Speedup Clustering with Hierarchical Ranking

Jianjun Zhou Joerg Sander
Department of Computing Science
University of Alberta
{jianjun,joerg}@cs.ualberta.ca

Abstract

Many clustering algorithms in particular hierarchical clustering algorithms do not scale-up well for large data-sets especially when using an expensive distance function. In this paper, we propose a novel approach to perform approximate clustering with high accuracy. We introduce the concept of a pairwise hierarchical ranking to efficiently determine close neighbors for every data object. We also propose two techniques to significantly reduce the overhead of ranking: 1) a frontier search rather than a sequential scan in the naïve ranking to reduce the search space; 2) based on this exact search, an approximate frontier search for pairwise ranking that further reduces the runtime. Empirical results on synthetic and real-life data show a speedup of up to two orders of magnitude over OPTICS while maintaining a high accuracy and up to one order of magnitude over the previously proposed DATA BUBBLES method, which also tries to speedup OPTICS by trading accuracy for speed.

1. Introduction and Related work

Clustering is an important problem in data-mining. However, many clustering algorithms especially hierarchical clustering algorithms do not scale-up well for large data-sets, which are becoming more and more common nowadays. For instance, the hierarchical clustering algorithm OPTICS [1] without index support has quadratic runtime complexity of $O(n^2)$, where n is the size of the data-set.

This quadratic time complexity could be a big problem for even a medium sized data-set in some applications, since such a time complexity in a clustering algorithm typically leads to the same amount of distance computations, which could be computationally very expensive, depending on the application. For instance, computing the dissimilarity between a pair of 3-dimensional protein structures using structure alignments can take up to tens of seconds on a state-of-the-art PC. In a data-set of size ten thousand, for example, if the average runtime of computing a distance is one second, computing all pairwise distances would require more than half a year.

Theoretically, it is possible to reduce the average runtime complexity of OPTICS to $O(n\log n)$, if an index structure is available that can perform similarity range queries in $O(\log n)$ time. OPTICS computes all pairwise distances only if it takes $O(n)$ time to evaluate a similarity range query. (A similarity range query retrieves all objects in a large database that are similar to a query object, typically using a distance function to measure the dissimilarity.)

There has been an intensive effort [7] over the last two decades to speedup similarity search in metric spaces. The typical approach is to build some form of tree-like indexing structures in advance to speedup the similarity range query in the application. While researchers have achieved significant progress in some types of data such as low dimensional Euclidean data, the problem is difficult for high dimensional space and general metric space [5], and finding consistently efficient index structures for high-dimensional and general metric spaces has been so far an elusive goal, due to a number of effects in these spaces, collectively referred to as the “curse of dimensionality”. Therefore, there is currently no indexing support that can efficiently and effectively speedup exact hierarchical clustering in these kinds of data spaces.

A different way of speeding up a clustering algorithm is to trade accuracy for speed. A naïve approach is to sample a small portion of the whole data-set and run the clustering algorithm on the sample only. The drawback of this approach is that the result could be distorted, depending on the sampling rate. The smaller the sample is, the faster the runtime, but the worse the accuracy.

To improve this naïve sampling scheme, different methods have been proposed. The general idea is to collect certain sufficient statistics such as the linear sum of the set of points in the region around each sample point. For instance, BIRCH [15] is the first method to incorporate sufficient statistics with a hierarchical partitioning structure to speedup clustering. It partitions the space along a hierarchical tree in which so-called clustering features for each node in the tree are collected. A clustering feature consists of the number of points, the linear sum, and the square sum of the set of points represented by a node. Clustering algorithms are then applied on the leaf nodes of the hierarchical tree rather than the original data or sample points. DATA BUBBLES [3] use different sets of statistics particularly targeted to speedup *hierarchical* clustering algorithms such as OPTICS. Results in [16] show that this approach is able to handle non-vector data as well as vector data.

While being a significant improvement over the naïve sampling, the approach of using sufficient statistics to derive approximate clustering results is still suffering from the problem of inadequate accuracy for some important real-life applications. For these methods, clusters with a size smaller than the number of points in the smallest abstract region, represented by a set of sufficient statistics, will typically be lost in the final clustering result. Even clusters that have a large number of points but are close to other clusters could be buried in bigger clusters in the output result, since gaps between clusters can often not be recovered correctly by BIRCH or DATA BUBBLES [16].

In this paper, we propose a novel approach to perform approximate clustering with high accuracy. The method is based on the observation that in some clustering algorithms such as OPTICS and single-link, the final clustering result depends largely on the nearest neighbor distances of data objects, which comprise only a very small portion of the quadratic amount of pairwise distances between data objects. We introduce a novel pairwise hierarchical ranking to efficiently determine close neighbors for every data object. The clustering will then be performed on the original data objects in stead of on sample points or sufficient statistics as in the previous methods. Since a naïve pairwise hierarchical ranking may introduce a large computational overhead, we also propose two techniques to significantly reduce this overhead: 1) a frontier search rather than a sequential scan in the naïve ranking to reduce the search space; 2) an approximate frontier search for pairwise ranking that further reduces the runtime. Empirical results on synthetic and real-life data show a speedup of up to two orders of magnitude over previous approaches.

The remainder of this paper is organized as follows. In Section 2 we introduce background knowledge including the OPTICS clustering algorithm; in Section 3, we state the motivation of the new method; Section 4 discusses the idea of ranking; in Section 5, we introduce our new ranking method; in Section 6, we compare our method empirically with the previous methods; finally, we conclude with Section 7.

2. Preliminaries

2.1 Three Major Clustering Approaches

Clustering algorithms can be categorized based on how they cluster the data objects. In this sub-section we briefly introduce three of the major categories: partitioning, hierarchical and density-based approaches. For a complete description of all categories, see [8].

The partitioning approach is represented by the *k*-means algorithm. This approach selects a set of centers and partitions the data-set by assigning data objects to their nearest center. The centers are then adjusted according to the objects in each group and the assignment process is repeated to refine the result. Each group of objects assigned to a center is considered a cluster.

The hierarchical approach is represented by the single-link algorithm. Starting from groups of individual data objects (one data object per group), the method agglomerates two nearest groups into a new group. The final result is a hierarchical ordering of all data objects that shows the process of the agglomeration.

The density-based approach is represented by the DBSCAN algorithm [5]. The method estimates the density of the region around each data object by counting the number of neighbor objects within a given radius. It then connects dense regions to grow them into clusters.

Although our algorithm can be applied to other clustering methods, due to the space limitations, we focus on OPTICS, which is a hierarchical clustering algorithm that uses density-based concepts to measure the dissimilarity between points.

