(1)	0.584(2)	0.501(3)
	F-measure	**0.561**(1)	0.461(3)	0.522(2)
	Adjusted Rand Index	**0.504**(1)	0.312(3)	0.417(2)
MC25	Homogeneity	**0.516**(1)	0.335(3)	0.443(2)
	Completeness	0.548(3)	**0.629**(1)	0.614(2)
	V-measure	**0.531**(1)	0.437(3)	0.515(2)
	F-measure	0.320(2)	0.304(3)	**0.476**(1)
	Adjusted Rand Index	**0.349**(1)	0.178(3)	0.244(2)
MC50	Homogeneity	**0.492**(1)	0.353(3)	0.389(2)
	Completeness	0.674(2)	**0.713**(1)	0.594(3)
	V-measure	**0.569**(1)	0.471(2)	0.471(2)
	F-measure	0.309(2)	0.253(3)	**0.372**(1)
	Adjusted Rand Index	**0.332**(1)	0.141(3)	0.167(2)
MC100	Homogeneity	**0.492**(1)	0.176(3)	0.249(2)
	Completeness	0.674(2)	**0.704**(1)	0.567(3)
	V-measure	**0.569**(1)	0.282(2)	0.346(2)
	F-measure	0.309(2)	0.141(3)	**0.227**(1)
	Adjusted Rand Index	**0.332**(1)	0.085(2)	0.018(3)

The name of the datasets is MCx, where x corresponds to the number of genotypes.
JP, Jarvis-Patrick; LSH, locality sensitive hashing; SNN, shared nearest neighbors.

Table 2. Performance on Real Metagenomic Data

Cluster_id	*No. of pathologic strain contigs*	*Total no. of contigs*
30	819	1082
406	244	324
33	9	24
1	8	7527
581	1	7

The two highest peaks in Figure 1 correspond to the first clusters (shaded) of Table 2; these clusters contain the bulk of the pathogen genome.

2% and 4% for MetaCluster. These figures underlie the lower Adjusted Rand Index and F-measure values achieved by LSH-JP. On the other hand, the relabeling step in the LSH-SNN algorithm specifically aimed at reducing the number of unclustered sequences.

Table 1 indicates that LSH-SNN outperforms LSH-JP and MetaCluster in terms of *homogeneity* and *V-measure*, with the latter being the harmonic mean between *homogeneity* and *completeness*. LSH-SNN also slightly outperforms the other methods on the *F-measure* in four out of seven datasets, and it consistently yields the best performance on all the datasets in terms of the *Adjusted Rand Index* metric.

The LSH-SNN algorithm is implemented in the C++ programming language, and it uses the OpenMP application programming interface to support multiprocessing. Experiments were conducted on a Linux x86_64 server endowed with multi-core CPUs and 2 TB of RAM. The LSH-SNN and LSH-JP computations were parallelized on 48 cores.

Overall, these results demonstrate that LSH-SNN achieves accurate binning for DNA sequences as short as 600 bp, as compared with LSH-JP and MetaCluster's *K*-means implementation, and despite the latter being aware of the correct number of clusters (genomes) as an input parameter.

