kNNVWC: An Efficient k-Nearest Neighbours Approach based on Various-Widths Clustering

Article *in* IEEE Transactions on Knowledge and Data Engineering · January 2015

DOI: 10.1109/TKDE.2015.2460735

Abdul Mohsen Almalawi, Adil Fahad, Zahir Tari, Muhammad Aamir Cheema, Ibrahim Khalil:
kNNVWC: An Efficient k-Nearest Neighbors Approach Based on Various-Widths Clustering. IEEE Trans. Knowl. Data Eng. 28(1): 68-81 (2016)

kNNVWC: An Efficient k-Nearest Neighbors Approach Based on Various-Widths Clustering

Abdul Mohsen Almalawi, Adil Fahad, Zahir Tari, *Member, IEEE*,
Muhammad Aamir Cheema, and Ibrahim Khalil

Abstract—The k-nearest neighbor approach (k-NN) has been extensively used as a powerful non-parametric technique in many scientific and engineering applications. However, this approach incurs a large computational cost. Hence, this issue has become an active research field. In this work, a novel k-NN approach based on various-widths clustering, named kNNVWC, to efficiently find k-NNs for a query object from a given data set, is presented. kNNVWC does clustering using various widths, where a data set is clustered with a global width first and each produced cluster that meets the predefined criteria is recursively clustered with its own local width that suits its distribution. This reduces the clustering time, in addition to balancing the number of produced clusters and their respective sizes. Maximum efficiency is achieved by using triangle inequality to prune unlikely clusters. Experimental results demonstrate that kNNVWC performs well in finding k-NNs for query objects compared to a number of k-NN search algorithms, especially for a data set with high dimensions, various distributions and large size.

Index Terms—Clustering, K-nearest neighbour, high dimensionality, performance, SCADA

1 Introduction

Given a set of objects O and a query object q, a k-nearest neighbor (k-NN) query returns from O the k closest (most similar) objects to q. For example, in an image database, a user might be interested in finding the images most similar to a given query image. k-NN is a classical problem and has applications in a wide range of domains such as pattern recognition [1], outlier detection [2], [3], [4], [5], intrusion detection [6], [7], classification [8], [9], [10], and spatial databases, to name a few.

The problem of k-NN has been extensively studied in the past and numerous algorithms have been proposed to compute *approximate* [11], [12], [13], [14], [15] or *exact* [16], [17], [18], [19], [20], [21], [22], [23], [24], [25] results, depending on the needs of the applications and/or the users. The approximate algorithms aim at achieving higher efficiency at the cost of quality (i.e., accuracy) of the results. In contrast, although the exact algorithms are more expensive, they provide exact results. The focus of this paper is on computing the exact k-nearest neighbors on high dimensional data sets.

The traditional approach to compute exact results, the called Exhaustive k-NN (Ek-NN) approach in this paper, requires scanning the whole data set and finds k-NNs by computing the distance between q and every object in O. This results in high computational cost. To address this, a large body of research has focused on pre-processing the data set (i.e., constructing an index) with an aim to compute k-NNs by accessing only a part of the data set. The techniques can be loosely classified into two categories: i) tree-based indexes; ii) flat indexes. Below, we briefly describe both.

Some of the most popular tree-based indexes include kd-tree [18], M-tree [22], vantage-point tree (vp-tree) [24], [25], multi-vantage point tree (mvp-tree) [23], Cover tree [21] and Ball-tree [16]. The tree-based indexes use a binary partitioning technique to build a tree of the data sets. Each leaf node of the tree contains the objects that are close to each other. The set of leaf nodes are grouped by a set of intermediate nodes where each node contains the leaf nodes that are close to each other. This procedure is recursively applied until all the nodes are contained by a single node (i.e., the root node). The k-NN algorithm exploits the tree index and uses the triangle inequality to prune the nodes that cannot contain the results.

Each tree-based index has its own different technique for partitioning a data set in a recursive fashion to build a tree of the data set. The major drawback of the tree-based indexes is that a binary partition technique is adopted to build the tree-structure, and this separation may not be the appropriate way for the object distributions especially when the data distribution is unknown and the dimensionality is high. Moreover, The high dimensional data appear to be sparse and dissimilar. Hence, the internal nodes will have a high overlap with each other [26] . As a consequence, the triangle inequality-based pruning fails to prune most of the intermediate nodes and the k-NN algorithms usually end up visiting the whole data set (or a large part of the data set). As discussed later in Section 2.2.2, the binary partition technique nullifies the effectiveness of the hierarchical structure of the tree.

- *A.M. Almalawi is with the School of Computer Science & Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia. E-mail: balmalowy@kau.edu.sa.*
- *A. Fahad is with the College of Computer Science & Information Technology, Department of Computer Science, Al Baha University, Al Baha, Saudi Arabia. E-mail: afalharth@bu.edu.sa .*
- *Z. Tari and I. Khalil are with the Distributed Systems and Networking (DSN) Discipline, School of Computer Science and Information Technology (CSIT), RMIT University, Melbourne, Vic 3000, Australia. E-mail: zahir.tari@cs.rmit.edu.au, brahimk@cs.rmit.edu.*
- *M.A. Cheema is with the Faculty of Information Technology, Monash University, Clayton, Vic. 3168, Australia. E-mail: aamir.cheema@monash.edu.*

As discussed above, the tree structure loses its effectiveness when the overlap between the nodes is high. Inspired by this, flat-indexes use the clustering techniques to directly partition the the data set into a number of clusters to obtain better partitioning results, instead of creating a tree-like structure, and use the clusters to efficiently compute k-NNs. Fixed-width clustering (FWC) [27], [28] and k-means [29] are two such approaches where the feature space is directly partitioned into a number of clusters. The centers and radii of the clusters are used to compute upper and lower bound distances (using triangle inequality) between the query and the objects in the clusters.

The performance of flat-index based approaches depends on the quality of clusters, e.g., radius and the number of objects in each cluster. This is because the compact and well-separated clusters result in increased effectiveness of the triangle inequality-based pruning which improves the efficiency of the k-NN algorithm. We note that the existing approaches may fail at producing the high quality clusters which adversely affects the performance of k-NN algorithm. For example, k-means assigns sparse objects to the closest cluster, and therefore some of the clusters may have an overwhelmingly large radius (resulting in overlapping with a lot of other clusters). This reduces the effectiveness of the triangle inequality. Similarly, FWC fixes the width of each cluster, i.e., each cluster has the same radius. Depending on the distribution, this may result in some clusters that have a very large number of objects and some clusters that have a very few objects. This adversely affects the performance because, if a cluster having a large number of objects cannot be pruned, the algorithm needs to compute the distance between q and all the objects in the cluster.

In this paper, we address the aforementioned mentioned limitations of the flat indexes and present a novel k-Nearest Neighbour approach based on Various-Widths Clustering (kNNVWC). Specifically, the proposed approach produces high quality clusters that improve the performance of k-NN algorithms. Our main contributions are the following:

1) We propose a novel partitioning clustering algorithm that aims to produce compact and well-separated clusters from high dimensional data of various distributions. In addition to the quality of decomposition results, the proposed algorithm shares useful features of tree-based and flat indexes. Firstly, no sensitive parameters are required, and it requires only a maximum cluster size parameter which, like the leaf size parameter in trees, can be easily determined from the size of the data set. Secondly, similar to flat indexes approach (e.g. k-means and FWC), the feature space is directly partitioned into a number of clusters to obtain better partitioning results. Finally, unlike FWC, various radii are automatically learned and used to efficiently partition the feature space of various distributions.
2) The effectiveness of kNNVWC is evaluated in comparison with the flat-index based approaches, namely, k-means and FWC and the well-known tree-based spatial algorithms such as KD-tree [18], Ball tree [16], Cover tree [21] and VP-tree [24].
3) We conduct an extensive experimental study using eleven benchmark data sets and one synthetic data set. The results demonstrate the advantages of our algorithm in terms of construction cost and the querying cost as compared to the most popular tree-based and flat indexes such as KD-tree, Ball-tree, Cover-tree, VP-tree, k-means and FWC.

The rest of the paper is organized as follows: Section 2 summarizes some of the most popular and effective k-NN search methods for high dimensional data. Section 3 describes kNNVWC in detail. The performance of kNNVWC in comparison with the other algorithms on a variety of data sets is reported and discussed in Section 4. Finally, we conclude the work in Section 5

2 Related Work

In the absence of an index, the whole data set needs to be scanned to compute the k nearest neighbors of a given query. A large body of research has focused on constructing an index to avoid scanning the whole data set. We first broadly categorise the existing work into two categories: *approximate* and *exact* k-NN search algorithms.

2.1 Approximate Search

The approximate k-NN search algorithms are well-known for their high performance in high dimensional data; however, they are less accurate because non-optimal neighbours are expected to be returned. The locality-sensitive hashing (LSH) method, that uses several hash functions, is one of the most interesting hash-based approaches [11]. This approach is based on the assumption that the close objects are likely to have hashes that are close to each other. Variations of LSH have been proposed such as multi-probe [13] LSH and LSH Forest [12]. The former reduces the number of hash tables in order to improve the high storage costs, while the latter removes the necessity of hand-tuning parameters. Recently, several space-partitioning structure-based algorithms such as the randomized k-d trees [14] and the priority search k-means tree [15] have been found to be more promising compared to the LSH algorithms in approximate nearest-neighbor searches.