2.2 Triangle Inequalities in Metric Space

Given a set of objects B and a distance function D between objects in B, by definition, a metric space satisfies the following properties:

1. (Positiveness) for all x, y in B, $D(x, y) \geq 0$,
2. (Symmetry) for all x, y in B, $D(x, y) = D(y, x)$,
3. (Reflexivity) for all x in B, $D(x, x) = 0$,

4. (Triangle inequality) for all x, y, z in B, $D(x,y) + D(y,z) \geq D(x,z)$.

The triangle inequality can be used in a technique called *pruning* to avoid distance computations in data retrieval operations and data-mining applications that require distance computations.

To apply the pruning technique, typically the distances between a selected small set of objects P and all other objects o in a data-set are pre-computed in a preprocessing step. The objects $p \in P$ are called a "pivots" or "reference points" in the literature.

In a range query, for example, a query object q is given and the task is to find objects within a given query radius r from q. For any data object o and pivot p, by a derived form of the triangle inequality, it holds that $D(q,o) \geq |D(q,p) - D(o,p)|$. Therefore, at query time, the distance $D(q,p)$ is also computed in order to determine if $|D(q,p) - D(o,p)| > r$. If this condition is true, then it follows that $D(q,o) > r$, and o can safely be excluded without actually computing the distance $D(q,o)$.

The triangle inequality has been incorporated in several indexing methods for metric data, for instance the M-TREE [10]. It can lead to a substantial saving of distance computations in low dimensional spaces and in metric spaces that can be mapped to a low dimensional space. In high dimensional space and general metric space, however, its effectiveness deteriorates.

Compared with the sampling and methods such as BIRCH and DATA-BUBBLES, the advantage of using triangle inequalities is that it can provide additional speedup for virtually any method on metric data (including our method) and it is an exact method that loses no accuracy.

2.3 Clustering with OPTICS

The OPTICS algorithm is a combination of the hiearchical and density-based clustering approaches. OPTICS estimates the density of the region around each data object as in the traditional density-based clustering algorithm, and then hiearchically orders all data objects in a bar plot, so that objects that are close in dense regions will be close in the ordering. An example reachability plot for a 2-dimensional data set is depicted in Figure 1. Such a plot is interpreted as following: "valleys" in the plot represent clusters, and the deeper the "valley", the denser the cluster. The tallest bar between two "valleys" is a lower bound on the distance between the two clusters. Large bars in the plot that are not at the border of a cluster represent noise, and "nested valleys" represent hierarchically nested clusters.

There are two important notions in the OPTICS algorithm: core-distance and reachability-distance for objects with respect to parameters *Eps* and *MinPts*. The parameter *MinPts* allows the core-distance and reachability-distance of a data object to capture the density around that object. Given a radius *Eps*, an object with at least *MinPts* neighbors within *Eps* is called a "core-object". For any core-object c, its core-distance is the *MinPts*-nearest neighbor distance, and its reachability-distance to another object o is the greater value of the core-distance of c and the distance between c and o.

The ordering of OPTICS is essentially a "walk" through all data objects. Starting from an arbitrary object marked as visited with an undefined reachability-distance that is set to "infinity", it always selects an object o with the smallest reachability distance d to any of the already visited objects as the next object to visit. The value d is then assigned to o. An *Eps*-range query is performed for o to determine o's core distance as well as all objects and their reachability distance (w.r.t. o) within o's *Eps* neighborhood. The output of the algorithm is a bar plot of the reachability values assigned to the objects in the order they were visited.

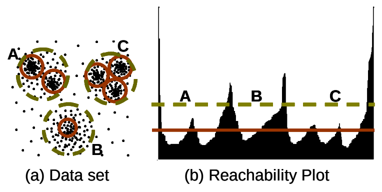


Figure 1: A data-set and its OPTICS output.

3. Motivation

Although the OPTICS algorithm without index support requires the computation of $O(n^2)$ distances, its final result depends largely on the *MinPts*-nearest neighbor distances only. Some large distances (larger than typical *MinPts*-nearest neighbor distances) between clusters also count, but OPTICS only needs a few of them, e.g., one per pair of clusters as depicted in Figure 2, while most of the reachability-distances plotted in the output are short distances within clusters. The exact values of these large distances can even been replaced by approximated values without significantly changing the cluster structure in the output plot, since as long as the approximation value is large enough, it can fulfil its function of separating a cluster from the remaining of the data-set.

It is also not necessary to figure out the exact *MinPts*-nearest neighbor for each object to compute its core-distance (approximately). Since OPTICS only uses the values of the *MinPts*-nearest neighbor *distances*, an object with a very similar distance as the *MinPts*-nearest neighbor can also serve the same purpose. Overall, in order to preserve the quality of the final output, we are only required to provide OPTICS with values that are similar to the *MinPts*-nearest neighbor distance for each object and a few large distances between clusters. To achieve this without computing $O(n^2)$ distances, we will introduce the method of pairwise hierarchical ranking in Section 5.

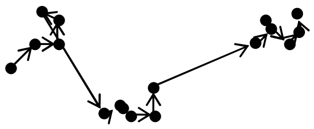


Figure 2: An OPTICS walk. The arrows represent the ordering in the walk. Although the corresponding reachability-distances are different from the distances between the pairs of objects, the lengths of the edges indicate the level of the reachability-distance values. It shows that most plotted reachability-distances are in small values.

4. *k*-Close Neighbor Ranking

The problem of ranking a list of objects has been well studied in social sciences, with the typical application of ranking political candidates. Different from this general problem of ranking, in this paper, we will focus on a special kind of ranking in computer science.

Definition 3.1 [*k*-cn ranking] Given a list of data objects and a query, the problem is how to rank the list of data objects so that the top *k* objects in the list contain many close neighbors. This problem is called the *k-close neighbor (k-cn) ranking* problem.

For our application, we do not require to find the true top *k* nearest neighbors, as long as (1) the ranking returns close neighbors that have similar distances to the query as the true top *k* nearest neighbors, and (2) the ranking is "consistent" among all query objects in the sense that the number of close neighbors returned by the ranking reflects consistently the density around all query objects, then the ranking is good enough to be used in our clustering method to estimate density. This low requirement on the accuracy of ranking is also due to the fact that our method will compute the actual distances between the query and the top *k* objects to filter out far away objects, as will be discussed in Section 5.

4.1 Ranking using Triangle Inequalities

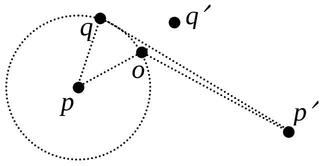


Figure 3: Ranking with triangle inequalities. Although *p* can not be used to estimate D(*q*,*o*), *p can* be used to estimate D(*q'*,*o*). While D(*q*,*o*) can not be estimated by using *p*, chances are that D(*q*,*o*) can be estimated by using another pivot *p'*.

