Distributed Antipole Clustering for Efficient Data Search and Management in Euclidean and Metric Spaces

Alfredo Ferro[1], Rosalba Giugno[1], Misael Mongiovì[1,2], Giuseppe Pigola[1], and Alfredo Pulvirenti[1]

[1]University of Catania
Dept. of Mathematics and Computer Science
Viale A. Doria, 6, 95125 Catania, Italy
{ferro,giugno,mongiovi,pigola,apulvirenti}@dmi.unict.it

[2]Proteo S.p.A.
Research and Development Department
via S. Sofia, 65 Catania, 95123, Italy
mongiovi@proteo.it

Abstract

In this paper a simple and efficient distributed version of the recently introduced Antipole Clustering algorithm for general metric spaces is proposed. This combines ideas from the M-Tree, the Multi-Vantage Point structure and the FQ-Tree to create a new structure in the "bisector tree" class, called the Antipole Tree*. Bisection is based on the proximity to an "Antipole" pair of elements generated by a suitable linear randomized tournament. The final winners (A, B) of such a tournament are far enough apart to approximate the diameter of the splitting set. A simple linear algorithm computing Antipoles in Euclidean spaces with exponentially small approximation ratio is proposed. The Antipole Tree Clustering has been shown to be very effective in important applications such as range and k-nearest neighbor searching, mobile objects clustering in centralized wireless networks with movable base stations and multiple alignment of biological sequences. In many of such applications an efficient distributed clustering algorithm is needed. In the proposed distributed versions of Antipole Clustering the amount of data passed from one node to another is either constant or proportional to the number of nodes in the network. The Distributed Antipole Tree is equipped with additional information in order to perform efficient range search and dynamic clusters management. This is achieved by adding to the randomized tournaments technique, methodologies taken from established systems such as BFR and BIRCH*. Experiments show the good performance of the proposed algorithms on both real and synthetic data.*

1 Introduction

Mining and indexing are basic problems in knowledge discovery in general metric spaces. Much efforts have been spent both in clustering algorithms (see BIRCH [73], DBSCAN [27], CLIQUE [3], BIRCH* [32], WaveClusters [65], CURE [38], and CLARANS [56]), and in the development of new indexing techniques (see, for instance, MVP-Tree [13], M-Tree [24], SLIM-Tree [67], FQ-Tree [6], R-tree [39], R*-tree [10], List of Clusters [21], SAT [55]; the reader is also referred to [22] for a survey on this subject). For the special case of Euclidean spaces, one can see [2, 35, 12], X-Tree [11], and CHILMA [66].

Let $(M, dist)$ be a finite metric space, let S be a subset of M and suppose the aim is to split it into the minimum possible number of clusters whose radii should not exceed a given threshold σ [42, 37, 29, 61]. The Antipole Tree [19] belongs to the class of "bisector trees" [22, 16, 57], which are binary trees whose nodes represent sets of elements to be clustered. Moreover, it combines and extends ideas from the M-Tree, MVP-Tree, and FQ-Tree structures. Its construction begins by first allocating a root r and then selecting two splitting points c_1, c_2 in the input set, which become the children of r. Subsequently, the points in the input set are partitioned according to their proximity to the points c_1, c_2. Then one recursively constructs the tree rooted in c_i associated with the partition set of the elements closer to c_i, for $i = 1, 2$. A good choice for the pair (c_1, c_2) of splitting points consists in maximizing their distance. Antipole Tree Clustering uses a simple approximate algorithm based on tournaments of the type described in [9]. The tournament is played as follows. At each round, the winners of the previous round are randomly partitioned into subsets of a fixed size τ and their 1-medians are discarded.

Rounds are played until one is left with less than 2τ el-

1-4244-0054-6/06/$20.00

ements. The farthest pair of points in the final set is the Antipole pair of elements.

The Antipole Tree Clustering has been shown to be very effective in important applications such as range and k-nearest neighbor searching [19], mobile objects clustering in centralized wireless networks with movable base stations [30] and multiple alignment of biological sequences [60, 59]. In many of such applications an efficient distributed clustering algorithm is needed. For example in pure Ad-Hoc wireless networks with movable base stations, objects clustering is dynamically executed by the base stations network. Data management of a sensors network may require distributed data clustering. Finally, multiple alignment of large sets of biological sequences may be out of the range of sequential systems such as ClustalW. In nowadays applications data may be located in different computers which are connected to each other through a local or wide area networks. Clustering or indexing by sending the data to a central data is often impossible. Much effort has been done in both clustering and indexing in distributed and parallel systems (see Section 2).

In this paper a distributed version of the Antipole Tree Clustering algorithm is proposed. In the case of Euclidean spaces a simple distributed approximate diameter computation is performed by computing the enclosing box. The furthest pair of points in the enclosing box is the Antipole pair. This requires linear time computation in each node and a constant amount of transmitted data which is proportional to the number of nodes. The number of items passed from one node to another is proportional to the number of cartesian reference axes used in the enclosing box computation. On the other hand the approximation error exponentially decreases with such number. Finally, in general metric spaces the global Antipole pair is computed by taking the farthest pair of elements among the winners of all local nodes tournaments.