2.2 Exact Search

Of particular interest is the problem of quickly finding the "exact" k-NN for a query object in a large and high dimensional data set using metric distance functions that satisfy the triangle inequality property. We loosely categorize these techniques into *tree* based and *flat* indexes.

2.2.1 Tree-Based Indexes

In this section, we briefly describe some of the most notable tree-based indexes.

KD-tree. Friedman et al. [30] introduced KD-tree to structure the data set in a balanced binary-tree, where the data set is recursively split into two parts along one axis (dimension). Variations of the KD-tree were proposed by Sproull [18] and Kim and Park [20]. However, none of these is appropriate for data that has more than 10 dimensions [17]. Unlike the variations of the KD-tree that use axis-

aligned partitioning hyperplanes, binary clustering techniques have been proposed instead to deal with the high dimensional data.

Ball-tree. Fukunaga and Narendra [16] proposed a metric tree algorithm called the Ball tree, to organize objects in a metric space into a hierarchy of clusters using binary partitioning clustering technique. Ball tree computes the centroid of the whole data set and then the centroid is used to partition the data set into two subsets. The farthest object from this centroid is assigned as the centroid of the first subset, while the farthest object from the centroid of the first subset is assigned as the centroid of the second subset. These steps are recursively applied until the entire data set is organized in a binary tree.

Omohundros [31] proposed variations of the Ball tree using five portioning techniques to produce good decomposition results. The number of objects assigned to either node cannot be constrained which may result in a highly unbalanced tree structure [32].

M-tree. Leaf nodes of any M-tree [22] store all data points, whereas the internal nodes store the so-called *routing objects*. All the objects that are within a given distance r from the routing object are stored in the sub-tree of the routing object called *covering tree*. Each routing object is also associated with its distance to the the routing object of its parent node. The triangle inequality based on these distances is used to quickly prune the search space.

Cover tree. Beygelzimer et al. [21] proposed cover tree which is a hierarchy of levels with the top level containing the root point and the bottom level containing every point in the metric space. Each level in the tree must satisfy *covering*, *nesting* and *separation* properties. It was shown that the cover trees can answer nearest neighbor queries in logarithmic time to the data set size multiplied by some constant. However, the constant depends on the actual data distribution and dimensionality and may be arbitrarily large [21].

vp-tree. In contrast to Ball tree, vp-tree partitions the data set based on the absolute distance from a single center, where a vantage point, which has the highest distance deviation, is chosen. Then, the median μ of all distances between objects and the chosen vantage point is calculated. The objects whose distances to the vantage point is less than μ are assigned to the left node, while the remaining ones are assigned to the right node. This is recursively preformed until the entire data set is organized as a balanced tree. Due to the recursive partitioning based on distances from the vantage points, the objects in the leaf nodes may be far from each other and overlap other nodes especially in a high-dimensional data [33].

A variant of vp-tree, called mvp-tree [23], uses two vantage points in each node. The first point is to divide the feature space into two parts, while the second point is to divide each resultant part into two. Interestingly, in [34] it is shown by experiments that vp-tree performs much better than M-tree and R^*-tree in finding k-NNs for query object. However, the results demonstrate that mvp-tree performs better but not significantly so. Overall, although these algorithms produce promising results with high dimensional data, their performances vary from one data set to another. As has been found in [35], this issue can be attributed to the different distance distributions of each data set.

2.2.2 Flat-Based Indexes

As noted in [29], tree-based indexes perform good if the search for nearest neighbors involves only a single (or a few) nodes at each level of the tree, i.e., for binary search tree, roughly $O(\log n)$ nodes will be visited. However, if the search involves most of the nodes in each level, then the tree structure loses its effectiveness and appeal because a large number of leaf nodes are required to be visited. This is especially true for high dimensional data because the triangle inequality fails to prune nodes due to the curse of dimensionality. Hence, in such cases, the indexes that do not use a hierarchical (tree-based) structure and use the non-binary partition technique (clustering technique) may perform better. This is because binary partition technique may not be the appropriate way for the object distributions and may result in an unbalanced and deep tree structure which requires unnecessary computation in walking through the internal/leaf nodes. For a more detailed discussion on this, please see [29].

Inspired by the above, several flat indexes, which use non-binary partition techniques, have been proposed for k-NN queries on high dimensional data sets, e.g., k-means [29] and fixed-width clustering [27], [28]. Flat indexes cluster the objects in the data set into a number of clusters. Then, based on the centers and radii of the clusters, the triangle inequality can be applied to prune the clusters that cannot contain the results. The performance of flat indexes depends on the quality of clusters. That is, the compact and well-separated clusters result in increased effectiveness of the triangle inequality in filtering out the objects that are far way from a query object.

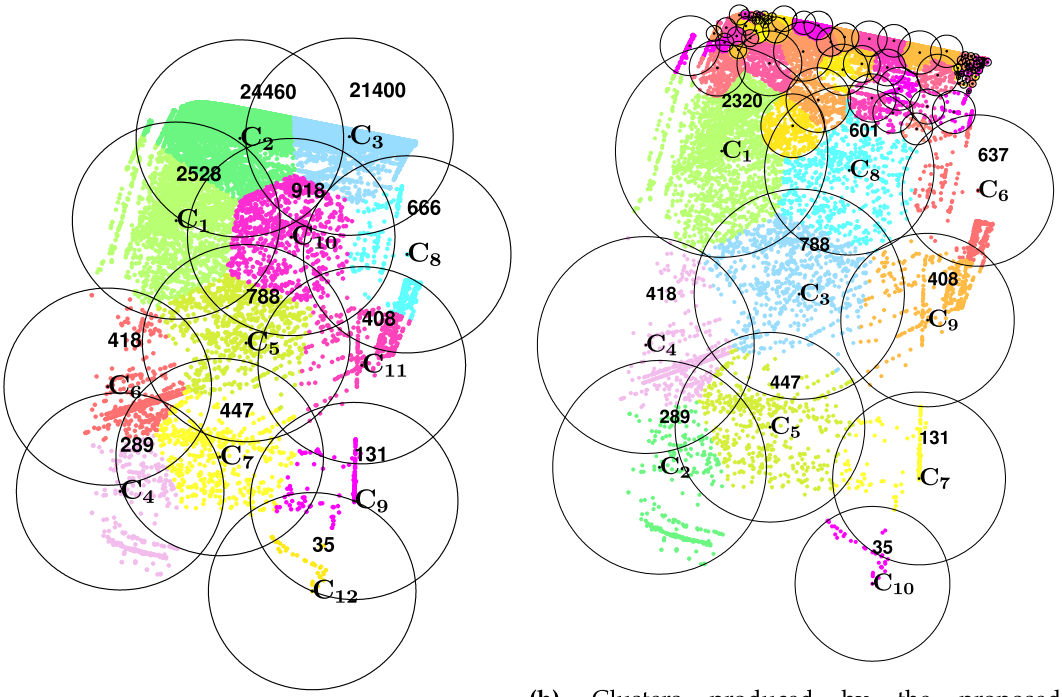
We observe that the existing approaches may be unable to produce high quality clusters For example, k-means assigns sparse objects to the closest cluster, and therefore some of the clusters may have an overwhelmingly large radius (resulting in overlapping with a lot of other clusters). A major limitation of FWC is that each cluster has the same width w where w is a parameter to the index construction algorithm. As we show later in Section 3, this results in poor clustering because the number of objects in some cluster may be huge and the k-NN algorithm may incur a very high cost if such cluster cannot be pruned. Motivated by this, we propose a flat index that uses different widths for different clusters and the width of each cluster is learned during the construction. The experimental study demonstrates that our proposed flat index significantly outperforms k-means, FWC and the tree-based indexes.

3 Proposed Approach

The clustering steps of FWC algorithm and its inability to cluster a data set that has various distributions are discussed here. Then, we introduce the two main parts of kNNVWC: the various-widths clustering part and the exploitation of triangle inequality to efficiently find k-NNs for a query object at a reasonable cost throughout the partitioned clusters.

3.1 FWC Algorithm and Its Limitations

The FWC algorithm is a simple and efficient technique for partitioning a data set into a number of clusters with a fixed radius w.



(a) Clusters produced by FWC

(b) Clusters produced by the proposed kNNVWC approach

Fig. 1. Clustering of the two first principal components of a sample of the original DARPA data set.

Algorithm 1 summarizes the steps of the FWC, where one pass is required to partition a data set. Let $U = \{j_1, j_2, \ldots, j_n\}$ be a set of objects, and $C = \{C_1, C_2, \ldots, C_n\}$ be the produced clusters from U using FWC. Initially, the first object j_1 is used to create the first cluster C_1, and it is also set as its centroid (Lines 7-8). Then, the next object j_i is pulled from U and assigned to the closest cluster C_i providing that the distance between j_i and $C_i \leq w$ and $n > 0$, where w is the cluster fixed-width, and n is the number of clusters (Lines 10-14). Otherwise, a new cluster is created C_{i+1} and j_i is set as its centroid (Lines 16-17). This will continue until all objects in U have been scanned.

The nature of FWC constrains its use in outlier detection [36], [37] and breaking down the data set into fixed-width clusters [27] where these clusters, which are created based on fixed-radius, may not be meaningful. Similar to our work, Eskin et al. [27] proposed a flat-index based approach using the FWC as a means of breaking down the data set into smaller and adjacent clusters. Then, this approach to efficiently find k-NNs for a query object, uses the centers and radii of these clusters and triangle inequality to remove the necessity of scanning all objects in a data set. However, the key issue of using the FWC in breaking down the data set can be illustrated in the following example.