It has been long observed empirically [14] that the triangle inequality in a metric space ($D(x,y) + D(y,z) \geq D(x,z)$ for data objects x, y, z) can be used to detect close neighbors for a given query object. While the triangle inequality can be used to speedup data-mining applications via the pruning technique [4] as discussed in Section 2.2, the use of triangle inequalities to perform ranking is only gaining the attention of researchers in recent years [2]. Given a distance function D(.,.), a query object q and data objects o and p, $E_p(q,o) = |D(q,p) - D(o,p)|$ is a lower bound estimation of $D(q,o)$ using p as a pivot. As shown in the 2-d example of Figure 3, when q and o are not on the circle centered at p, then the absolute difference value $|D(q,p) - D(o,p)|$ will be larger than zero and can indicate the actual $D(q,o)$ values. The estimation will be the better, the closer q, o, and p are located on a straight line. Using several pivots will typically improve the estimation, since if one pivot fails to estimate $D(q,o)$ well, chances are that it can be estimated better using another pivot. The rankings of different individual pivots in a set of pivots P can be easily combined in order to obtain the best estimation as the largest lower bound:

$$E_P(q,o) = \max_{p_i \in P} |D(q,p_i) - D(o,p_i)|.$$

The merit of this ranking method lies in its ability to save distance computations. In the preprocessing stage, the distances between all data objects and all pivots in P are computed; then, in the application, when the estimation of $D(q,o)$ is needed, e.g., to retrieve the nearest neighbor of a query q, the above formula can be applied for all data objects o and the one with the smallest $E_P(q,o)$ value is the estimated nearest neighbor. All required distances except those between q and pivots in P have been computed in the preprocessing stage, so that in the application, only computationally cheap operations and $|P|$ distance computations are performed. When the number of pivots is set to be small and the distance function is computationally expensive, the total amount of computations is much smaller than in the brute-force approach of computing all distances between q and all data objects to find the nearest neighbor. In most scenarios, the runtime of an application is much more important than that of a possible preprocessing, since the preprocessing is usually performed in advance and only once for several applications. But even when the runtime of the preprocessing stage is counted in the total runtime of an application, the ranking method can still significantly speedup our intended applications where the runtime is dominated by the runtime of typically very expensive distance computations such as hierarchical clustering where the closest neighbors have to be determined for each object in the data-set. In these applications, the total amount of computed distances is $O(n|P|)$, which is much smaller than $O(n^2)$.

4.2 An Analysis of When Ranking Works

In this sub-section, we give a theoretical analysis to show why $E_P(q,o)$ can be used to estimate $D(q,o)$ in general metric spaces.

For any pivot p, a query q and a close neighbor c of the query, $E_p(q,c) = |D(q,p) - D(o,p)|$ is bounded by $D(q,c)$ since, by triangle inequality, $|D(q,p) - D(o,p)| \leq D(q,c)$. This result can be extended directly to the case of using a set of pivots P, with $E_P(q,c) \leq D(q,c)$. Therefore, if a neighbor of the query is very close to it, then $D(q,c)$ is small, and consequently $E_P(q,c)$ must be small. This means that when ranking objects according to their estimated distance to q, c can be expected to be ranked high, if not many objects that are farther away from q have estimated distances lower than $E_P(q,c)$. The important question is therefore: How large will $E_P(q,o)$ on average be for a *randomly chosen* object o?" If $E_P(q,o)$ has a high probability of being larger than $E_P(q,c)$, then close neighbors will mostly be ranked higher than random objects. Theorem 3.1 below gives the probability of random objects o getting a $E_P(q,o)$ value lager than a given value.

Theorem 4.1 Given a data-set B with metric distance function D(.,.), let query q, data object o and pivot set P be selected randomly from B.

Let $Z = \{D(q,p_i) - D(o,p_i),\ p_i \in P\}$, and let $P_Z(x)$ be the probability that for an arbitrary z in Z, $|z - \mu| \leq x$, where μ is the mean of values in Z. Then

$\Pr[E_P(q,o) > x] = 1 - (P_Z(x))^{|P|}$.

Proof: Let $S = \{v|\ v = D(q,p_i)$ or $v = D(o,p_i)$, p_i in $P\}$. Since q, o and the pivots in P are selected randomly from B, elements in S are independent of each other. Thus the $z_i = D(q,p_i) - D(o,p_i)$ are also independent of each other. Therefore

$$\Pr[E_P(q,o) \leq x] = \Pr[\max_{p_i \in P} |D(q,p_i) - D(o,p_i)| \leq x]$$

$$= \prod_{p_i \in P} \Pr[|D(q,p_i) - D(o,p_i)| \leq x] = (P_Z(x))^{|P|}$$

Theorem 4.1 provides us with a clue of when the ranking will be effective. Let $x = D(q,c)$ be a distance between a query q and an object c. By Theorem 4.1

$\Pr[E_P(q,o) \leq D(q,c)] = (P_Z(D(q,c)))^{|P|}$

Although the distribution of Z is unknown, $P_Z(D(q,c))$ is always a monotonic function of $D(q,c)$. The smaller the $D(q,c)$, the smaller $P_Z(D(q,c))$ and consequently the smaller will be $(P_Z(D(q,c)))^{|P|}$. It also holds that the larger the number of pivots $|P|$, the smaller $(P_Z(D(q,c)))^{|P|}$. The trend of the function $f(x) = b^x$ is illustrated in Figure 4 for several values of b, varying x.

Therefore, the closer a neighbor c is to a query q, and the more pivots we use, the higher the probability that a random object is ranked lower than c.

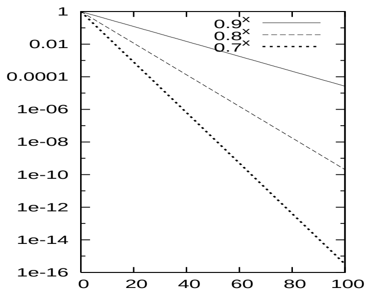


Figure 4: f(x) = b^x, for b = 0.9, 0.8, 0.7. The values decrease exponentially.

5. Pairwise Hierarchical Ranking

In this section, we propose a new method using ranking to reduce distance computations in OPTICS. The method performs a "pairwise" ranking to detect close neighbors for each object. In a *pairwise* ranking of a set of m object, every object will in turn be the query so that the ranking contains m sub-ranking of the m objects. At the end, OPTICS is run on the distances between each object and its detected close neighbors and a few additional distances between objects that are far away from each other.

As indicated by Theorem 4.1, to rank close neighbors of a query object high, we should use as many pivots as we can in the ranking, since the larger the number of pivots, the larger is the probability that a random object is ranked lower than close neighbors of the query. However, more pivots also means more distance computations between pivots and data objects, as well as the overhead of the ranking. Selecting a suitable number of pivots to balance these two factors in the traditional way of using a set of pivots is not easy.