The proposed distributed version of the Antipole Tree Clustering may be used to compute range searches. In order to do that, clusters and centroids may be used to prune the process using triangle inequality. Each node in the tree must contain, for each cluster, the centroid and the radius of the cluster portion contained in nodes belonging to the subtree rooted in that node. To maintain such structure boundary elements in each cluster must be stored. Moreover, such tree is shown to be adaptable to perform dynamic clustering management by adding the most central elements in each cluster fragment. This choice has been suggested by established systems such as BIRCH* [32] and BFR [28] .

The paper is organized as follows. In the next Section, a briefly review of relevant previous work is given. The Antipole Tree Clustering is described in Section 3. Here, for the first time an analysis of the approximation ratio of the algorithm in Euclidean space is given. In Section 4 the distributed version of the Antipole Tree Clustering is introduced for both Euclidean and general metric spaces. Section 5 describes a procedure for searching on the Distributed Antipole Tree. In Section 6 clustering management is described. In Section 7 experiments on large datasets are reported. Finally, Section 8 concludes the paper and suggests future research directions.

2 Related Works

Large scale clustering is one of the widely used mining and indexing techniques.

Antipole Tree belongs to the class of hierarchical clustering algorithms such as BIRCH [73], BIRCH* [32] and CURE [38]. BIRCH [73] uses a threshold on the clusters radius and organizes the data in a CF-tree. Such tree is used to guide data points insertions. BIRCH* [32] extends BIRCH to general metric spaces. Each leaf node stores the centroid, the radius, the size of the cluster and the $rowsum$ of the centroid. $rowsum$ is the summation of the square distances from a point to each element of the cluster. In order to maintain cluster the p farthest (resp. closest) elements to the centroid and their $rowsum$ are stored. Each internal node stores representative elements of its subtrees clusters. The p farthest elements are candidates to be the new centroids of two clusters merging. On the other hand, the p elements closest to the centroids are candidates to be the new centroids when dynamic insertion is executed. In Euclidean spaces, an interesting algorithm is BFR (Bradley-Fayyad-Reina) [28]. It is a generalization of k-means [52]. Each cluster has a core of points (discard set) definitively belonging to it. There are also partial aggregation of points (compression set) which will be assigned to the closest cluster. Each aggregation of points (discard or compression) is represented by the following statistics: the size of the cluster, the vector of the summations of each coordinate and the vector of the summations of the square of each coordinate. Such information is sufficient to compute the center of gravity and the standard deviation (radius). This is enough to evaluate the area in which most of the points in the cluster are located. Other revelent cluster algorithms are for example DBSCAN [27], CLIQUE [3], WaveClusters [65], and CLARANS [56]. A survey is given in [40].

In [71, 26] parallel versions of DBSCAN and k-means algorithms are proposed, respectively. Both algorithms proceed in the same way. Data are clustered in a central site and distributed among all processors. Central site data may be organized in suitable data structure such as R*-tree [10]. Parallel algorithms for hierarchical clustering are proposed in [58].

In distributed environments, data are located in different independent sites and can not be collected to a central site. On the other hand, the central site can aggregate

partial information coming from nodes to perform global computation such as clustering. The outcome of the central site elaboration are then broadcasted to all nodes which may compute local clusters fragments. In [5, 45] a distributed density-based clustering algorithm which makes use of DBSCAN [27] is proposed. In [69] a distributed version of the k-means clustering is presented. In this context each agent contains a subset of the attributes of n entities. The goal is to obtain k clusters of entities maintaining privacy and security. This means that, for each entity, every agent knows its cluster and only those attributes associated with that agent. Other distributed density based algorithms are proposed in [70, 48]. Similar distributed clustering approaches which take into account constraints on the type of information that nodes may share are proposed in [46, 34].

Mobile wireless networks management strongly requires efficient distributed clustering [33, 50, 8, 51, 4, 4]. In these models network topology may change. Moreover, often the network must be self organizing and a leader node may not be required.

Clustering is also a basic step in indexing large amount of data to perform efficient search and management. The Antipole Tree is a recently proposed structure which has been shown to efficiently accomplish such tasks. The structure combines and extends ideas from the M-Tree [24, 23], the Multi-Vantage Point structure [13] and the FQ-Tree [6].

The FQ-Tree [6], an example of a structure using pivots (see [22] for an extended survey), organizes the items of a collection ranging over a metric space into the leaves of a tree data structure. FQ-Trees maintains a list of reference objects and their distances from all objects in the data set.

M-Trees [24, 23] are dynamically balanced trees. Each parent node corresponds to a cluster with a radius and every child of that node corresponds to a subcluster with a smaller radius.

VP-Trees [68, 72] organize items coming from a metric space into a binary tree. The items are stored both in the leaves and in the internal nodes of the tree. The items stored in the internal nodes are the "vantage points". The processing of a query requires the computation of the distance between the query point and some of the vantage points.