Consider the example given in Fig. 1. These are the first two principal components of a sample of the original DARPA data set [38]. This sample contains 52,488 objects. 241 objects are labelled as attacks while the rest are labelled as normal. To find the five nearest neighbours for an object X_i, all the objects $n-1$ (52,487 objects) must be checked although only five objects are required. Fig. 1a shows the importance of using FWC to remove the necessity of checking all objects. For instance, the 5-NNs for an object X_i in the cluster C_{12} can be found in clusters C_{12}, C_9, C_7, and C_4. However, the efficiency of this algorithm is decreased if some of the produced clusters are relatively large. For instance, the clusters C_3 and C_2, which are adjacent, encompass more than 87 percent of the data set. Hence, finding the 5-NNs of any object in these clusters requires checking more than 87 percent of the objects.

Algorithm 1. Fixed-Width Clustering

```
1  Input: U
   /* A list of objects,                                          */
2  Input: w
   /* The predefined radius of a cluster                          */
3   C ← ∅; C^r ← ∅; C^w ← ∅; /* A set of clusters and
    their centroids and widths                                    */
4  n = 0; /* The number of created clusters                       */
5  foreach j_i in U.objects do
6      if n==0 then
7          n+ = 1;
8          put j_i in C_n; put j_i in C^r_n; C^w_n      0;
9      else
10         < id, dis > ⟵ ClosestCluster (j_i, C^r);
            /* Find the ID and distance of the closest
            cluster to j_i                                        */
11         if dis ≤ w then
12             put j_i in C_id;
13             if C^w_id < dis then
14                 C^w_id     dis;
15         else
16             n+ = 1;
17             put j_i in C_n; put j_i in C^r_n; C^w_n      0;
18 return [C, C^r, C^w];
```

Setting a fixed-width parameter with a small value can prevent large clusters from being produced. However, this will

result in the following issues: (i) a large number of clusters can be created that increases the computation time of the FWC, which is $\mathcal{O}(nc)$ where n the number of objects and c the number of clusters. This is mainly due to the sparsely distributed objects in n-dimensional space, contributing to the creation of a large number of clusters with very few members in each. This fact is highlighted in Fig. 1a where the data objects in clusters C_8, C_4 and C_9 show a very sparse distribution. (ii) The large number of created clusters also increases the computation time of the search for k-NNs (see Section 3.3). The two main elements of kNNVWC, namely *various-widths clustering* and *k-NN search*, are elaborated in the following.

3.2 Various-Widths Clustering

In this section, the various-widths clustering part is presented, where a data set is partitioned into a number of clusters whose sizes are constrained by the user-defined threshold. Three processes are involved in this operation: *cluster-width learning*, *partitioning* and *merging* and they are looped sequentially and executed until the criteria are met. That is, the processes of partitioning and merging are stopped when the size of the largest cluster is less than a user-defined threshold β, or when the number of clusters prior to and after the execution of the mentioned processes are equal to β. Algorithm 2 shows the steps of the mentioned processes, and the variables, data structures, functions employed by the algorithm are summarized in Table 1.

3.2.1 Cluster-Width Learning

Let D be a data set to be clustered, and $NN_k(H_i)$ be the function of k-nearest neighbours for the object H_i, $clsWidth$ be the function computing the width (radius) of $NN_k(H_i)$, where the width is the distance between the object H_i and the farthest object among its neighbours. The value of k is set to $50\% \times |D|$ to guarantee a large cluster. To find the appropriate global width, we randomly draw a few objects from D, $H = \{H_1, H_2, \ldots, H_r\}$ where $r \ll |D|$, and for each randomly selected object, the radius of its k-nearest neighbours is computed, and the average is used as a global width for D as follows:

$$w = \frac{1}{r}\sum_{i=1}^{r} clsWidth(NN_k(H_i), H_i)). \quad (1)$$

3.2.2 Partitioning Process

This process partitions a data set into a number of clusters using a large width to resolve the issue of clustering the sparsely distributed objects in n-dimensional space. However, large clusters from dense areas will be created such as clusters C_2 and C_3 in Fig. 1a. Therefore, each large cluster whose size exceeds a user-defined threshold (maximum cluster size) will be divided into a number of clusters using a width that suits the density of that cluster. This process continues until the sizes of all clusters are less than or equal to the user-defined threshold. Fig. 1b illustrates the produced clusters using various-widths clustering, as can be seen in Fig. 1b, where large clusters are partitioned into a number of smaller clusters.

The main steps of this process are summarized in the procedure for *Partitioning* in Algorithm 2. This procedure has two variables: $Clusters$ and β. The former is a list of class objects, where each object contains members and properties of a cluster. In the initial step, the whole data set is considered as a cluster and its centroid and width is set with zeros (Line 4). The latter variable is the threshold of the maximum cluster's size. Then, the function *LargestCluster* returns the largest cluster U, that is not assigned

TABLE 1
Variables, Data Structures, and Functions Employed by kNNVWC

Algorithm 2: Various-widths clustering	
$Data$	A data set need to be partitioned
$Clusters$	A list of objects, each object consists of two numeric variables (width and ID) and two arrays that contain members (instances) and centroid of cluster respectively
β	The maximum cluster size, where any cluster that exceeds this threshold will be further partitioned
w	The predefined radius of cluster. The value of this parameter learned from the data set in (Eq 1)
$LargestCluster$	This function returns the largest cluster among the set of created clusters
$finished$	A variable that represents the status of various widths clustering. 0 indicates either further partitioning or merging process is required
$Partitioning$	A procedure that partitions a data set into a number of clusters using various widths, where the size of each cluster does not exceed the predefined threshold β
$Merging$	A procedure that is responsible for merging the child clusters with their parents
$isParent$	This function is used to check that the child cluster, which will be merged with its parent, is not a parent cluster for others
$MergeClus$	A function that associates the members of a cluster C_i with another C_j and removes cluster C_i from the list of clusters $Clusters$
Algorithm 3: Search for k-NNs throughout partitioned clusters.	
$Clusters$	A list of objects, each object consisting of two numeric variables (width and ID) and two arrays that contain members (objects) and centroid of cluster respectively
k	The number of nearest neighbours that need to be found
N	A list of IDs of objects and their distances to an object p
Z	An array of IDs of candidate clusters for an object p
NN_k	This function returns k-nearest neighbors of an object p in data set d, where $p \notin d$
$ClusAscOrder$	This function returns a list of IDs of clusters and their distances from the current query object. This list is sorted by the distances in ascending order

as non-partitionable, from $Clusters$ (Line 14). If the size of U is greater than (or equals) β, Equation (1) is used to calculate an appropriate width w for partitioning U. If the value of w equals zero, U is assigned as non-partitionable (Lines 15-20). This is because the objects in U have similarities in terms of the distance function, and therefore they cannot be partitioned. Otherwise, Algorithm 1 is called to partition U (Line 21). If the number of produced clusters is just one, the value of w is very large and it should be minimized by 10% and used again (Line 27). Otherwise, the new clusters produced from U are added to $Clusters$ instead of U, and the largest cluster again is pulled from $Clusters$ (Lines 22-25). The steps (Lines 15-27) are repeated until the partitionable largest cluster in $Clusters$ is less β.

Algorithm 2. Various-Width Clustering

```
1  Input: Data
2  Input: β
3  Output: Clusters
4  Clusters ← ϕ; add(Clusters, [Data, zeros, 0]);
5  finished ← 0;
6  while finished == 0 do
7      ClsSize ← Clusters.getSize /*
         The number of clusters */
8      Partitioning(Clusters, β);
9      Merging(Clusters, β);
10     if |LargestCluster(Clusters)| ≤ β or
       Clusters.getSize==ClsSize then
11     finished ← 1
12 return [Clusters];
13 Procedure Partitioning (Clusters, β)
14     U ← LargestCluster(Clusters);
15     while |U.objects| > β do
16       w ← using eq. (1);
17       if (w == 0) then
18         U.nonPartitioned(1);
19         update(Clusters, U);
20         continue;
21       < tmpClusters > ← Algorithm 1(U, w);
22       if ClusterNum (tmpClusters) > 1 then
23         remove(Clusters, U);
24         add(Clusters, tmpClusters);
25         U ← LargestCluster(Clusters);
26       else
27         w ← w − (w ∗ 0.1);
28         go to line 21
29 Procedure Merging(Clusters, β)
30     MergingList ← ϕ /* list of tuples < */
         /* childClusterID, parentClusterID> */
31     foreach U in Clusters do
32       j ← using eq. (2) and eq. (3);
         /* ID of cluster contained U */
33       if j ≠ 0 then
34         put < U.getID, j > in MergingList;
35     while MergingList ≠ ϕ do
36       foreach tuple in MergingList do
37         < i, j > ← tuple;
38         if !isParent(MergingList, i) then
39           MergeClus(Clusters, i, j);
40           remove tuple from MergingList;
```