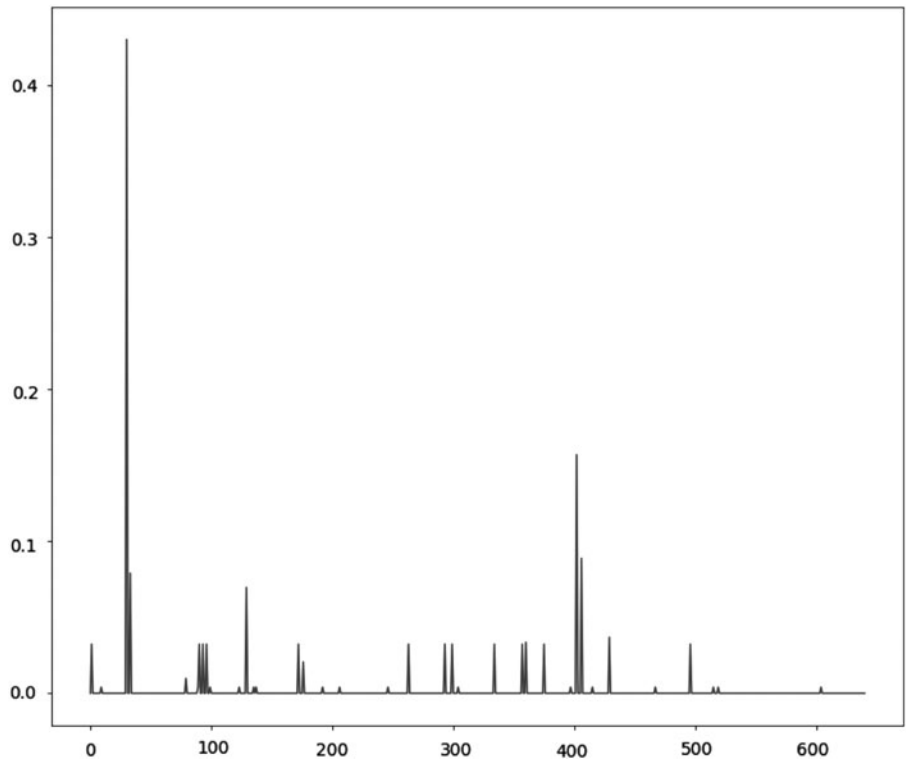

FIG. 1. The output clusters are arbitrary labeled with their identifier on the x axis. The y axis displays mutual information of the final clusters, which reflects the correlation of the clusters (treated as variables) with metadata labels (i.e., health status) of the individuals from the cohort.

6. EXPERIMENTS ON REAL DATASETS

To assess the behavior of the LSH-SNN method on real metagenomic data, we ran our algorithm on a set of 142,723 contigs resulting from a cross-assembly of 53 human microbiomes (Alneberg et al., 2014), 43 of which harbored the *Escherichia coli* strain responsible for the 2011 Shiga toxin-producing *E. coli* (STEC) outbreak in Germany. We observed that (1) the number of output clusters was consistent with species diversity estimates derived from 16S ribosomal small subunit RNA analyses of the same samples (Alneberg et al., 2014), (2) the pathologic *E. coli* strain genome was distributed among a very limited number of homogeneous clusters (Table 2), and (3) these pathogen-bearing clusters were the most discriminant variables to separate the individuals infected with the STEC strain from the remaining ones (Figure 1).

These observations demonstrate that the LSH-SNN algorithm performs rather decently on real-world datasets of biomedical relevance.

7. CONCLUSION

We have proposed an unsupervised composition-based method for binning large volumes of metagenomic sequences, without any prior knowledge of their reference genomes or the number of distinct genotypes present in the sample. LSH-SNN is based on two key steps: the hashing/indexing of the data space and the linking of sequences to build clusters. After computing the l-mer distribution of each sequence, LSH partitions the input space into buckets containing smaller subsets of sequences whose connectivity is evaluated via the SNN rule. A third step was added to reduce the number of singletons or unclustered sequences.

The LSH-SNN algorithm can scale to datasets containing millions of sequences, and it does not require the number of output clusters to be predetermined. Although the presented algorithm makes use of the SNN rule, we envision that the LSH concept could be combined with other sequence processing (including clustering) methods dealing with large data volumes, or it could be used on its own as exemplified in Berlin et al. (2015), where a MinHash LSH scheme was used to compute similarities between long noisy reads generated with a new single-molecule real-time sequencing technology.

The LSH-SNN algorithm was evaluated on seven semi-synthetic metagenomic datasets of different sizes and complexities (i.e., harboring different numbers of organisms/genotypes). We observed that LSH-SNN performs comparably or better on these datasets than the two other clustering algorithms tested (LSH-JP and *K*-means/MetaCluster). We should note, however, that, even though LSH-SNN significantly increases the size of the datasets that can be handled as compared with what can be achieved with the SNN method alone, its complexity does not compare favorably with Lloyd's heuristic underlying most *K*-means clustering engines. Therefore, the latter is probably more suited to the analysis of very large datasets containing billions of sequences, which are already generated nowadays from complex metagenomics samples (e.g., from soil) using high-throughput sequencing platforms. Alternatively, the LSH-SNN approach could be used to cluster contigs (sets of overlapping sequences) resulting from a preliminary (meta)genome assembly step, instead of being applied to raw (unassembled) reads as in this study.

ACKNOWLEDGMENTS

This work was supported by the French Alternative Energies and Atomic Energy Commission (Commissariat l'Energie Atomique et aux Energies Alternatives), through its transversal program Technologies pour la Santé (MetaTarget project). The authors would like to thank Anestis Gkanogiannis for generating and sharing the semi-synthetic datasets, Dr. Christopher Quince for sharing the STEC outbreak contigs, and Marcel Salanoubat for overall support.

AUTHOR DISCLOSURE STATEMENT

No competing financial interests exist.

REFERENCES

Aggarwal, C.C. 2009. A framework for clustering massive-domain data streams. In *IEEE 25th International Conference on Data Engineering, 2009. ICDE'09*. IEEE, 102–113.

Agrawal, R., Gehrke, J., Gunopulos, D., et al. 1998. Automatic subspace clustering of high dimensional data for data mining applications. *ACM Sigmod Record* 27, 94–105.