In order to increase the number of pivots without significantly increasing the number of distance computations, we propose to perform the ranking hierarchically. The method can be roughly described as follows. First, the data is partitioned in a hierarchical way, so that for each group of objects on the bottom level, their distances to $O(\log n)$ ancestor objects are computed. Using these $O(\log n)$ ancestors as pivots, our method then performs a pairwise ranking for each group of objects to find close neighbors within the group. To find close neighbors across different groups, the method also performs ranking across several groups at a time. Since different groups of objects have different sets of ancestors, the pivots our method uses will be their common ancestors. In other words, the rankings will be performed layer by layer, along the generated hierarchical partitioning.

Our hierarchical ranking method can save distance computations because not every pivot is associated with the same amount of distance computations. The top level of pivots have distances to all objects, but in the next level, for each pivot, since its set of descendants is only a fraction of the whole data-set, the number distances associated with it is reduced to the same fraction. In this way, the pivots are constructed similar to the pyramidal hierarchy of a government: some pivots are global pivots, responsible for every member of the data-set, but some are local pivots that are responsible for members within their territories only.

The processes of partitioning and ranking will be explained in more details in the following sub-sections.

5.1 Partitioning

Our method first partitions the data in a hierarchical way, creating a tree, which we call "pivot tree". Initially, the root node of the hierarchical tree contains all objects in the data-set. During the construction, if a node v is "split", a set of f representative objects are randomly selected from the set of objects in v, and f associated child nodes (one per representative) are created under v. The set of objects in v is then distributed among its children by assigning each object in v to the child node with the closest associated representative. Each representative and the objects assigned to it will form a new node. The construction proceeds recursively with the leaf node that contains the largest number of objects until the tree has a user-specified number of leaf nodes. At the end, all data objects are contained in the leaf nodes of the tree, and all nodes except the root contain a representative. For any data object o and a node v, o is said to be *under* v if o is in v or o is in a leaf node that is a descendant of v.

5.2 Ranking

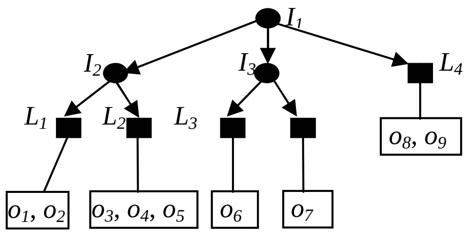


Figure 5: An example for hierarchical ranking.

Multiple rankings are performed using subsets of the representatives in the constructed hierarchical tree as pivots. We will show an example before giving the formal description of the algorithm. In Figure 5, I_i (i = 1, 2, 3) are internal nodes and L_i (i = 1, 2, 3, 4) are leaf nodes in different layers. o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, and o_9 are data objects under them. Let $I_i.rep$ and $L_i.rep$ be the representatives of internal node I_i and leaf node L_i respectively. For o_1, o_2, o_3, o_4, and o_5 since the distances between them and the representatives of L_1 and L_2 are computed when the algorithm partitions the internal node I_2 into L_1 and L_2, we can use $L_1.rep$ and $L_2.rep$ as pivots to rank data objects o_1, o_2, o_3, o_4, and o_5. Since the distances between these data objects and $I_2.rep$, $I_3.rep$, $L_4.rep$ are also computed in earlier partitions, $I_2.rep$, $I_3.rep$, $L_4.rep$ should also be used as pivots to rank them. Therefore, $\{L_1.rep, L_2.rep, I_2.rep, I_3.rep, L_4.rep\}$ is the set of pivots to perform the ranking for objects of L_1 and L_2. In the upper layer ranking of objects under I_2, I_3 and L_4, we can only use $\{I_2.rep, I_3.rep, L_4.rep\}$ as pivots to rank the whole set of $\{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9\}$, since distances between data objects o_6, o_7, o_8, o_9 and representatives $L_1.rep$, $L_2.rep$ may not be computed (they are computed only when $L_1.rep = I_2.rep$ or $L_2.rep = I_2.rep$).

The formal description of the ranking algorithm is shown in Figure 6. For any node v (if it is to pairwise rank the whole data-set, v = root and $P = \varnothing$), function rankNode performs a pairwise k-cn ranking of the objects under v, using the child representatives of v and the higher-level pivots with known distances to the objects under v as the current set of pivots. rankNode is then recursively applied on all child nodes of v. Therefore, any object o under v takes part in multiple rankings: the ranking in v as well as the rankings in all descendant nodes of v that o is also under. The lower the level of the node, the more pivots are used in its pairwise ranking and the less objects are involved in the pairwise ranking. The method will maintains a close neighbor set for each data object o. In any k-cn ranking, the top k objects with the smallest $E_P(q,o)$ values are retrieved and stored in this close neighbor set of o. The distances between objects in this set and o are computed at the end and will be used in the clustering. It is easy to prove that the number of distances computed in the partition and ranking is $O(fn\log_f n + kn\log_f n)$, where f is the branching factor in the tree, and n is the size of the data-set. However, the overhead of this ranking using naïveKcnRank can have a quadratic time complexity (although using computationally cheap operations), since the function naïveKcnRank essentially scans all objects and all pivots to compute and sort the distance estimation values $E_P(q,o)$ of each object o. In the next subsection, we will propose two new techniques to reduce this overhead.

```
rankAll(node root, int k)
    for all objects o under root
        o.closeSet ← ∅;
    rankNode(root, k, ∅);
    for all objects o under root
        for all neighbors x in o.closeSet
          o.distSet ← D(o,x);/* o.distSet stores distances associated with object o */

rankNode(node v, int k, set P)
    P′← P ∪ {all child representatives of v};
    rankObjects(v, k, P′);
    for all child nodes c of v
        rankNode(c, k, P′);

rankObjects(node v, int k, set P)
    for all objects o under v /* i.e., ∀o ∈ v.objectSet */
        topK = naïveKcnRank(o, k, P, v.objectSet);
        o.closeSet ← o.closeSet ∪ topK;

naïveKcnRank(object q, int k, set P, set objectSet)
    sortedList ← ∅;
    for all objects o in objectSet
        sortedList ← $E_P(q,o) = \max_{p_i \in P} |D(q,p_i) - D(o,p_i)|$.;
    return  top k objects in sortedList;
```

Figure 6: Hierarchical ranking algorithm.