3.2.3 Merging Process

The partitioning process of a large cluster can lead to the creation of a cluster that is totally contained in another cluster. Therefore, merging such clusters decreases the number of clusters produced, thereby increasing the performance of the search for k-NNs because the distance computations between a query object and the clusters' centroids will be less when fewer clusters exist. Fig. 1a shows that many objects of the cluster C_2 are located within the range of cluster C_1, but they are associated with the cluster C_2 because it is the closest cluster. To partition the cluster C_2 (as it is large) into a number of clusters, the objects that are located within the range of the cluster C_1 might form a new cluster inside the cluster C_1. Therefore, the merging process is proposed to address this potential problem. Let $C = \{C_1, C_2, \ldots, C_n\}$ be the created clusters. The cluster C_i is contained by the cluster C_j, if the following criteria are met:

$$\begin{cases} D(C_i, C_j) + w_i \leq w_j & C_i \subset C_j, i \neq j, \\ Otherwise & C_i \not\subset C_j, \end{cases} \tag{2}$$

where the $D(C_i, C_j)$ is the distance function between the centroids of clusters i and j, where w_i and w_j are the radii of clusters i and j respectively. C_i might be contained by a number of clusters $C = \{C_1, C_2, \ldots, C_f\}$. Therefore, C_i is merged with the closest C_j that is defined as follows.

$$C_j = \min_{j=1}^{f} D(C_i, C_j). \tag{3}$$

The final step of the merging process involves merging C_i with C_j while C_i is a parent of C_{j+1}. This is undertaken quite simply by firstly merging the child clusters (that are not parents of other clusters) with their parents. The procedure *Merging* in Algorithm 2 summarizes the steps of the merging process. The list of all clusters $Clusters$ is iterated to obtain the IDs of child clusters and their respective parents (Lines 12-34). In the next step, all members (objects) of each child cluster are associated with its parent cluster, and the object of this child cluster is removed from the list $Clusters$. To avoid removing any child cluster that might be a parent of other clusters, bottom-up merging is used, where the child clusters that are not parents of any clusters (like leaf nodes in tree structures) are merged first (Lines 35-40).

3.3 The k-NN Search

In this section, we present our k-NN processing algorithm that follows the structure of existing k-NN algorithms such as [39], [40]. Specifically, our algorithm contains the following steps: (i) building initial k-NN candidates by including data points from the nearest clusters which minimally contain k data points (lines 8-14); (ii) using the initial k-NN candidates to prune clusters containing inadmissible data points (lines 16-21); (iii) comparing data points in the remaining clusters with the initial candidates and retaining the k-NNs (lines 22-26).

We show how the triangle inequality can be exploited to efficiently find k-NNs using the clusters produced by Algorithm 2. To eliminate clusters that cannot possibly

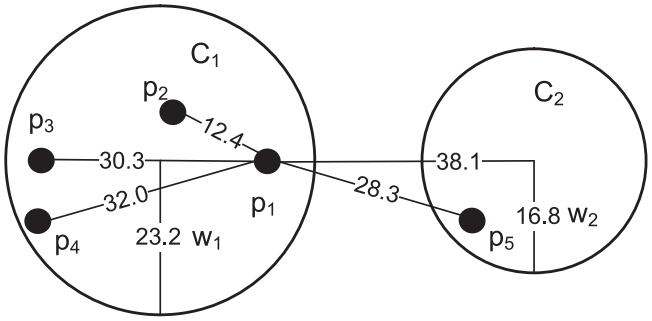


Fig. 2. An illustration of the use of the triangle inequality for searching nearest neighbors.

have k-NNs for a query object q, the distance similarity between each cluster and the query object q must be calculated. Prior to discussing the use of triangle inequality for removing impossible clusters, two definitions are presented first.

Definition 1 (k-nearest neighbors). *The k-nearest neighbors of an object p are objects whose distances from p are less than the remaining objects in the data set. Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n objects in the data set and $D(X_i, X_j)$ is the distance function between pairs of objects. Given $B = \{B_1, B_2, \ldots, B_k\}$ be a set of k-nearest neighbors of the object p where $B \subset X$, $p \in X$ and $p \notin B$. B is k-NNs of p if $\max_{n=1}^{k} D(p, B_n) < \min_{n=1}^{|Z|} D(p, Z_n)$, $Z = X \backslash \{B \cup \{p\}\}$ is held.*

For example, Fig. 2 shows two clusters C_1 and C_2. The 2-nearest neighbors of object p_1 are objects $B = \{p_2, p_5\}$ even though p_5 resides in a different cluster. This is because the object p_5 holds the definition of the 2-nearest neighbors of object p_1: $\max_{n=1}^{2} D(p_1, B_n) < \min_{n=1}^{|Z|} D(p, Z_n)$, $Z = \{p_3, p_4\}$.

Definition 2 (Candidate cluster for an object *p*). *Let $C = \{C_1, C_2, \ldots, C_n\}$ be the set of clusters, and $B_i = \{B_1^i, B_2^i, \ldots, B_k^i\}$ be the k-NNs for an object p in a cluster C_i. The cluster C_j is considered as a candidate cluster for the object p, such that $i \neq j$, if it holds: $D(p, C_j) - w_i < D(p, B_z^i)$, $z = 1, 2 \ldots, k$, where w_i is the radius of the cluster C_j.*

This can be illustrated in Fig. 2. The objects p_2, p_3 and p_4 are the 3-NNs of the object p_1 in the cluster C_1. The cluster C_2 is considered as a candidate cluster for the object p_1 because the distance between the object p_1 and the cluster C_2 minus the radius of this cluster is at least less than one of the distances between the object p_1 and its neighbors p_2, p_3 and p_4. For example, $D(p_1, C_2) - w_2 = (38.1 - 16.8) = 21.3 < D(p_1, p_3) = 30.3$ and $D(p_1, p_4) = 32$. Hence, the candidate cluster C_2 might contain objects that are nearer than p_3 and p_4, which is, in this example, the object p_5, and therefore, the object p_4 is replaced with it. Algorithm 3 summarizes the steps involved in k-NN search. The variables, data structures and functions employed by this algorithm are summarized in Table 1. When a query object q is received, the function *ClusAscOrder* returns a list of clusters' IDs and their respective distances to q, the list is named $cluID$. The returned data is sorted by distances in ascending order (Line 6). A temporal cluster object $tmpU$ is created to represent the closest cluster to q. If the size of $tmpU$ is less than k, the objects of the second closest cluster are merged in $tmpU$. This process continues until the size of $tmpU$ becomes greater than (or equal to) k. Note that all IDs of the joined clusters are removed from the list $cluID$ (Lines 7-13). In this step, $|tmpU| > k$, and therefore, it is initially assumed to have k-NNs for the object q. The function NN_k returns a list T of the IDs of the k-NNs from $tmpU$ alongside their cross-posting distances to the object q (Line 14). In fact, the objects in T cannot be guaranteed as "exact" k-NNs for q. Thus, the candidate clusters, which might contain objects that are nearer than the objects in T w.r.t the object q, have to be defined from the clusters that have not been merged into $tmpU$. This can be done with fast and simple calculations; no distance computation is required because the distances from the object q to all the clusters and the objects in T have already been computed (see Lines 6 and 14). Therefore, any cluster whose distance from the object q is d, and the radius w of this cluster is subtracted from $d = d - w$, the cluster is assigned as a candidate cluster when d is less than the distance between the object q and any assumed k-NN objects in T (Lines 15-21).

Algorithm 3. Search for k-NNs for a Query Object Throughout Clusters

```
1  Input: clusters
2  Input: k
3  Input: q
   /* The query object                                              */
4  Output: N
5  cluID ← φ;
   /* list of tuples < clusterID, distance > sorted
      by distance in ascending order                                */
6  cluID ← ClusAscOrder(clusters, q);
7  tmpU ← φ;
8  foreach {clusterID, distance} in cluID do
9     U ← get(clusters, clusterID);
10    tmpU ← tmpU ⋃ U.objects;
11    remove < clusterID, distance > from cluID;
12    if |tmpU| > k then
13       break;
14 T ← NN_k(p, tmpU, k)
   /* list of tuples < objectID, distance >                         */
15 N  φ; Z  φ;
16 foreach {clusterID, cluDis} in cluID do
17    U  get(clusters, clusterID);
18    foreach {objID, objDis} in T do
19       if (cluDis − U.radius) < objDis then
20          put clusterID in Z;
21          break;
22 foreach clusterID in Z do
23    U  get(clusters, clusterID);
24    put NN_k(p, U.objects, k) in N;
25 N = N ∪ T;
26 return top k-nearest objects from N;
```

To find the "exact" k-NNs for q, the similarity distance between each object in the candidate clusters in Z and the object q is computed. All these objects, in addition to the ones in T, are stored in the list N, and they are sorted by their distances to q in ascending order. The final step is to return the top k objects from N as the "exact" k-NNs for q (Lines 22-26).

4 Experiment Results and Comparison

In this section, we evaluate the performance of the proposed kNNVWC approach on various data sets. Three aspects are considered in this evaluation: (i) the total number of distance computations; (ii) the computational cost of the construction process; and (iii) the search for k-NNs. Seven algorithms are compared with kNNVWC. The q-fold-cross validation is applied to each data set, by dividing it into q equal size subsets. Each time, one of the subsets is used as testing queries, while the remaining $q - 1$ subsets are used as a train data set. Then, the average value of the results of all folds is used as an overall result. In this evaluation, q is set to 10 as suggested by Kohavi [41] in order to reliably demonstrate the efficiency of any proposed algorithm. All algorithms were implemented in Java, and executed using Windows 7 enterprise with 3.4 GHz CPU and 8 GB memory. In the following, all experiment elements are broadly discussed.