The Multi-Vantage-Point tree [13] is an intellectual descendant of the vantage point tree and the GNAT [14] structure. The fundamental idea is that, given a point p, one can partition all objects into m partitions based on their distances from p, where the first partition consists of those points within distance d_1 from p, the second consists of those points whose distance is greater than d_1 and less than or equal to d_2, etc.

Another relevant recent work, due to Chávez et al. [21], proposes a structure called List of Clusters. Such list is constructed in the following way: starting from a random point, a cluster with bounded diameter (or limited number of objects) centered in that random point is constructed. Then such a process is iterated by selecting a new point, for example the farthest from the previous one, and constructing another cluster around it. The process terminates when no more points are left.

In all the above methods, triangle inequality is used to prune the tree during the search. Antipole Tree uses a similar idea except that the reference objects are the centroids of clusters.

Concerning Euclidean spaces a widely used data structure, R-Tree, was proposed in [39]. The objects are enclosed in minimum bounding rectangles. The hierarchical division of the space in rectangles is represented in the tree. Given a query Q, during the search, subtrees corresponding to rectangles which do not intersect Q are pruned. Other relevant results on indexing data structures are presented in [15, 25, 31, 36, 64, 63, 67, 55].

Several works have been done in distributed indexing structure. In [43] parallel Quadtree is used to join distributed data. In [47] a distributed version of the R-tree in multi-disk machine is proposed. Equal distribution of nodes in the disks is achieved. In [49] a parallel version of R-tree with data distributed among several computers is proposed. One selected processor stores only R-tree internal nodes. Leaves are stored in the remaining computers. An extension in which processors are allowed to contain internal nodes is proposed in [62]. Versions which support join operations on the base of the spatial access to objects were proposed in [62, 54, 53].

3 The Antipole Tree Clustering in Euclidean and General Metric Spaces

The Antipole Clustering [19] of bounded radius σ is performed by a recursive top-down procedure starting from the given finite set of objects S and checking at each step if a given splitting condition Φ is satisfied. If this is not the case, then splitting is not performed, the given subset is a cluster, and a centroid having distance approximatively less than σ from every other object in the cluster is computed by the procedure described in Section 3.1.

Otherwise, if Φ is satisfied then a pair of objects (A, B) of S called the Antipole pair is generated by the algorithm described in Section 3.2. It is used to split S into two subsets S_A and S_B obtained by assigning each object O of S to the subset containing the endpoint of the Antipole (A, B) closest to O. The splitting condition Φ states that $dist(A, B)$ is greater than the cluster diameter threshold corrected by the error coming from the Euclidean case analysis described in Section 3.3. The tree obtained by the above procedure is called an Antipole Tree (the pseudo-code is reported in Fig. 1). All nodes are annotated with the Antipole endpoints

and the corresponding cluster radius; each leaf contains also the 1-median of the corresponding final cluster.

In Euclidean spaces, the construction of the Antipole Tree differs from the one reported in Fig. 1 in the Antipole pair computation (see Section 3.3). This is obtained by replacing the procedure APPROX_ANTIPOLE with the procedure APPROX_DIAGONAL of Fig. 5. Moreover, the centroid of each cluster is the center of the enclosing box (see Section 3.3).

The Antipole Tree Clustering Algorithm

```
BUILD(S, σ)
1  Q ← APPROX_ANTIPOLE(S);
2  if Q = ∅ then // splitting condition Φ fails
3     T.Centroid ← APPROX_1_MEDIAN(S);
4     T.Radius ← max_{x ∈ S} dist(x, Cluster.C)
5     T.Cluster ← S;
6     return T;
7  end if;
8  {A,B} ← Q;
9  S_A ← {O ∈ S | dist(O, A) < dist(O, B)};
10 S_B ← {O ∈ S | dist(O, B) ≤ dist(O, A)};
11 T.left ← BUILD(S_A,σ);
12 T.right ← BUILD(S_B,σ);
13 return T;
14 END BUILD.
```

Figure 1. The Antipole Tree Clustering algorithm.

3.1 1-Median computation in general metric spaces

In this Section, a randomized algorithm for the approximate 1-median selection [17] is reported. It is an important subroutine in the Antipole Tree construction. It is based on a tournament played among the elements of the input set S. At each round, the objects which passed the preceding turn are randomly partitioned into subsets, say $X_1, \ldots, X_k$. Then, each subset X_i is locally processed by a procedure which computes its exact 1-median x_i. The objects $x_1, \ldots, x_k$ move to the next round. The tournament terminates when a single object $\overline{x}$ is left, that is the final winner. The winner approximates the exact 1-median in S. Fig. 2 contains the pseudocode of this algorithm. The local optimization procedure 1-MEDIAN (X) returns the exact 1-median in X. A running time analysis (see [17] for details) shows that above procedure takes time $\frac{t}{2}n + o(n)$ in the worst-case. An optimization in terms of precision of 1-median computation can be found in [18].