5.3 Reducing the overhead in ranking

One issue that arises in the ranking algorithm shown in Figure 6 is that for the ranking in each node, the worst case time complexity is $O(m^2\log n)$, where m is the number of objects to rank (m decreases as the algorithm proceeds from the root to the bottom layer). This is due to the fact that the algorithm needs to perform a k-cn ranking for each data object and each ranking has a time complexity of $O(m\log n)$. (Note, however, that the time complexity is on computationally cheap operations, i.e., simple comparisons of pre-computed distance values, rather than expensive distance computations.) To reduce this overhead, we propose two new techniques: 1) a best-first frontier search [11] (rather than the sequential scan in the naïve ranking) to significantly reduce the search; 2) based on this frontier search (which results in the same pairwise ranking as the naïveKcnRank) an approximate pairwise ranking that further reduces the runtime without sacrificing too much accuracy in the application to hierarchical clustering.

5.3.1 Best-First Frontier Search

While the naïve k-cn ranking performs a sequential scan of distances between pivots and all data objects to be ranked, we propose to use instead a best-first frontier search, based on a new data structure that organizes the distances associated with pivots in the following way. Given a set of objects R under a particular node of the pivot tree and the corresponding set P of pivots for the k-cn ranking of the objects in R, for each pivot $p_i \in P$, we store the distances between p_i and $o \in R$ in a list of pairs $(o, D(o,p_i))$, and sort the list by the distance value of $D(o,p_i)$. Using $|P|$ pivots, we have $|P|$ sorted lists, and each object $o \in R$ will have exactly one occurrence in each of these lists. Between the lists of different pivots we link the occurrences of the same object together in order to efficiently access all occurrences of a particular object in all lists. The data structure is illustrated in Figure 7.

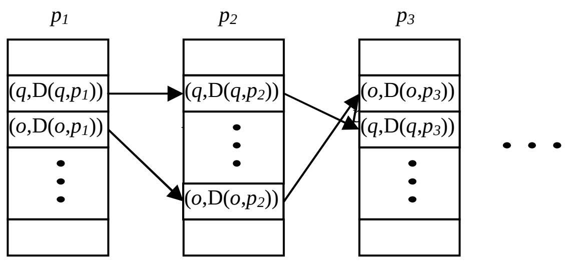


Figure 7: Linking the occurrences of each object.

When computing a pairwise k-cn ranking, each object q will be used in turn as a query object, and all other objects $o \in R$ will be ranked according to their estimated distances to q.

Instead of solving this problem with a sequential scan, our new *k*-cn ranking algorithm first retrieves all occurrences of the current query q from the given data structures. These occurrences virtually form a starting line. Then, our method searches from the starting line, advances upward and downward along the $|P|$ sorted lists, to search for the top k objects with the smallest $E_P(q,o)$ distance estimation values.

```
init(object $q$, priorityQ frontier, set $P$)
  for all occurrences $(q,D(q,p_i))$ of $q$, with $p_i \in P$
    for all immediate adjacent occurrences $(o,D(o,p_i))$ of $(q,D(q,p_i))$
      frontier $\leftarrow (o,D(o,p_i))$

/* perform $k$-$cn$ ranking for query $q$. $P$ is the pivots set */
kcnRank(object $q$, int $k$, set $P$)
  frontier $\leftarrow \varnothing$; /* the top is the pair $(o,D(o,p_i))$ with
                         $|D(o,p_i) - D(q,p_i))|$ the smallest in it */
  init($q$, frontier, $P$);
  topK $\leftarrow \varnothing$;
  while frontier $\neq \varnothing$  /* perform frontier search */
    $(o,D(o,p_i)) \leftarrow$ frontiers.pop();
    if $o$ is not in count
      count[$o$] $\leftarrow$ 0;
    else count[$o$] $\leftarrow$ count[$o$] + 1;
    if count[$o$] = $|P|$
      topK $\leftarrow o$;
      if topK = $k$
        return topK;
    frontier $\leftarrow (o',D(o',p_i))$; /* $(o',D(o',p_i))$ is the adjacent pair of $(o,D(o,p_i))$ outside the frontier */
```

Figure 8: *k-cn* ranking algorithm with best-first frontier search.

The rationale is as follows. For a query q, let object o be one of the top k objects that is returned by the naïve ranking, i.e., its distance estimation value $E_P(q,o) = \max_{p_i \in P} |D(q,p_i) - D(o,p_i)|$ is one of the k-smallest among all objects to be ranked. That also means that for object o, the values $|D(q,p_i) - D(o,p_i)|$ for each pivot p_i are all small (since $|D(q,p_i) - D(o,p_i)| \le \max_{p_i \in P} |D(q,p_i) - D(o,p_i)| = E_P(q,o)$). Consequently, the occurrences of (a top k object) o in all the lists will in general be close to the occurrences of the query q because the lists are sorted by the distances of the objects to the pivot $D(o,p_i)$, and for a difference $|D(q,p_i) - D(o,p_i)|$ to be small, $D(q,p_i)$ and $D(o,p_i)$ have to be similar values and will hence appear close to each other when sorted. Therefore, we can start from the occurrences of q and look in the nearby positions in the $|P|$ sorted lists for the top k objects by a frontier search. At the end, the number of occurrences we visit will be typically only a fraction of the total occurrences in the lists that belong to the pivots, leading to a speedup over the sequential scan.

The pseudo-code of the new *k*-cn ranking algorithm is given in Figure 8. Function kcnRank maintains a priority queue as the frontier such that its top element is a pair $(o,D(o,p_i))$ with $|D(o,p_i) - D(q,p_i))|$ the smallest in the queue. After all occurrences of q in the lists that belong to the pivots are retrieved, the frontier is initialized with occurrences immediately adjacent to those occurrences of q upward and downward. Then the function performs a frontier search in all the sorted lists, always advancing in the list that the current top element of the queue lies in. For objects already encountered when the frontier advances, the function maintains a count of the number of their occurrences. If this number is equal to the number of pivots used in the ranking, then the object is one of the top k objects returned in the final ranking. This process continues until all top k objects are found.

In the remaining of this subsection, we prove the correctness of algorithm kcnRank.

Lemma 5.1 In algorithm kcnRank, let occurrence $(a,D(a,p_i))$ be popped out of the priority queue before another occurrence $(b,D(b,p_j))$, then $|D(a,p_i) - D(q,p_i))| \le |D(b,p_j) - D(q,p_j))|$.

Proof: When $(a,D(a,p_i))$ is popped out the priority queue, $(b,D(b,p_j))$ can only be either in the frontier queue or outside the frontier (i.e. the occurrence has not yet been visited by the frontier). If $(b,D(b,p_j))$ is in the queue, then by the property of the priority queue, $|D(a,p_i) - D(q,p_i))| \le |D(b,p_j) - D(q,p_j))|$. If $(b,D(b,p_j))$ is outside the frontier, since all the lists are sorted, there must be a third occurrence $(c,D(c,p_j))$ in the list of pivot p_j with an absolute difference value $|D(c,p_i) - D(q,p_i))| \le |D(b,p_j) - D(q,p_j))|$. Since $|D(a,p_i) - D(q,p_i))| \le |D(c,p_j) - D(q,p_j))|$, $|D(a,p_i) - D(q,p_i))| \le |D(b,p_j) - D(q,p_j))|$.