4.1 The Data Sets

Twelve data sets with high, medium and low-dimensional feature spaces are used for evaluating the proposed approach. These data sets have been selected because they come from various domains and are widely used for the evaluation of data mining techniques. Ten of them were obtained from the UCI Machine Learning Repository [42], while three come from various other places. Notably, these data sets are intended to be used for the purposes of classification and outlier detection; however, we use them only to evaluate the performance of the proposed approach to searching k-nearest neighbours in various domains and dimensions. The characteristics of these data sets are briefly described below:

- *arcene*: It contains mass-spectra obtained with the SELDI technique. This data set has 900 objects, each being described by 1,000 features that can be used to separate the cancer patients from healthy patients [42].
- *SimSCADA*: This data set consists of 12,000 objects, each being described by 113 features. Each feature represents one sensor or actuator reading in the water network systems, which is simulated using the proposed SCADA testbed SCADAVT in [43].
- *multiplefeaturs*. This data set consists of 2,000 patterns of handwritten numerals ("0"-"9") extracted from a collection of Dutch utility maps,where each pattern is represented by 649 numeric features [42].
- *arrhythmia*: This data set consists of 452 objects, each represented by 279 parameters (features) of ECG measurements. It is used to classify a patient into one of the 16 classes of cardiac arrhythmia [42].
- *gasSensors* This data set contains 13,910 measurements that come from 16 chemical sensors, and each feature vector contains the eight features extracted from each particular sensor. This results in a 128-dimensional feature vector [44], [45].
- *spambase* This data set contains two classes: spam and non-spam e-mails. It consists of 4601 objects (e-mails), each being described by 57 continuous features denoting word frequencies [42].
- *kddcup99*: set comes from the 1998 DARPA Intrusion Detection Evaluation Data [38]
- *waveform*: It consists of 5,000 objects having 40 continuous features, some of which are noise [42].
- *DUWWTP*: It comes from the daily measures of sensors in an urban waste water treatment plant and consists of 527 objects, each represented by 38 features [42].
- *shuttle*: This data set consists of 43,500 objects, each represented by nine numerical features. It is used in the European StatLog project to compare the performances of machine learning, statistical, and neural network algorithms on data sets from real-world industrial areas [42].
- *slices*: This data set consists of 53,500 computed tomography (CT) images scanned from 74 different patients (43 male, 31 female). Each CT image is described by two histograms in polar space from which 384 features are extracted [42].
- *MORD*: This data set consists of 4,690 objects, each represented by two features (e.g.Temperature and Humidity). It is collected from a real wireless sensor network and used for outlier detection purpose [46].

4.2 The Performance Metrics

The key problem of Ek-NN is that it requires a lengthy computation time because each object in the data set must be checked to find the k-NNs for a query object. Therefore, the efficient search for k-NNs might be the main aim of any optimized algorithm. To measure this efficiency, the construction (if a preprocessing step is required) and search times and the number of distance computations should be considered in the evaluation step.

4.2.1 Reduction Rate of Distance Computations

This metric is platform-independent where the number of distance computations is not affected by high/low platform resources, although it is data-dependent. To measure the reduction rate of distance computations (RD) for an algorithm Ω, the Ek-NN is used as the baseline. The number of the distance computations for Ek-NN is fixed for any k, which is $q \times n$, where n is the number of objects in data set and q is the number of query objects. Then, the performance of Ω for finding k-nearest neighbours in m distance computations is calculated as follows:

$$RD = 1 - \frac{m}{q \times n}. \tag{4}$$

4.2.2 Speedup Rate

We evaluate the performance of the algorithms relative to the performance of Ek-NN and report the *speedup rate*. Let t_1 be the running time for Ek-NN and t_2 be the running time for an algorithm Ω. Then, the speed-up rate of the algorithm Ω is t_1/t_2. The speed-up rate reflects how fast (slow) an algorithm is with respect to Ek-NN, i.e., a speedup rate of 2

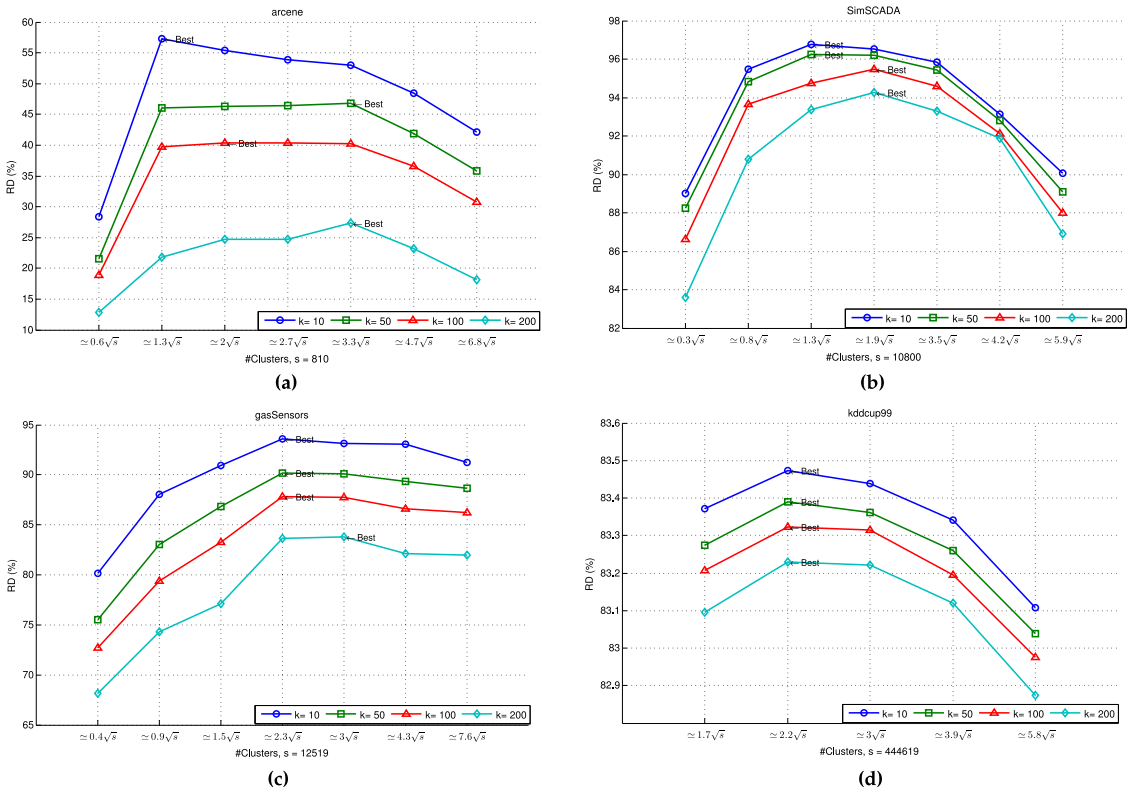


Fig. 3. An investigation of the impact of cluster size, where s influenced by the value of the parameter β, on the performance of the proposed approach, where s is the size of the training data set.

means the algorithm is two times faster and a speed-up rate of 0.5 means the algorithm is two times slower than Ek-NN.

4.3 The Impact of Cluster Size

In the proposed approach, the number of produced clusters is influenced by the parameter β which is the maximum size of any produced cluster. Obviously, as shown in Fig. 3, this parameter can influence the performance of the proposed approach. As a large value of β can result in a small number of large-sized clusters, a large number of objects will be checked to find k-NNs. This is illustrated in Section 3.1. On the other hand, a small value can result in a large number of small clusters, and this can minimize the search boundaries, especially when k is small. However, the distances between each query object and all produced clusters are required for each object, and this adds extra cost in addition to distance computations within the minimized search boundaries.

Clearly, a near-optimal performance relies on the optimal value of β. In this section, we demonstrate how a near-optimal value of β can be inferred from the size of a data set. Since the proposed approach inherits the nature of the FWC algorithm, the exact number of produced clusters cannot be determined; therefore, we tested various values of β to produce a number of clusters that range approximately from $0.5\sqrt{s}$ to $8\sqrt{s}$, where s is the size of the data set. In this investigation, data sets varying in terms of domain, size and dimensionality are used.

As can also be seen in Fig. 3, the performance of the proposed approach is not very sensitive to the number of produced clusters within the range of $\sqrt{s}$ and $3\sqrt{s}$. Moreover, it can be seen that the variance in the performance for each k within this range is small. Therefore, we suggest using a value of the parameter β that produces clusters within this range. However, there might be some data sets that do not comply with this. Thus, a user can find out the near-optimal value of β with few attempts.

4.4 The Baseline Methods

In the literature, many fast and accurate k-NN search algorithms were proposed. The dimensionality, distribution and size of a data set play a role in the selection of an appropriate algorithm. We compare kNNVWC with the flat index based approaches, namely, FWC [27], [28] and k-means [29] and well-known tree-based spatial algorithms such as KD-tree [18], Ball tree [16], [31], [47], Cover tree [21], M-tree [22] and VP-tree [24], [25].