The approximate 1-Median selection algorithm

```
1-MEDIAN (X)
1  for each x ∈ X do
2     S_x ← Σ_{y ∈ X} dist(x, y);
3  Let m ∈ X be an element
   such that S_m = min_{x ∈ X}(S_x);
4  return m

APPROX_1_MEDIAN (S)
1  while |S| > threshold do
2        W ← ∅;
3        while |S| ≥ 2τ do
4              Choose randomly a subset T ⊆ S, with |T| = τ;
5              S ← S \ T;
6              W ← W ∪ {1-MEDIAN (T)};
7        end while;
8        S ← W ∪ {1-MEDIAN (S)};
9  end while;
10 return 1-MEDIAN (S);
```

Figure 2. The 1-Median selection algorithm.

3.2 The Approximate Diameter (Antipole) Computation in General Metric Spaces

The *diameter computation problem* or *furthest pair problem* is to find the pair of points (A, B) in S such that $dist(A, B) \geq dist(x, y),\ \forall x, y$ in S.

As observed in [44], a metric space where all distances among objects are set to 1 except for one (randomly chosen) which is set to 2, can be constructed. In this case any algorithm that tries to give an approximation factor greater than $1/2$ must examine all pairs, so a randomized algorithm will not necessarily find that pair.

Nevertheless, a good outcome in nearly all cases is expected. A randomized algorithm [19] inspired by the one proposed for the 1-median (see Section 3.2) is reported. In this case, each subset X_i is locally processed by a procedure LOCAL_WINNER which computes its exact 1-median x_i and then returns the set $\overline{X}_i$, obtained by removing the element x_i from X_i. The elements in $\overline{X}_1 \cup \overline{X}_2 \ldots \cup \overline{X}_k$ are used in the subsequent step. The tournament terminates when a single set is left, $\overline{X}$, from which the final winners (A, B), as the furthest objects in $\overline{X}$, are obtained. The pair (A, B) is called the *Antipole pair* and their distance represents the approximate diameter of the set S.

The pseudocode of the Antipole algorithm APPROX_ANTIPOLE, similar to that of the 1-Median algorithm, is given in Fig. 3. Given an input set of size $n \in \mathbb{N}$, a constant tournament size $\tau \geq 3$, and a threshold $\vartheta = o(\sqrt{n})$, the Antipole selection algorithm performs $\frac{\tau(\tau-1)}{2}n + o(n)$ distance computations. A faster (but less accurate) variant of APPROX_ANTIPOLE is given in [19, 18].

The approximate Antipole selection algorithm

```
LOCAL_WINNER(T)
1  return T \ 1-MEDIAN(T);
2  END LOCAL_WINNER

FIND_ANTIPOLE(T)
1  return P1, P2 ∈ T such that
          dist(P1, P2) ≥ dist(x, y)
          ∀x, y ∈ T;
2  END FIND_ANTIPOLE

APPROX_ANTIPOLE(S)
1  while |S| > threshold do
2        W ← ∅;
3        while S ≥ 2τ do
4              Choose randomly a subset T ⊆ S : |T| = τ;
5              S ← S \ T;
6              W ← W ∪ {LOCAL_WINNER(T)};
7        end while
8        S ← W ∪ {LOCAL_WINNER(S)};
9  end while
10 return FIND_ANTIPOLE(S);
11 END APPROX_ANTIPOLE
```

Figure 3. The pseudocode of Antipole Algorithm. τ is the tournament size.

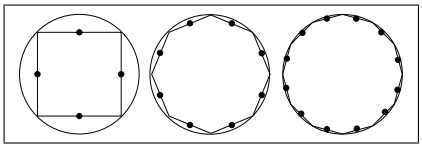


Figure 4. These pictures show the worst cases in the first three iterations of the algorithm.

```
APPROX_DIAGONAL(S, δ)
1  Let BBox = {P_{X_m}, P_{X_M}, P_{Y_m}, P_{Y_M}}
   be the minimum bounding box of S;
2  V ← {{S}};
3  for i = 1 to ⌈ π / (4 × arccos(1−δ)) − 1 ⌉ do
4      V' = ROTATE_SET V, π/2^{i+1} ;
5      Let BBox_{π/2^{i+1}} = {P_{X_m}, P_{X_M}, P_{Y_m}, P_{Y_M}}
       be the minimum bounding box of the rotated sets in V';
6      V = Set catalog of V';
7      BBox = BBox ∪ BBox_i;
8  end for
9  return FIND_ANTIPOLE(BBox);
```

Figure 5. Algorithm for the Pseudo-Diameter Computation.