Theorem 5.1 The algorithm of kcnRank and the naïve *k*-cn ranking algorithm naïveKcnRank return the same result.

Proof: Let the last top k objects found by the frontier search be t. Thus the last occurrence popped out of the priority queue belongs to t. Denote this occurrence by $(t,D(t,p_i))$. For any object o other than the returned k objects in topK, it must have an occurrence $(o,D(o,p_j))$ that can only be popped out of the priority queue after $(t,D(t,p_i))$. By lemma 5.1, $|D(t,p_i) - D(q,p_i))| \le |D(o,p_j) - D(q,p_j))|$. Thus $E_P(q,t) \le E_P(q,o)$. Also by Lemma 5.1, $(t,D(t,p_i))$ has an absolute difference value $|D(t,p_i) - D(q,p_i))|$ no less than those of the previous occurrences popped out the queue. Since for the other top k objects returned by kcnRank, all of their occurrences are popped out before $(t, D(t,p_i))$, their distance estimation values are all no greater than $E_P(q,t)$. So the elements in topK have the smallest distance estimation values among all objects to rank. Therefore, they will also be returned by the naïveKcnRank algorithm.

5.3.2 Approximate Pairwise *k*-cn Ranking

As indicated by Theorem 4.1 in Section 4.2, the larger the number of pivots, the greater the ranking accuracy. Given a fixed set of pivots, if the number of pivots is too small to effectively perform *k*-cn ranking, e.g., $k = 5$ and only 3 of the top 5 objects returned by the ranking are actually close neighbors, then some of the occurrences of the top *k* objects in the ranking may be located farther away from the corresponding occurrences of the query object in one of the sorted pivot lists. Thus the frontier search in algorithm kcnRank in Figure 8 may have to advance to a place far away from the starting point to find the occurrence of all the top *k* objects, and still incur a large overhead.

Our solution to this problem is to limit the steps that the frontier can advance from the starting position. The returned result is then no longer exactly the same as the naïve *k*-cn ranking, so that the new algorithm performs an *approximate* pairwise *k*-close neighbor ranking. When the search stops, if only k' of the top *k* objects ($k' < k$) have all occurrences within the frontier, then the remaining $k\text{-}k'$ objects are selected from those objects (besides the k' objects already selected) that have the largest numbers of occurrences within the frontier.

The rationale behind this idea is that objects with occurrences located far away from the corresponding occurrences of the query objects are more likely to be random neighbors that can not contribute short distances to be used by OPTICS, even if the frontier search goes all the way to find their occurrences. Thus setting a step limit for the frontier search will not hurt the final clustering accuracy, even if some of the top but not so close objects are not returned by the search.

Let the step limit be *s*. The approximate pairwise *k*-cn ranking algorithm has worst case time complexity of $O(sn\log n)$, where *n* is the size of the data set. Empirical results in Section 6 show that *s* can be as small as $2k$ to generate clustering results with high and robust accuracy.

5.4 Integration with OPTICS

After close neighbors of all objects have been detected by the pairwise ranking based on distance estimations, our method computes the actual distances between each object and these close neighbors. Another set of distances we will use are the distances computed in the partition stage when creating the pivot tree, i.e., the distances between the representatives of nodes and the objects under them. These are the only distances that OPTICS will use in the clustering. All other distances are assumed to be "infinitely" large.

The value of *k* should not be significantly smaller than the *minPts* parameter of OPTICS, otherwise the cluster result can be distorted because there are enough computed distances associated with each object to estimate the core-distances. In the pairwise hierarchical ranking, each object can take part in several sub-ranking, i.e., rankings of different layers, so that the number of distances associated with each object is usually a little larger than *k*. And since in practice the *minPts* parameter only needs to be relatively small to provide good results, *k* can also be set to a small value (typically <10).

6. Experimental Evaluation

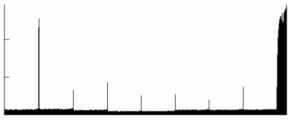
In this section, we compare our method and the DATA BUBBLES method on synthetic as well as real-life data-sets. Both methods are used to speedup the OPTICS clustering algorithm. We denote our method using approximate pairwise hierarchical ranking by OPTICS-Rank, and the DATA-BUBBLE method by OPTICS-Bubble. All experiments are performed on a Linux workstation with dual AMD Opteron 2.2GHz CPUs and 5GB of RAM, using one CPU only.

6.1 Synthetic Data

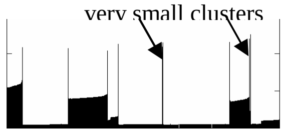
We use the two synthetic data-sets studied in [16] to show that our new method has better accuracy in detecting subtle cluster structures.

The first synthetic data we use, denoted by *DS-Vector*, is a synthetic 2-dimensional point data set. It contains 50000 points distributed over 8 clusters and 4% background noise. The clustering output of OPTICS is depicted in Figure 9 (a). Some of the 8 clusters are very close to each other as indicated by the relatively low reachability values that separate them.

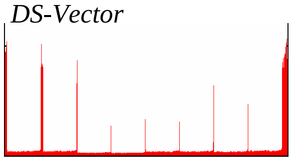
The second data set, called *DS-Tuple*, is a synthetic set of binary strings. Each object of *DS-tuple* is a 100-bit 0/1 sequence, and the similarity between two such sequences s_1 and s_2 is measured using the Jaccard coefficient, i.e. $|s_1 \cap s_2| / |s_1 \cup s_2|$. 80% of the objects form 10 clusters and the remaining 20% are noise. Two of the clusters are very small (123 and 218 objects), making the problem of finding them very challenging. The reachability plot obtained when clustering the whole data set using OPTICS is depicted in Figure 9 (b) (the two tiny clusters are indicated by arrows).



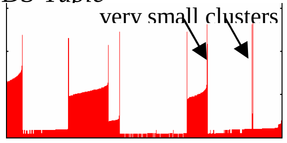
(a) OPTICS output for *DS-Vector*



(b) OPTICS output for *DS-Tuple*



(c) OPTICS-Rank output for *DS-Vector*



(d) OPTICS-Rank output for *DS-Tuple*

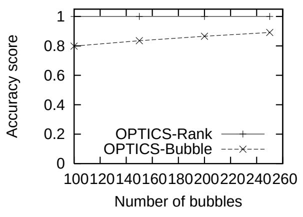
Figure 9: The Reachability plots from OPTICS (a and b) and OPTICS-Rank (c and d) are almost identical.