4.5 Choice of Parameters

The proposed and baseline methods require parameter settings. The maximum leaf size is the most important parameter that can influence the performance in tree-based spatial algorithms. Similarly, in the proposed method is the maximum cluster size parameter. While, for FWC and k-means methods are the cluster-width and number of clusters respectively. For all the experiments the leaf size of binary tree-based methods is set to 40. Overall, this value experimentally shows the best results on the most data sets. The maximum cluster size in the proposed method is set to $\sqrt{s}.$, where s is the size of the data set. This value is nearly-optimal as investigated in Section 4.3. The number of clusters in k-means is set to $2.0\sqrt{s}$ as suggested by Wang [29], and experimentally shows the best results. As for FWC method, it is a challenging task to find the nearly-optimal value.

TABLE 2
The Average Reduction Rate of Distance Calculations against Ek-NN

Data set	K	EkNN	KD-tree	Ball tree	k-means	VP-tree	FWC	kNNVWC	M-tree
arcene = 900×10000	10	7.29×10^4	-46.54%	3.56%	55.91%	50.58%	50.87%	**57.00%**	14.17%
	50		-46.54%	-0.48%	45.14%	32.45%	38.70%	**45.71%**	8.49%
	100		-46.54%	-1.88%	22.30%	29.44%	29.11%	**37.30%**	5.09%
	200		-46.54%	-2.53%	8.86%	12.91%	9.88%	**20.74%**	-0.30%
SimSCADA = 12000×850	10	1.30×10^7	86.47%	93.13%	81.37%	84.32%	83.55%	**94.12%**	49.23%
	50		70.35%	92.11%	74.08%	79.50%	79.25%	**93.39%**	48.55%
	100		48.27%	90.25%	72.91%	75.63%	75.30%	**91.98%**	46.76%
	200		36.12%	86.72%	68.70%	64.95%	65.70%	**91.47%**	44.33%
multiplefeaturs = 2000×649	10	3.60×10^5	-19.50%	-1.37%	**74.45%**	55.32%	55.66%	68.74%	22.16%
	50		-19.50%	-1.39%	**67.22%**	50.49%	46.93%	62.13%	20.14%
	100		-19.50%	-1.39%	**63.03%**	50.03%	40.12%	57.46%	18.28%
	200		-19.50%	-1.39%	**56.75%**	44.67%	31.32%	53.78%	16.30%
arrhythmia = 452×279	10	1.83×10^4	-14.00%	-1.72%	-4.20%	**9.72%**	-12.03%	-10.40%	-0.98%
	50		-14.00%	-1.72%	-4.21%	**9.06%**	-13.22%	-11.02%	-0.98%
	100		-14.00%	-1.72%	-4.18%	**8.53%**	-13.98%	-11.09%	-0.98%
	200		-14.00%	-1.72%	-4.23%	**7.62%**	-14.87%	-11.79%	-0.98%
gasSensors = 13910×128	10	1.74×10^7	83.11%	77.95%	89.81%	74.25%	89.42%	**93.32%**	52.32%
	50		61.00%	71.17%	86.66%	67.72%	86.01%	**89.85%**	48.69%
	100		47.24%	66.95%	83.48%	68.23%	83.52%	**87.36%**	46.56%
	200		32.99%	61.89%	80.24%	66.71%	80.40%	**83.18%**	43.70%
spambase = 4601×57	10	1.90×10^6	42.01%	0.70%	6.15%	77.68%	75.43%	**86.77%**	17.04%
	50		20.17%	-1.18%	3.21%	73.88%	73.28%	**82.69%**	16.35%
	100		13.02%	-2.70%	33.29%	73.21%	71.67%	**79.58%**	16.08%
	200		4.90%	-2.73%	27.69%	70.41%	69.08%	**74.42%**	15.61%
waveform = 5000×40	10	2.25×10^6	-7.36%	-1.62%	1.92%	**15.02%**	-3.38%	-1.90%	-0.36%
	50		-7.36%	-1.62%	-0.61%	**14.13%**	-5.56%	-3.58%	-0.60%
	100		-7.36%	-1.62%	-0.69%	**12.83%**	-6.27%	-4.24%	-0.66%
	200		-7.36%	-1.62%	-1.02%	**12.49%**	-6.89%	-4.55%	-0.72%
DUWWTP = 527×38	10	2.51×10^5	0.14%	-9.49%	-1.01%	**9.39%**	-1.87%	0.51%	0.64%
	50		-9.70%	-1.01%	-1.35%	**8.15%**	-4.71%	-1.66%	0.64%
	100		-9.70%	-1.01%	-1.74%	**8.10%**	-5.66%	-1.88%	0.64%
	200		-9.70%	-1.01%	-1.99%	**8.10%**	-6.46%	-2.08%	0.64%
shuttle = 43500×9	10	1.70×10^8	**97.51%**	96.20%	51.79%	83.12%	92.69%	93.29%	42.55%
	50		**96.57%**	95.55%	48.91%	75.85%	90.18%	89.88%	39.31%
	100		**96.79%**	95.07%	47.24%	74.07%	88.28%	87.36%	37.52%
	200		**94.79%**	93.37%	44.57%	70.69%	85.82%	83.91%	35.35%
MORD = 4690×2	10	1.98×10^6	**97.49%**	93.79%	87.86%	58.69%	90.02%	93.17%	77.17%
	50		**96.01%**	92.81%	86.49%	48.85%	88.90%	91.97%	76.22%
	100		**94.33%**	90.10%	86.94%	52.48%	87.63%	90.47%	75.36%
	200		**91.38%**	87.57%	84.12%	52.35%	84.76%	88.19%	73.08%
Slice = 53500×386	10	2.58×10^8	14.77%	46.70%	84.17%	74.44%	71.62%	**90.49%**	82.88%
	50		-7.54%	19.49%	81.44%	71.89%	69.23%	**89.48%**	77.49%
	100		-10.43%	14.22%	74.94%	70.73%	55.69%	**85.99%**	75.73%
	200		-11.70%	10.59%	70.69%	69.30%	50.81%	**80.48%**	74.21%
kddcup99 = 494021×41	10	2.20×10^{10}	78.36%	77.98%	5.10%	69.84%	73.26%	**81.79%**	76.46%
	50		78.30%	77.87%	5.06%	67.12%	73.18%	**81.74%**	75.89%
	100		78.24%	77.81%	5.02%	65.94%	73.14%	**81.68%**	75.56%
	200		78.15%	76.34%	4.66%	65.32%	73.08%	**81.59%**	73.93%

Therefore, we choose the value that can produce a number of clusters as close as possible to $2.0\sqrt{s}$.

4.6 Distance Metric

There are two steps in finding the exact k-NNs of a query object in all baseline methods and the proposed kNNVWC. In the first step, the search area in the feature space, which definitely has the k-NNs, is defined. In tree-based spatial algorithms, distance calculations between the query object and the closet nodes in traversing down and up the tree are required to find the leaf nodes that define this area. However, it is defined in flat index based methods only by calculating distances between the query object and the centers of clusters. In the second step, the distances between a query object and all objects in the search area are calculated to find the exact k-NNs. Notably, we exclude the Cover tree algorithm in this comparison because the library, which is the weka jar file [48], does not provide a public method for obtaining the number of distance calculations for a query object. However, in the next Section 4.7.2, we will compare kNNVWC with the Cover tree algorithm based on the computation and construction times that are the ultimate goals of the end-users.

Various values of k 10, 50, 100 and 200 are tested in this experiment. These values are chosen to compare the performance on small, medium and large k values. As previously discussed, the distance computations for Ek-NN are fixed for all values of k. Therefore, we compare all the results with Ek-NN, and demonstrate the amount of deduction that each algorithm can achieve.

Table 2 shows the results of the comparison of seven algorithms whose total distance calculations are compared with Ek-NN which represents the worst case scenario where all objects in a data set are visited to find the k-NNs for a query object. As previously discussed, the best algorithm according to this comparison is the one that minimizes as much as possible the total number of distance calculations. From these results, we can see that kNNVWC performed the best on the highest dimensional data set *arcene* that has 10,000 features, while in overall k-means, VP-tree and FWC have competitive results. However, the large values of k could degrade their performance. This can be attributed to the small size of the data set where all optimized algorithms will visit all objects when the value of k is set equal to the data set size. On the other hand, Ball tree and KD-tree did not show any improvement with this data set. This is because the search boundaries for each query object overlapped most of nodes in the pre-created tree. Hence, all objects in these nodes must be checked. Again, kNNVWC competed against all baseline methods with high dimensional data sets such as *SimSCADA*, *gasSensors* and *Slice*, except the two data sets; *multiplefeaturs* and *arrhythmia*. kNNVWC ranks second with *multiplefeaturs*; however, it performs nearly as well as k-means. Clearly, all methods failed with *arrhythmia*, except the VP-tree that demonstrated insignificant results. The reason for this is the small size of this data set. Thus, the Ek-NN method is more appropriate for any small data set. As for the medium-dimensional data sets *spambase*, *waveform*, *DUWWTP*, *gasSensors* and *kddcup99*, kNNVWC demonstrated the best results only with *spambase* and *kddcup99*. However, it can be seen that all methods are worse on the data set *waveform* even though its size is relatively large, except the VP-tree that produced better results but is still not significant. This is because this data set has a dense data which might form one single condensed cluster, and therefore the range of the search area for any query object overlaps the majority of the data in n-dimensional space. In this case, a large number of objects must be checked for each query. Clearly, all methods performed well on *kddcup99*. However, due to this, the data set has several large clusters each of which contains objects that have identical similarities in terms of the euclidean distance, kNNVWC could not split these large clusters. Thus, the search for k-NNs for a query object that is close to one of these clusters will be expensive. This is because all objects in these large clusters will be checked. In fact, kNNVWC would certainly have shown a more significant result if these clusters had been removed.