3.3 Sequential Approximate Diameter Computation in Euclidean Spaces

In this Section, an approximation algorithm for the approximate diameter computation of a finite set of points in the Euclidean plane is proposed. Several studies in the literature [1, 7, 41, 20] have provided efficient algorithms for the approximate diameter computation in multidimensional Euclidean Spaces. The proposed approach can be regarded as the binary search version of [1]. For the sake of simplicity, let's start with a finite set of points in the plane S. The endpoints of an approximate diameter constitute the Antipole. An Antipole of S is computed as follows. Let (P_{X_m}, P_{X_M}), (P_{Y_m}, P_{Y_M}) be the four points of S having minimum and maximum Cartesian coordinates: the so called minimum area bounding box *BBox*. Notice that such four points belong to the convex hull of the set S and all of S is included in the rectangle bounded by $(P_{X_M}.x - P_{X_m}.x)$ and $(P_{Y_M}.y - P_{Y_m}.y)$. The two endpoints (A, B) of the diameter of such four points constitute the Antipole. Let $Diagonal$ be the longest diagonal in *BBox* and $Diameter$ be the exact diameter of S. The Antipole distance (called $Pseudo\_Diameter$) is not less than $Diagonal/\sqrt{2}$. This yields $\frac{Pseudo\_Diameter}{Diameter} \geq \frac{1}{\sqrt{2}}$ proving that the approximation ratio in the plane is $1 - 1/\sqrt{2}$.

Next, a generalization of this method is reported. It gives an approximation algorithm able to obtain an exponentially arbitrary low approximation ratio δ for the real diameter (see the pseudocode in Fig. 5). A $\pi/4$ rotation of the Cartesian coordinates is performed. It implies a bisection of the axes, and compute the maximum and minimum coordinate points for such two new axes. 8 points are obtained. Let (A, B) be the diameter of this set. It is easy to see (middle picture in Fig. 4) that $dist(A, B)/\cos\frac{\pi}{8} > Diameter$. By iterating the bisecting process d times, $dist(A, B)/\cos\frac{\pi}{2^{d+2}} > Diameter$. Therefore the approximation ratio introduced by the algorithm is:

$$\delta = \frac{|Diameter - Pseudo\_Diameter|}{Diameter} \leq \left|1 - \cos\frac{\pi}{2^{d+2}}\right|.$$

Hence, the following conclusion holds:

Theorem 3.1 *Let S be a set of points in the plane and let $0 < \delta \leq 1 - \sqrt{2}/2$. Then a call to* APPROX_DIAGONAL(S, δ) *returns an Antipole pair (A, B) ($Pseudo\_Diameter$) which approximates the diameter with a error bounded by δ.* ■

4 Distributed Antipole Construction in Euclidean and General Metric Spaces

In this Section, a distributed version of the Antipole Clustering algorithm (see Section 3) is proposed.

Let assume n be the number of nodes storing a finite set S of m objects belonging to an Euclidean or general metric space. Suppose that, there is a specific interconnection topology. Nodes communicates through messages and each message is correctly transmitted to a unique successor (which may have several predecessors). There is one designated final node f aggregating the results of all local computations. f will also broadcasts the result of this aggregation to all nodes. Clustering is locally performed on the basis of the broadcasted data. This will produce a final set of clusters piecewise distributed among the nodes.

The Distributed Antipole Clustering (DAC) algorithm proceeds as follows.

4.1 Distributed Antipole Construction in Euclidean spaces

Each node j computes the enclosing box $\{P^j_{X_m}, P^j_{X_M}, P^j_{Y_m}, P^j_{Y_M}\}$ of its local set of points S^j. Each local enclosing box is passed to the successor node which will merge all received enclosing boxes with its local enclosing box. This merging is nothing else than the enclosing box of the enclosing boxes. It is immediate to see that the final node f will contain the global enclosing box $\{P_{X_m}, P_{X_M}, P_{Y_m}, P_{Y_M}\}$. f computes the farthest pair (global Antipole) of points (A, B) in this enclosing box. If $dist(A, B)$ is smaller than the cluster threshold then a termination message is broadcasted to all nodes. Otherwise, cluster splitting is performed in the following way. (A, B) is broadcasted to all nodes. Each node will split the local data using the global Antipole (A, B). The algorithm proceeds recursively until splitting is no longer performed.

4.2 Distributed Approximate Diameter (Antipole) Computation in General Metric Spaces

If objects belong to a general metric space equipped with a distance function then a slightly different procedure must be used. Each node j computes the Antipole pair (A^j, B^j) of its local set of objects S^j by the procedures APPROX_ANTIPOLE (see Fig. 3). Each local Antipole pair is passed to the successor node which will add its local Antipole pair to the set of all received Antipole pairs. The final node f will computes the farthest pair (global Antipole) (A, B) of objects in the set of all local Antipole pairs. The algorithm will proceed as in the Euclidean space.

5 Efficient Distributed Searching in Metric Spaces

Range and k-nearest neighbor searching are core problems in knowledge discovery. In order to efficiently perform these searches clusters and centroids may be used to prune the process using triangle inequality. More precisely if $|dist(q, c) - r| > t$ then every element x in the cluster with centroid c and radius r has distance more than the threshold t from the query q. Symmetrically, if $dist(q, c) + r <= t$ then every element x in this cluster is part of the query answer. Otherwise if $|r - t| =< dist(q, c) < r + t$ all the elements in the cluster must be checked to see whether they belong or not to the query answer.

Suppose that distributed data are clustered by the above Antipole Tree algorithm. In order to do efficient search each node in the tree must contain, for each cluster, the centroid and the radius of the portion contained in nodes belonging to the subtree rooted in that node.