The outputs of OPTICS-Rank for *DS-Vector* and *DS-Tuple* are shown in Figure 9 (c) and (d) respectively. For the parameters of OPTICS-Rank, the number of top objects to return in each ranking, *k*, is set to 5, and the step limit *s* for the best-first frontier search is set to 10. The number of leaf nodes for the pivot tree is set to 5000, and the branching factor is set to 10. The plots generated by OPTICS-Rank are almost identical to those generated by OPTICS, only that some clusters have switched position, which is a normal phenomenon when clustering with OPTICS and which does not affect the clustering accuracy. OPTICS-Rank uses only a fraction of the total number of distances used by OPTICS. The number of distances computed by OPTICS is 2.5x10^9 for both data-set, while OPTICS-Rank uses 2.4x10^6 and 2.7x10^6 distances for *DS-Vector* and *DS-Tuple* respectively.

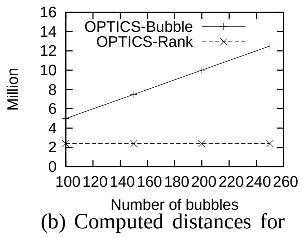
We compare the clustering accuracy of OPTICS-Rank and OPTICS-Bubble on *DS-Vector* and *DS-Tuple* using the measurement defined in [16]. The measure evaluates different cut-lines through a reachability plot in equidistant intervals, and selects the cut-line that corresponds most closely to the clustering obtained for the whole data set. This cut-line is assigned a score based on the number of clusters that are present with respect to this cut through the reachability plot. If *k* clusters are found (0 ≤ *k* ≤ maximum number of clusters in the original data set, *k_max*), then the cut-line gets a score of *k/k_max*.

The clustering accuracy on *DS-Vector* is shown in Figure 10 (a). OPTICS-Rank uses a fixed setting as mentioned above while the number of bubbles used by OPTICS-Bubble varies from 100 to 250. The experiment is repeated 10 times and OPTICS-Rank always succeeds to find all the clusters so that it has a score of 1. This accuracy is consistently better than that of OPTICS-Bubble. The corresponding numbers of computed distances by the two algorithms are shown in Figure 10 (b). As the number of bubbles increases, the number of distances computed by OPTICS-Bubble increases linearly. It uses as many as 12.5x10^6/2.4x10^6 ≈ 5.2 times amount of distances as OPTICS-Rank.

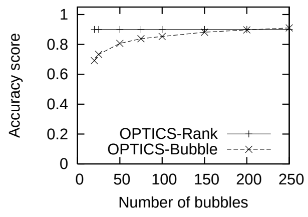
The clustering accuracy on *DS-Tuple* is shown in Figure 11 (c). OPTICS-Rank uses the same setting as in the previous experiment and the number of bubbles used by OPTICS-Bubble varies from 20 to 250. Similar to the previous experiment, OPTICS-Rank outperforms OPTICS-Bubble and is only matched by the latter when the number of bubbles reaches 200. The numbers of computed distances for both methods are shown in Figure 10 (d). It shows that when we use a larger amount of bubbles (≥ 200) for OPTICS-Bubble to match the accuracy of OPTICS-Rank, OPTICS-Bubble will need to perform many more distance computations.



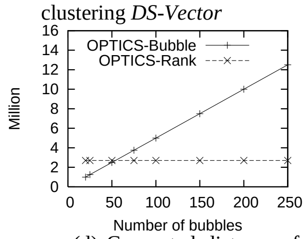
(a) Accuracy on *DS-Vector*



(b) Computed distances for clustering *DS-Vector*



(c) Accuracy on *DS-Tuple*



(d) Computed distances for clustering *DS-Tuple*

Figure 10: Clustering accuracy and number of distances computed.

6.2 Real-life Data

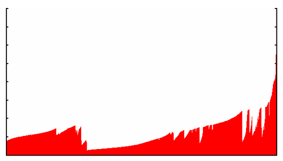
The first real-life data-set we used, denoted by *DS-Protein*, are all the 170,158 short structure motifs of length 4 extracted from 755 protein structures. The distance function we use is the rmsd structure alignment score. Another real-life data-set we use, denoted by *DS-Jaspar*, consists of 73,253 DNA sequence motifs extracted from the first human chromosome using the transcription factor binding patterns in the JASPAR database [13]. The distance function we use is the average mutual edit (Levenstein) distance for all sequences in two motifs [12]. The pairwise distances for this data-set were pre-computed using massive parallel processing.

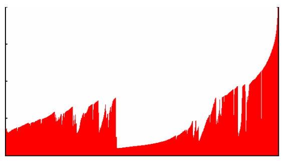
6.2.1 Protein Structure Alignment Data

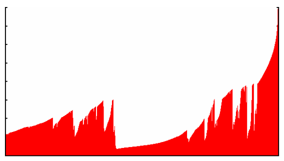
The resulting plots of applying different algorithms on the whole data-set of *DS-Protein* are shown in Figure 11.

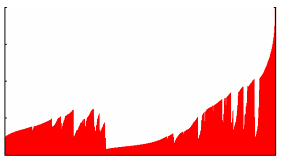
For OPTICS-Rank, the parameter *k* is set to the smallest possible value of 1, and the frontier search step limit *s* is varied. Any larger values of *k* will improve the clustering accuracy but in the worst case increase the number of distance computations linearly. For OPTICS-Bubble, the number of bubbles it uses, denoted by *b*, is set to 500 and 5000.

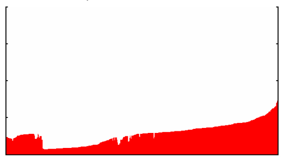
Figure 11 (a) is the output of OPTICS. Figure 11 (b-d) show the outputs of OPTICS-Rank, with *s* = 5, 100, and 1000 respectively. They show that the plots of OPTICS-Rank are very similar to that of OPTICS, and increasing the step limit *s* will not significantly improve the accuracy. Figure 11 (e) and (f) are the plots of OPTICS-Bubble. They are less similar to the OPTICS output than those of OPTICS-Rank, showing that even with as many as 5000 data bubbles, much of the actual clustering structure is lost.


(a) Result of OPTICS, runtime = 695 min.


(b) Result of OPTICS-Rank, *s* = 5, runtime = 6 min.


(c) Result of OPTICS-Rank, *s* = 100, runtime = 36 min.


(d) Result of OPTICS-Rank, *s* = 1000, runtime = 240 min.


(e) Result of OPTICS-Bubble, *b* = 500, runtime = 14 min.

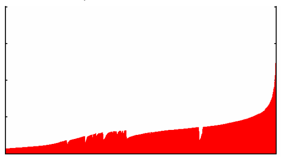

(f) Result of OPTICS-Bubble, *b* = 5000, runtime = 147 min.