For low-dimensional data sets *shuttle* and *MORD*, all methods produced better results overall. In fact, the KD-tree demonstrated the best results and this confirms its appropriateness for low dimensions (e.g. < 10).

4.7 The Complexity Metrics

In this comparison, we evaluate kNNVWC against the seven baseline methods. Two salient aspects of this

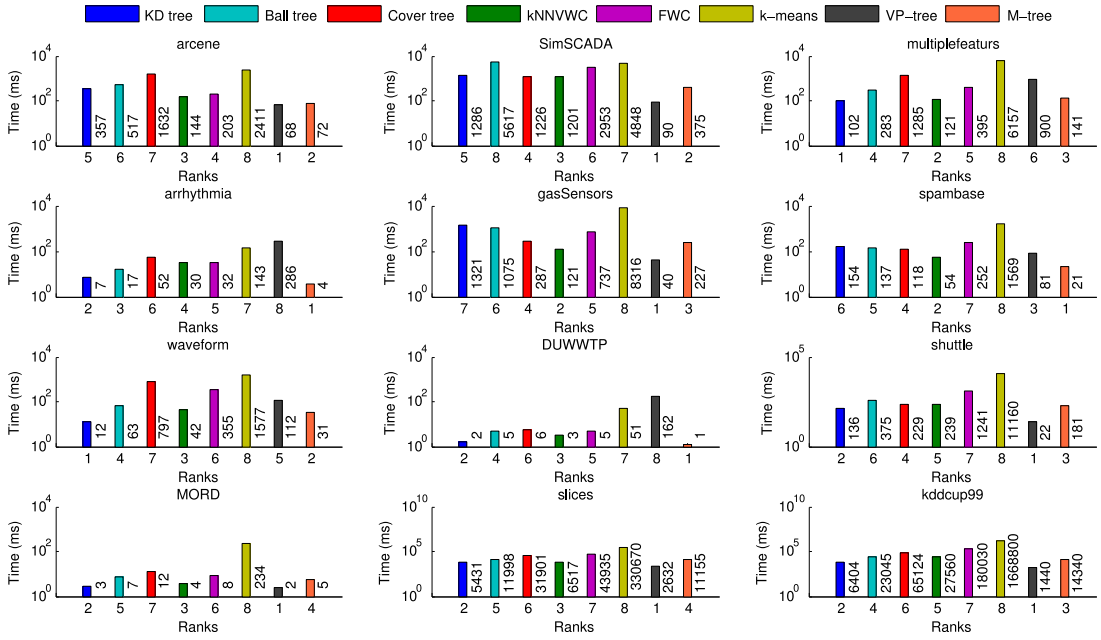


Fig. 4. The construction time of the baseline methods and kNNVWC for each data set.

evaluation are considered: *construction time* taken by each method to construct its own index and the *search time* taken to find k-NNs for a number of query objects

4.7.1 Construction Time

All methods involve a preprocessing step prior to the k-NN search, and each has its own decomposition technique to construct its own index. This part provides an experimental evaluation of the efficiency of each method in constructing its own index.

Fig. 4 demonstrates the construction time for each of the methods on various data sets. It can be seen that the VP-tree is the most efficient, while the KD-tree, M-tree and kNNVWC ranked second and they can bee seen competitive with each other. Cover tree and Ball tree in comparison with other methods are influenced by the number of dimensions and the distribution of a data set. For instance, Cover tree exhibits the worst construction time on the high dimensional data *arcene*, while Ball tree is relatively acceptable compared to others. The opposite is true for the high-dimensional data *SimSCADA*. This fluctuation can be seen in other data sets. Therefore, this might be a result not only of dimensionality, but of distribution. Clearly, k-means is the worst method and is influenced by the size of the data set. Similarly, FWC is considered as the second worst method.

4.7.2 Search Time

This section presents an evaluation of the efficiency of each method against EkNN in searching for k-NNs. Note that the efficiency value for each method is calculated as in Section 4.2.2 and this value is categorized into three statuses (Better, Similar and Worse) as follows:

$$\begin{cases} SR > 1 & Better \\ SR = 1 & Similar \\ SR < 1 & Worse. \end{cases} \quad (5)$$

That is, Ω method is better when it takes less time than EkNN to find k-NNs, and the similar status when its search time is exactly the same as the time consumed by EkNN. Otherwise, the status is worse, and this occurs when the search time is longer than the time taken by EkNN. Therefore, any method producing a result greater than one can be an alternative to EkNN. However, the method with the highest efficiency value is the best choice. On the other hand, EkNN is the best when $SR \leq 1$ because it does not require a pre-processing step and is simple to implement.

In this evaluation, the same values of k are used here as were used in Section 4.6. Fig. 5 shows the efficiency of each method when searching for the k-NNs in various data sets. Overall, the tree-based indexes such as KD-tree, Ball tree, Cover tree and M-tree demonstrate poor results for the high dimensional data sets except M-tree on *Slice* especially when the value of k is large. However, the VP-tree can be the best among them. On the other hand, flat-based indexes such as FWC, k-means kNNVWC demonstrate promising results except with the small size data set *arrhythmia* and their efficiency is stable even with a large value of k. Nevertheless, for high dimensional data, kNNVWC is the most efficient method compared with all other methods, except with *multiplefeaturs* that demonstrated slightly poorer results.

With the medium-dimensional data sets, in contrast to VP-tree, all tree-based indexes show good results only for *kddcup99*. The relatively large size of this data set is the possible reason for this improvement. This is because EkNN, with which all results were compared, is very expensive with this data set. On the other hand, the VP-tree and all flat-based indexes except k-means demonstrate interesting results for *kddcup99* and *spambase*. However, they failed with *waveform* and *DUWWTP*. As previously discussed, the former is very dense and relatively small, while the latter is very small. However, it is quite obvious that kNNVWC demonstrated the best results. As for the low-dimensional

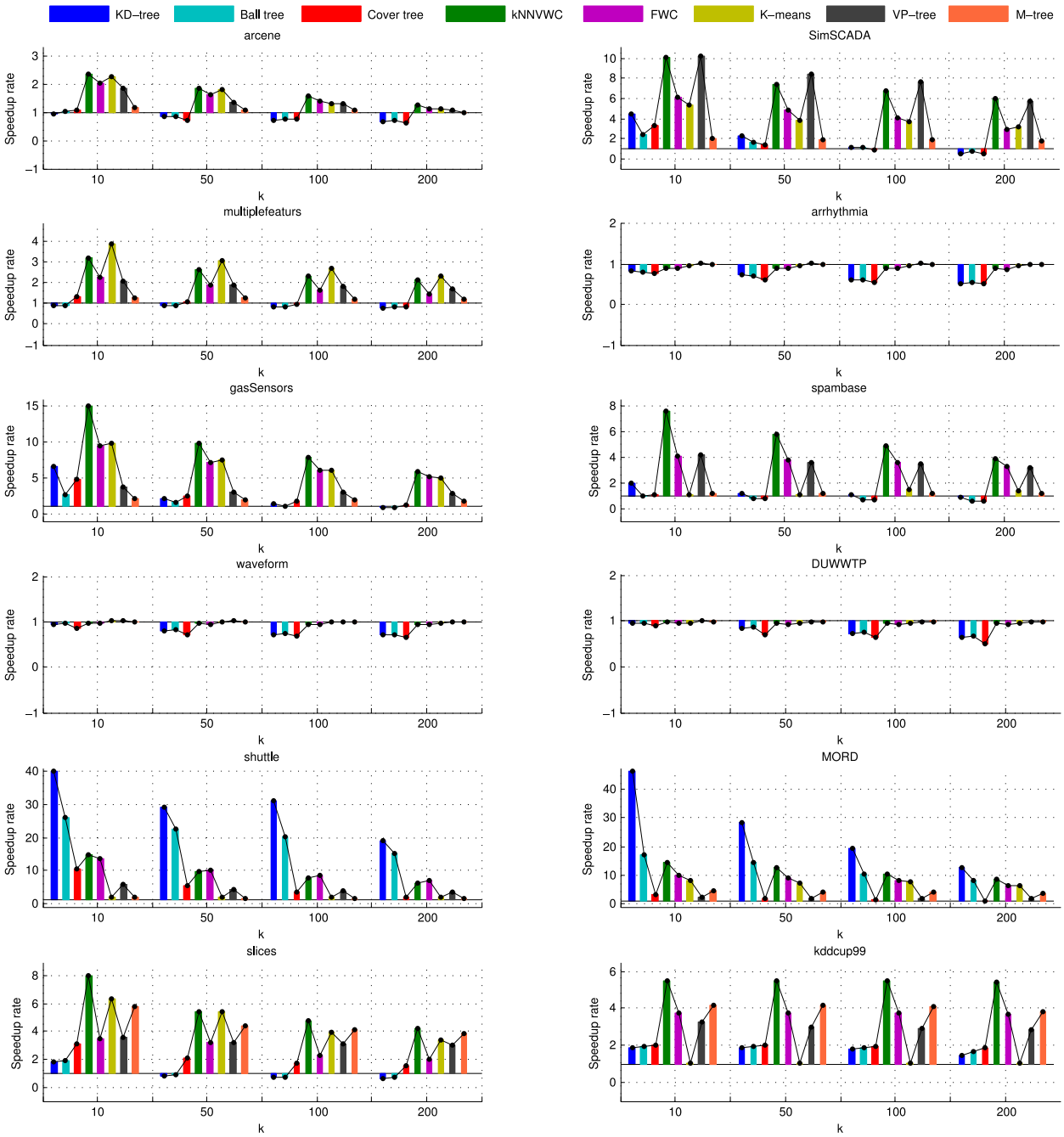


Fig. 5. The efficiency of the baseline methods and kNNVWC against EkNN in searching for k-NNs.

data sets, KD-tree and Ball tree demonstrated significant results with slightly less efficiency when dimensionality increases. kNNVWC ranks the second with interesting and stable results.