Euclidean case. In Euclidean spaces, local and global cluster centroids are simply centers of local and global enclosing boxes. Similarly, local and global cluster radiuses are semi-diagonals of the local and global enclosing boxes. Each node j, for each cluster C_i, contains:

1. the local portion C_i^j of the cluster C_i;
2. the local centroid c_i^j and radius r_i^j of C_i^j;

   moreover, if j is not a leaf:

3. the centroid c_i^J and radius r_i^J of the union of all portions of C_i stored in nodes belonging to the subtree J rooted in j, C_i^j included.

Searching is performed in the following way. Starting from the final site f (see Section 3), nodes j in the interconnection topology are visited. For each cluster C_i, the following three cases may occur.

- If $|dist(q, c_i^J) - r_i^J| > t$ (resp. $dist(q, c_i^J) + r_i^J <= t$) holds, then total pruning (resp. inclusion) of the portion of C_i stored in J with root j is performed. In this case cluster C_i^J will be ignored in the successive visit of J.
- Otherwise, if $|dist(q, c_i^j) - r_i^j| > t$ (resp. $dist(q, c_i^j) + r_i^j <= t$) holds, then total pruning (resp. inclusion) of the local portion C_i^j of C_i is executed.
- Otherwise, all the objects in C_i must be examined to check whether or not they belong to the query answer.

The search proceeds until each cluster has been entirely visited.

General metric spaces case. In general metric spaces, local centroids can be computed by randomized tournaments [19] and local radius can be evaluated. In order to keep information about portion of clusters stored in subtrees a technique developed in BIRCH* [32] for cluster merging can be used. The idea is to try to minimize the number of pairwise distance computations. Let $rowsum(x)$ be the summation of the square distances of object x from all objects in its cluster. Each node j, for each cluster C_i, contains:

1. the local portion C_i^j of the cluster C_i;
2. the local centroid c_i^j and radius r_i^j of C_i^j;
3. the set F_i^j of the p farthest objects from c_i^j in C_i^j together with the $rowsum_i^j$ of each element of F_i^j;
4. the cardinality N_i^j of C_i^j;
5. the $rowsum_i^j(c_i^j)$;

   moreover, if j is not a leaf:

6. the centroid c_i^J and radius r_i^J of the union C_i^J of all portions of C_i stored in nodes belonging to the subtree J rooted in j, C_i^j included;
7. the set F_i^J of the p farthest objects from c_i^J in C_i^J together with the $approx\_rowsum_i^J$ of each element of F_i^J;
8. the cardinality N_i^J of C_i^J;
9. the $approx\_rowsum_i^J(c_i^J)$.

Items 1, 2 and 6 are used for range search as in the Euclidean case. The remaining items are used for cluster merging as follows. Suppose node j has predecessors l and m. In order to merge two portions of a cluster C_i, C_i^L and C_i^M, the $rowsum_i^J$ of all $x \in F_i^L \cup F_i^M$ should be evaluated. An exact computation would be too expensive. To overcome this problem an approximate evaluation can be applied. In [32] the applicability of Pythagorean Theorem to highly dimensional metric spaces is exploited. Let x and y_i^M be elements in F_i^L and C_i^M, respectively. Applying Pythagorean Theorem to the triangles x,y_i^M,c_i^L and c_i^L, c_i^M, y_i^M it is possible to evaluate the $approx\_rowsum_i^J$ of elements x in F_i^L by the following expression:

$$\begin{aligned} approx\_rowsum_i^J(x) = approx\_rowsum_i^L(x) + \\ + N_i^M(dist^2(x, c_i^L) + dist^2(c_i^L, c_i^M)) + \\ + approx\_rowsum_i^M(c_i^M) \end{aligned} \quad (1)$$

Similarly, let y and x_i^L be elements in F_i^M and C_i^L, respectively. Considering the triangles x_i^L,y,c_i^M and c_i^L, c_i^M, x_i^L yields to the following evaluation of $approx\_rowsum_i^J$ of elements y in F_i^M:

$$\begin{aligned} approx\_rowsum_i^J(y) = approx\_rowsum_i^M(y) + \\ + N_i^L(dist^2(y, c_i^M) + dist^2(c_i^L, c_i^M)) + \\ + approx\_rowsum_i^L(c_i^L) \end{aligned} \quad (2)$$

The element in C_i^J with minimum $approx\_rowsum_i^J$ is the new centroid c_i^J. The cardinality $N_i^J = N_i^L + N_i^M$.

The new centroid c_i^J is passed back to l and m. Each of them recursively computes their p farthest objects from c_i^J together with their approx_rowsum. Each node q in this recursion sets:

- r_i^Q to be the maximum distance received;
- F_i^Q to be the set of the p objects, among the received $2p$, farthest from c_i^J;
- the $rowsum$ of elements in F_i^Q are computed by approximation formulas 1 and 2 applied to the children of q.

The total merging of all cluster fragments can be obtained by iterating pairwise merging of subtrees and of local cluster with subtrees.