Alneberg, J., Bjarnason, B.S., De Bruijn, I., et al. 2014. Binning metagenomic contigs by coverage and composition. *Nat. Methods* 11, 1144–1146.

Ankerst, M., Breunig, M.M., Kriegel, H.-P., et al. 1999. Optics: Ordering points to identify the clustering structure. *ACM Sigmod Record* 28, 49–60.

Berkhin, P. 2006. A survey of clustering data mining techniques. *Grouping Multidimensional Data*. Springer, Berlin, Heidelberg, 25–71.

Berlin, K., Koren, S., Chin, C.-S., et al. 2015. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nat. Biotechnol.* 33, 623–630.

Boydell, O., Landowski, M., Wu, G., et al. 2013. High-throughput continuous clustering of message streams. *Real-World Challenge for Data Stream Mining* 2.

Broder, A.Z. 1997. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*. IEEE, 21–29.

Buchfink, B., Xie, C., and Huson, D.H. 2015. Fast and sensitive protein alignment using diamond. *Nat. Methods* 12, 59–60.

Buhler, J. 2001. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics* 17, 419–428.

Charikar, M.S. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thirty-Fourth Annual ACM symposium on Theory of Computing*. ACM, 380–388.

Cheeseman, P., and Stutz, J. 1996. Bayesian classification (autoclass): Theory and results. *In* Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., eds. *Advances in Knowledge Discovery and Data Mining*. AAAI, Menlo Park, CA, 153–180.

Dasgupta, A., Kumar, R., and Sarlós, T. 2011. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 1073–1081.

Datar, M., Immorlica, N., Indyk, P., et al. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*. ACM, 253–262.

Dempster, A.P., Laird, N.M., and Rubin, D.B. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)* 39, 1–38.

Ertöz, L., Steinbach, M., and Kumar, V. 2003. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM International Conference on Data Mining*. SIAM, 47–58.

Ertöz, L., Steinbach, M., and Kumar, V. 2004. Finding Topics in Collections of Documents: A Shared Nearest Neighbor Approach. *Clustering and Information Retrieval. Network Theory and Applications*, 11. Springer, Boston, MA.

Ester, M., Kriegel, H.-P., Sander, J., et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*. AAAI Press, 226–231.

Girotto, S., Pizzi, C., and Comin, M. 2016. Metaprob: Accurate metagenomic reads binning based on probabilistic sequence signatures. *Bioinformatics* 32, i567–i575.

Gkanogiannis, A., Gazut, S., Salanoubat, M., et al. 2016. A scalable assembly-free variable selection algorithm for biomarker discovery from metagenomes. *BMC Bioinform*. 17, 311.

Gowda, K.C., and Krishna, G. 1978. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognit*. 10, 105–112.

Guha, S., Rastogi, R., and Shim, K. 1998. Cure: An efficient clustering algorithm for large databases. *ACM Sigmod Record* 27, 73–84.

Har-Peled, S., Indyk, P., and Motwani, R. 2012. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.* 8, 321–350.

Hartigan, J.A., and Wong, M.A. 1979. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28(1), 100–108.

Hastie, T., Tibshirani, R., and Friedman, J. 2009. Overview of supervised learning. *The elements of statistical learning*. Springer New York, 9–41.

Haveliwala, T., Gionis, A., and Indyk, P. 2000. Scalable techniques for clustering the web. In *Proceedings of the WebDB Workshop*, 129–134.

Hinneburg, A., and Keim, D.A. 1998. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* 98, 58–65.

Holtgrewe, M. 2010. Mason—A Read Simulator for Second Generation Sequencing Data. Technical report FU Berlin.

Huang, Z. 1997. Clustering large data sets with mixed numeric and categorical values. In *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. Singapore, 21–34.

Huang, Z. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data Mining and Knowledge Discovery* 2. Kluwer Academic Publishers, 283–304.

Hubert, L., and Arabie, P. 1985. Comparing partitions. *J. Classif.* 2, 193–218.

Indyk, P., and Motwani, R. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, ACM, 604–613.

Jain, A.K., and Dubes, R.C. 1988. *Algorithms for Clustering Data.* Prentice-Hall, Inc.

Jarvis, R.A., and Patrick, E.A. 1973. Clustering using a similarity measure based on shared near neighbors. *IEEE Trans. Comput.* 100, 1025–1034.

Karypis, G., Han, E.-H., and Kumar, V. 1999. Chameleon: Hierarchical clustering using dynamic modeling. *Computer* 32, 68–75.

Katayama, N., and Satoh, S. 1997. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *ACM Sigmod Record* 26, 369–380.