5 Conclusion

This work presented a novel nearest-neighbor approach based on various-widths clustering called kNNVWC. This approach is able to produce compact and well-separated clusters from high dimensional data of various distributions. The quality of decomposition results has maximized the efficiency of using triangle inequality to prune unlikely clusters in searching for k-NNs. The proposed algorithm shares efficiency feature in producing less overlapping clusters with non-binary partitioning techniques, and meanwhile its clustering time is as fast as that of binary partitioning methods. In the experiments, twelve data sets, which vary in domains, size and dimensionality, have been used. The experimental results have shown that kNNVWC has demonstrated promising and stable performance comparing to seven algorithms.

In future, we aim to further improve the performance of kNNVWC by reducing both the pre-processing cost and the querying cost. Specifically, we will investigate parallel algorithms to reduce the pre-processing cost. We aim to further improve the querying cost using the

following ideas: i) learning optimal clustering radius for different distribution with an aim to minimise the overlap among clusters; and ii) grouping the centers (and radii) of the clusters into a tree-like index to effectively prune more clusters.

ACKNOWLEDGMENTS

The research of Muhammad Aamir Cheema is supported by ARC DE130101002 and DP130103405.

REFERENCES

[1] G. Shakhnarovich, T. Darrell, and P. Indyk, "Nearest-neighbor methods in learning and vision," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, p. 377, Feb. 2008.

[2] F. Angiulli and F. Fassetti, "DOLPHIN: An efficient algorithm for mining distance-based outliers in very large datasets," *ACM Trans. Knowl. Discovery Data*, vol. 3, no. 1, p. 4, 2009.

[3] F. Angiulli, S. Basta, and C. Pizzuti, "Distance-based detection and prediction of outliers," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 2, pp. 145–160, Feb. 2006.

[4] A. Ghoting, S. Parthasarathy, and M. E. Otey, "Fast mining of distance-based outliers in high-dimensional datasets," *Data Mining Knowl. Discovery*, vol. 16, no. 3, pp. 349–364, 2008.

[5] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 29–38.

[6] P. Cunningham and S. J. Delany, "k-nearest neighbour classifiers," *Multiple Classifier Systems*, pp. 1–17, 2007.

[7] A. Almalawi, X. Yu, Z. Tari, A. Fahad, and I. Khalil, "An unsupervised anomaly-based detection approach for integrity attacks on scada systems," *Comput. Security*, vol. 46, pp. 94–110, 2014.

[8] A. Shintemirov, W. Tang, and Q. H. Wu, "Power transformer fault classification based on dissolved gas analysis by implementing bootstrap and genetic programming," *IEEE Trans. Syst., Man Cybern. C, Appl. Rev.*, vol. 39, no. 1, pp. 69–79, 2009.

[9] S. Magnussen, R. E. McRoberts, and E. O. Tomppo, "Model-based mean square error estimators for k-nearest neighbour predictions and applications using remotely sensed data for forest inventories," *Remote Sensing Environ.*, vol. 113, no. 3, pp. 476–488, 2009.

[10] M. Govindarajan and R. Chandrasekaran, "Evaluation of k-nearest neighbor classifier performance for direct marketing," *Expert Syst. Appl.*, vol. 37, no. 1, pp. 253–258, 2010.

[11] 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.*, 2006, pp. 459–468.

[12] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: self-tuning indexes for similarity search," in *Proc. 14th Int. Conf. World Wide Web.*, 2005, pp. 651–660.

[13] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search," in *Proc. 33rd Int. Conf. Very Large Data Bases*, 2007, pp. 950–961.

[14] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2008, pp. 1–8.

[15] M. Muja and D. Lowe, "Scalable nearest neighbour algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 1, 2014.

[16] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE Trans. Comput.*, vol. C-100, no. 7, pp. 750–753, Jul. 1975.

[17] 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.

[18] R. F. Sproull, "Refinements to nearest-neighbor searching ink-dimensional trees," *Algorithmica*, vol. 6, no. 1–6, pp. 579–589, 1991.

[19] J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, no. 10, pp. 1000–1006, Oct. 1975.

[20] B. S. Kim and S. B. Park, "A fast k nearest neighbor finding algorithm based on the ordered partition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. TPAMI-8, no. 6, pp. 761–766, Jun. 1986.

[21] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learning*, 2006, pp. 97–104.

[22] P. Patella, M. Ciaccia, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. 23rd Int. Conf. Very Large Data Bases*, 1997, pp. 426–435.

[23] T. Bozkaya and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," *ACM SIGMOD Rec.*, vol. 26, no. 2, pp. 357–368, 1997.

[24] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM-SIAM Sym. Discr. Algorithms*, 1993, pp. 311–321.

[25] J. K. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. Process. Lett.*, vol. 40, no. 4, pp. 175–179, 1991.

[26] C. T. Jr., A. Traina, B. Seeger, and C. Faloutsos, *Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes*. New York, NY, USA: Springer, 2000.

[27] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in *Applications of Data Mining in Computer Security*. New York, NY, USA: Springer, 2002, pp. 77–101.

[28] M. J. Prerau and E. Eskin, "Unsupervised anomaly detection using an optimized k-nearest neighbors algorithm," Undergraduate thesis, Columbia Univ., New York, NY, USA, Dec. 2000.

[29] W. Xueyi, "A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality," in *Proc. Int. Joint Conf. Neural Netw.*, 2011, pp. 1293–1299.

[30] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[31] S. M. Omohundro, *Five Balltree Construction Algorithms*. Int. Comput. Sci. Inst. Berkeley, CA, USA, 1989.

[32] N. Kumar, L. Zhang, and S. Nayar, "What is a good nearest neighbors algorithm for finding similar patches in images?" in *Proc. 10th Eur. Conf. Comput. Vis.*, 2008, pp. 364–378.

[33] S. Brin. (1995). Near neighbor search in large metric spaces. *Proc. 21th Int. Conf. Very Large Data Bases* [Online]. Available: http://ilpubs.stanford.edu:8090/113/

[34] A. W. chee Fu, P. M. shuen Chan, Y.-L. Cheung, and Y. S. Moon, "Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances," *VLDB J.: Int. J. Very Large Data Bases*, vol. 9, no. 2, pp. 154–173, 2000.

[35] T. Bozkaya and M. Ozsoyoglu, "Distance-based indexing for high-dimensional metric spaces," *ACM SIGMOD Rec.*, vol. 26, no. 2, pp. 357–368, 1997.

[36] P. Portnoy, E. Eskin, and S. Stolfo, "Intrusion detection with unlabeled data using clustering," in *Proc. ACM CSS Workshop Data Mining Appl. Security*, 2001, pp. 5–8.

[37] J. Oldmeadow, S. Ravinutala, and C. Leckie, "Adaptive clustering for network intrusion detection," in *Proc. 8th Pacific-Asia Conf. Adv. Knowl. Discovery Data Mining*, 2004, pp. 255–259.

[38] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection," in *Proc. 6th Int. Symp. Recent Adv. Intrusion Detection*, 2003, pp. 220–237.

[39] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, San Jose, CA, USA, May. 22–25, 1995, pp. 71–79.

[40] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Trans. Database Syst.*, vol. 24, no. 2, pp. 265–318, 1999.

[41] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. Int. Joint Conf. Artif. Intell.*, 1995, vol. 14. pp. 1137–1145.

[42] A. Frank and A. Asuncion. (2013). UCI machine learning repository [Online]. Available: http://archive.ics.uci.edu/ml

[43] A. Almalawi, Z. Tari, I. Khalil, and A. Fahad, "SCADAVT-a framework for SCADA security testbed based on virtualization technology," in *Proc. IEEE 38th Conf. Local Comput. Netw.*, 2013, pp. 639–646.

[44] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sens. Actuators B: Chemical*, vol. 166, pp. 320–329, 2012.

[45] I. Rodriguez-Lujan, J. Fonollosa, A. Vergara, M. Homer, and R. Huerta, "On the calibration of sensor arrays for pattern recognition using the minimal number of experiments," *Chemometrics Intell. Laboratory Syst.*, vol. 130, pp. 123–134, 2014.

[46] S. Suthaharan, M. Alzahrani, S. Rajasegarar, C. Leckie, and M. Palaniswami, "Labelled data collection for anomaly detection in wireless sensor networks," in *Proc. 6th Int. Conf. Intell. Sensors, Sensor Netw. Inf. Process.*, Dec. 2010, pp. 269 –274.

[47] T. Liu, A. W. Moore, and A. Gray, "New algorithms for efficient high-dimensional nonparametric classification," *J. Mach. Learning Res.*, vol. 7, pp. 1135–1158, 2006.

[48] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.