Searching proceeds as in the Euclidean case.

6 Distributed Dynamic Clustering Management in Metric Spaces

In this Section we will briefly describe how dynamic maintenance of clusters in the above models is achieved. New points are added to the original set and decision about which clusters are assigned to each point must be taken. This may imply update of clusters parameters.

Euclidean case. One very simple solution is the following. Starting with the final node f and following the connection tree, each added element is assigned to a cluster portion (local or subtree) according to center vicinity. This may involve straightforward enclosing box, center and radius updating.

A more elaborate technique may use Bradley-Fayyad-Reina (BFR) [28] methodology for dynamic management of Euclidean clusters. In this case center of gravity and standard variation of each cluster are maintained. Added elements are assigned on the basis of Mahalanobis radius evaluation of cluster vicinity.

General metric spaces case. In the case of general metric spaces, following the methodology of BIRCH* extended with randomized tournaments techniques, dynamic clusters management proceeds as follows. Add to the parameters of each node an additional information giving the q most "central" elements in each cluster. These elements are

candidates to become the new center when new elements are added to the cluster portion. This may happen since $rowsum$ of elements may change their ordering after new elements insertion. On the other hand, if an element y is added to C_i^J, $rowsum_i^J(x)$ can be easily updated by just adding $dist^2(x,y)$. Moreover, the $rowsum_i^J$ of the newly added element y may be estimated by applying Pythagorean Theorem to the triangle y, c_i^J, z where $dist(z, c_i^J) = r_i^j$:

$$approx\_rowsum_i^J(y) = N_i^J (r_i^j)^2 + N_i^J dist^2(c_i^J, y) \quad (3)$$

Moreover, if $dist(c_i^J, y) > r_i^j$ then radius must be updated.

Finally the initialization and propagation of the q most "central" elements in each cluster is described. Initially, in each local cluster fragment a randomized tournament for centroid computation will produce the final q winners including the approximate centroid. $rowsum$ of these central elements is computed.

Concerning propagation of these q elements, following Section 5, the centroid of the father node is calculated using the $approx\_rowsum$ formulas 1 and 2. This centroid is passed to the subtrees which will return the p farthest elements together with the q closest elements. As in Section 5, the $approx\_rowsum$ of these new elements are calculated by 1 and 2.

Periodically the system can check if the radius of a certain cluster exceeds some given threshold, then a splitting global Antipole procedure can be applied to that cluster, and the relative updates are executed.

7 Experimental Analysis

In this section experimental results on some of the proposed algorithms are presented. Experiments show a good performance in terms of speed-up, scalability and network traffic.

Simulation was implemented in C++ on a Pentium IV 3GH with 512MB RAM under Windows XP Professional Edition Operating System. The simulator emulates a network composed by a maximum of 10 workstations connected to each other as a binary tree. An algorithm is composed of sequential and parallel parts. The parallelism is simulated by running each parallel part of the algorithm in a different workstation. The parallel time is the maximum running time taken over all workstations. The total running time is obtained by summing the parallel and the sequential time.

Analysis was performed on both synthetic (10^5 and 10^6 elements in $\mathbb{R}^{20}$) and real data (10^5 biosequences with average sequence length 80 taken from NCBI[1] and 45×10^2 words taken from Linux dictionary). Fig. 6 reports the error of the approximate diameter computation compared to the exact one. The error, which was evaluated by varying the number of workstations, does not increase when the number of workstations increases. Comparison with the sequential Antipole Tree Clustering can be deduced by the single workstation behavior. Experimental evidence of the sequential Antipole Tree performance in searching is reported in [19].

Figs. 7, 8 and 9 report scalability, speed-up and network traffic by using different input data, respectively. Each workstation has the same input size. Therefore by increasing the number of workstations the total amount of data linearly grows. Notice that an optimal scalable algorithm gives a constant running time when the number of workstations grows.

Speed-up and Network traffic were evaluated by fixing the total amount of input data and by varying the number of workstations. The speed-up is defined as the ratio between the running time of the sequential and the parallel parts of the algorithm. The best theoretical speed-up (linear speed-up) is equal to the number of workstations. However in some cases the speed-up can be super-linear due to the reduced use of memory.

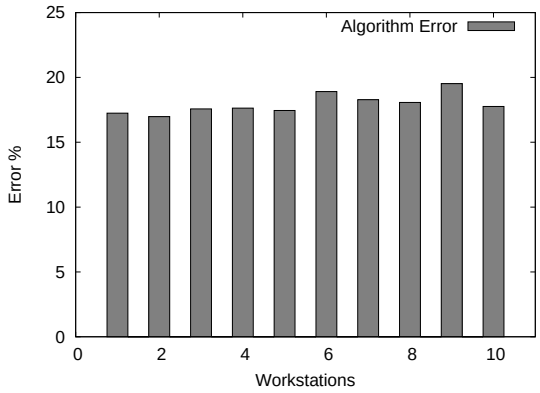
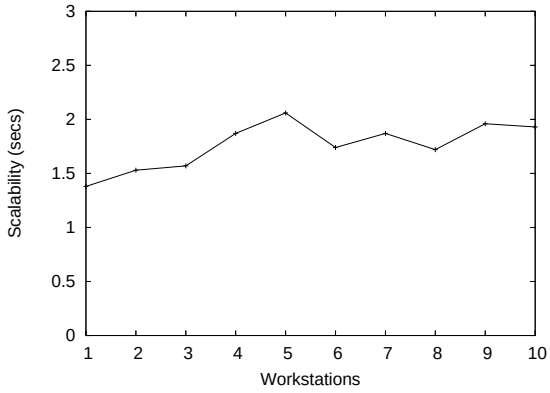