Figure 11: Reachibility plots on *DS-Protein* for OPTICS, OPTICS-Rank, and OPTICS-Bubble.

The runtime and the number of computed distances of the three methods are shown in Table 1. It shows that OPTICS-Rank computes far fewer distances than both OPTICS and OPTICS-Bubble. It also shows that increasing *s* can only slightly decrease the number of computed distances while dramatically increasing the runtime. Therefore, in practice, using a small value of *s* has a better balance of accuracy and runtime.

Table 1: Runtime and computed distances for OPTICS, OPTICS-Rank and OPTICS-Bubble

Method	Runtime (min)	Number of Computed Distances (million)
OPTICS	695	14476.9
OPTICS-Rank, s=5	6	7.2
OPTICS-Rank, s=10 0	36	7.1
OPTICS-Rank, s=1000	240	7.0
OPTICS-Bubble, b=500	14	85.0
OPTICS-Bubble, B=5000	147	838.3

Figure 12 shows the scalability of OPTICS-Rank with respect to the size of the data-set. It shows that with a cut-off step limit set to 10, OPTICS-Rank achieves a sub-quadratic time complexity in practice.

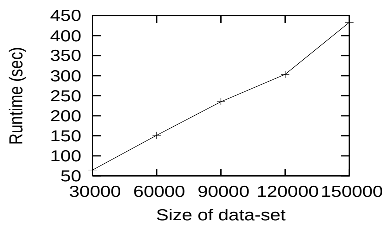
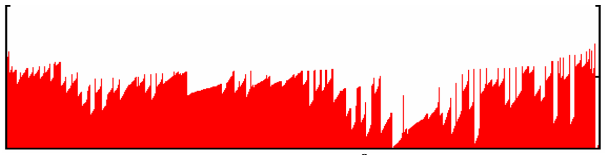


Figure 12: Scalability of OPTICS-Rank w.r.t. the size of the *DS-Protein* data-set. *k*= 1, *s* = 10.

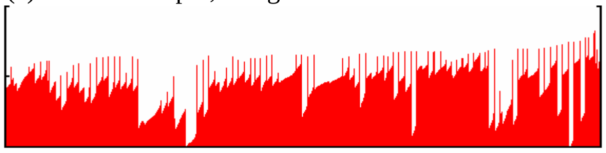
6.2.2 JASPAR Data

The clustering results on *DS-Jaspar* are depicted in Figure 13. While using 1000 times less distances, OPTICS-Rank generates a plot that captures the same cluster structure as the output of the original OPTICS (with some switching of cluster positions).

In order to measure the accuracy numerically, we apply the F-score measure [9] on the clustering results. The F-score is defined as $F = 2*p*r/(p+r)$, where *p* is precision and *r* is recall. The closer an F-score is to 1, the better is the result. To determine the F-scores, we manually extracted the clusters from both output plots (98 clusters for OPTICS and 101 clusters for OPTICS-Rank), and each cluster in the OPTICS output is matched to the cluster in the OPTICS-Rank output that has the highest F-score. The F-score distribution of the matched clusters is shown in Figure 14. It shows that the majority of the clusters in the OPTICS output can be matched with a cluster in the OPTICS-Rank output with an F-score of more than 0.95. The average F-score weighted by the size of the cluster in the OPTICS output is 0.86.



(a) OPTICS output, using 5.4x10^9 distances.



(b) OPTICS-Rank output, using 4.9x10^6 distances.

Figure 13: Clustering results on *DS-Jaspar*.

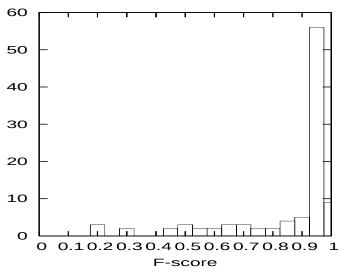


Figure 14: Distribution of F-score. The weighted F-score is 0.86.

7. Conclusions

In this paper, we proposed a novel approach to perform approximate clustering with high accuracy. We introduced a novel pairwise hierarchical ranking method to efficiently determine close neighbors for every data object. We also proposed a frontier search rather than a sequential scan in the naïve ranking to reduce the search space and a heuristic that approximates the frontier search but further reduces the runtime. Empirical results on synthetic and real-life data showed the high efficiency and accuracy or our method in combination with OPTICS, obtaining a speedup up to two orders of magnitude over OPTICS while maintaining a very high accuracy and up to one order of magnitude over DATA BUBBLES combined with OPTICS while obtaining a much more accurate result.

Acknowledgment

- We are grateful to Gordon Robertson and Mikhail Bilenky in Canada's Michael Smith Genome Sciences Centre for fruitful discussions and providing us the *DS-Jaspar* data-set.
- Research partially funded by NSERC and CFI.

8. References

[1] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. SIGMOD'99, pp. 49-60.

[2] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff. Query-sensitive embeddings. SIGMOD'05, pages 706–717.

[3] M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering. SIGMOD'01, pp. 79-90.

[4] C. Elkan. Using the Triangle Inequality to Accelerate k-Means. ICML'03, pp. 147-153.

[5] E. Chavez, G. Navarro, R. A. Baeza-Yates, and J. L.Marroquın. Searching in metric spaces. ACM Comput. Surv., 33(3):273–321, 2001.

[6] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. KDD'96, pp. 226-231.

[7] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *TODS*, 28(4):517–580, 2003.

[8] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2001.

[9] B. Larsen and C. Aone. Fast and Effective Text Mining Using Linear-time Document Clustering. KDD'99, 16-22.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. VLDB'97, pp. 426-435.

[11] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach*, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey, 1995.

[12] G. Robertson, M. Bilenky, K. Lin, A. He, W. Yuen, M. Dagpinar, R. Varhol, K. Teague, O. L. Griffith, X. Zhang, Y. Pan, M. Hassel, M. C. Sleumer, W. Pan, E. D. Pleasance, M. Chuang, H. Hao, Y. Y. Li, N. Robertson, C. Fjell, B. Li, S. B. Montgomery, T. Astakhova, J. Zhou, J. Sander, A. S. Siddiqui, and S. J. M. Jones. cisRED: a database system for genome-scale computational discovery of regulatory elements. *Nucleic Acids Research*, 34: D68-D73, 2006.

[13] G.D. Stormo. DNA binding sites: representation and discovery. *Bioinformatics*. 2000 Jan;16(1):16-23

[14] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. Pattern Recogn. Lett., 4(3):145–157, 1986.

[15] T. Zhang, R. Ramakrishnan, Linvy M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. SIGMOD'96, pp. 103-114.

[16] J. Zhou and J. Sander. Data bubbles for non-vector data: Speeding-up hierarchical clustering in arbitrary metric spaces. VLDB'03, pp 452–463.