Kaufman, L., and Rousseeuw, P. 1987. *Clustering by Means of Medoids*. Elsevier North Holland.

Koga, H., Ishibashi, T., and Watanabe, T. 2004. Fast hierarchical clustering algorithm using locality-sensitive hashing. *Discovery Science.* Springer, Berlin, Heidelberg,114–128.

Kohonen, T. 2001. Self-Organizing Maps. *Springer Series in Information Sciences*, 30. Springer, Berlin, Heidelberg.

Kotsiantis, S., and Pintelas, P. 2004. Recent advances in clustering: A brief survey. *WSEAS Transactions on Information Science and Applications* 1, 73–81.

Kriegel, H.-P., Kröger, P., and Zimek, A. 2009. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 3 (1).

Leskovec, J., Rajaraman, A., and Ullman, J.D. 2014. *Mining of Massive Datasets*. Cambridge University Press.

Liao, W.-k., Liu, Y., and Choudhary, A. 2004. A grid-based clustering algorithm using adaptive mesh refinement. In *7th Workshop on Mining Scientific and Engineering Datasets of SIAM International Conference on Data Mining*, 61–69.

Lv, Q., Josephson, W., Wang, Z., et al. 2007. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*. VLDB Endowment, 950–961.

McLachlan, G., and Krishnan, T. 2007. *The EM Algorithm and Extensions*, 382. John Wiley & Sons.

Moëllic, P.-A., Haugeard, J.-E., and Pitel, G. 2008. Image clustering based on a shared nearest neighbors approach for tagged collections. In *Proceedings of the 2008 International Conference on Content-Based Image and Video Retrieval.* ACM, 269–278.

Ounit, R., Wanamaker, S., Close, T.J., et al. 2015. Clark: Fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics* 16, 236.

Patidar, A.K., Agrawal, J., and Mishra, N. 2012. Analysis of different similarity measure functions and their impacts on shared nearest neighbor clustering approach. *International Journal of Computer Applications* 40, 0975–8887.

Paulevé, L., Jégou, H., and Amsaleg, L. 2010. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognit. Lett.* 31, 1348–1358.

Rasheed, Z., Rangwala, H., and Barbara, D. 2012. Efficient clustering of metagenomic sequences using locality sensitive hashing. In *Proceedings of the 2012 SIAM International Conference on Data Mining*. SIAM, 1023–1034.

Rokach, L., and Maimon, O. 2005. Clustering methods. *Data Mining and Knowledge Discovery Handbook*. Springer, Boston, MA, 321–352.

Rosenberg, A., and Hirschberg, J. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* 7, 410–420.

Sheikholeslami, G., Chatterjee, S., and Zhang, A. 1998. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB* 98, 428–439.

Steinbach, M., Ertöz, L., and Kumar, V. 2004. The challenges of clustering high dimensional data. *New Directions in Statistical Physics*. Springer, Berlin, Heidelberg, 273–309.

Tanaseichuk, O., Borneman, J., and Jiang, T. 2012. Separating metagenomic short reads into genomes via clustering. *Algorithms Mol. Biol.* 7, 27.

Wang, J., Shen, H.T., Song, J., et al. 2014a. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927.*

Wang, W., Yang, J., Muntz, R., et al. 1997. Sting: A statistical information grid approach to spatial data mining. In *VLDB* 97, 186–195.

Wang, Y., Leung, H.C.M., Yiu, S.M., et al. 2014b. Metacluster-ta: Taxonomic annotation for metagenomic data based on assembly-assisted binning. *BMC Genomics* 15, S12.

Wood, D.E., and Salzberg, S.L. 2014. Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biol.* 15, R46.

Wu, J. 2012. The uniform effect of k-means clustering. *Advances in K-means Clustering*, Springer Theses. Springer Berlin Heidelberg, 17–35.

Wu, Y.-W., Tang, Y.-H., Tringe, S.G., et al. 2014. Maxbin: An automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome* 2, 26.

Yang, B., Peng, Y., Leung, H., et al. 2010. Metacluster: Unsupervised binning of environmental genomic fragments and taxonomic annotation. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*. ACM, 170–179.

Yeung, K.Y., Fraley, C., Murua, A., et al. 2001. Model-based clustering and data transformations for gene expression data. *Bioinformatics* 17, 977–987.

Yeung, K.Y., and Ruzzo, W.L. 2001. Details of the adjusted rand index and clustering algorithms, supplement to the paper: an empirical study on principal component analysis for clustering gene expression data. *Bioinformatics* 17, 763–774.

Zhang, T., Ramakrishnan, R., and Livny, M. 1996. Birch: An efficient data clustering method for very large databases. *ACM Sigmod Record* 25, 103–114.