Figure 6. Error in Diameter Computation

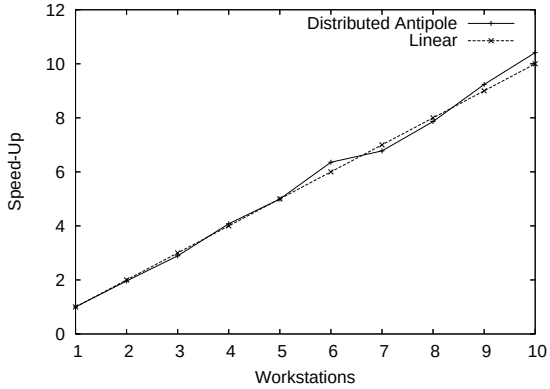
8 Conclusions and Future work

A simple and efficient distributed version of the Antipole Clustering algorithm for general metric spaces was presented. The Antipole Tree structure belongs to the "bisector tree" class. It recursively partitions a set of objects following vicinity to the endpoints of a pseudo-diametrical pair called Antipole until cluster radius exceeds a given threshold. The Antipole pair is computed by a suitable randomized tournament. In the proposed distributed versions elements are stored in network nodes and global clustering

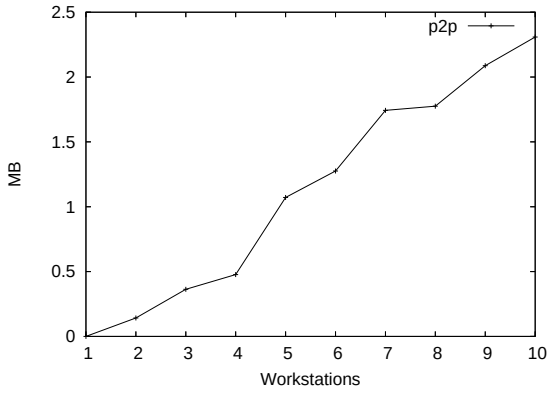
[1] http://www.ncbi.nlm.nih.gov/



(a)

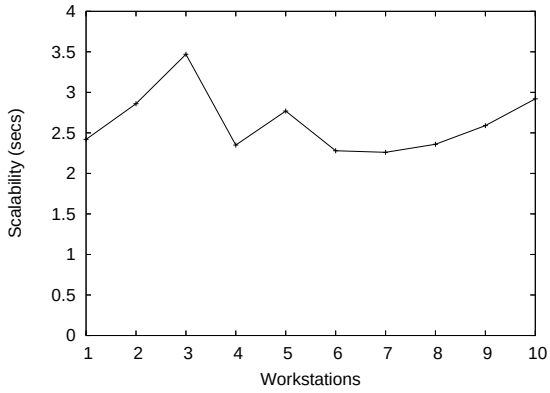


(b)

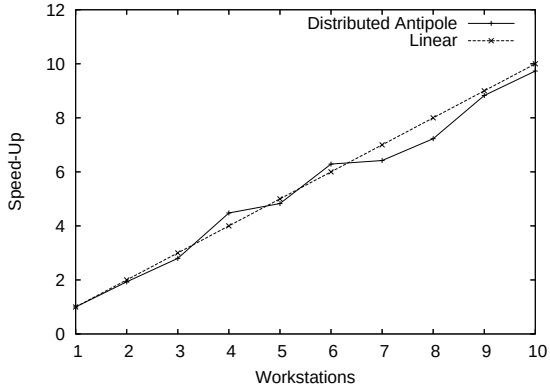


(c)

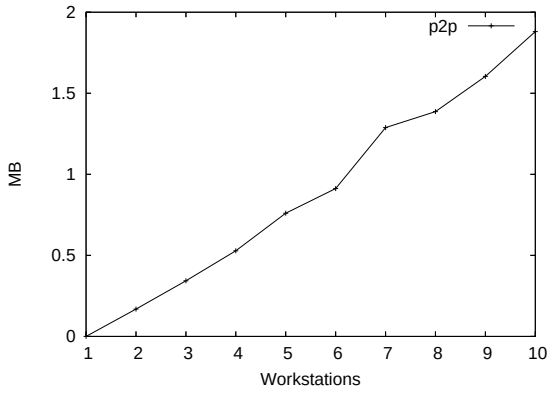
Figure 7. Scalability, speed-up and network traffic on 10^6 points in $\mathbb{R}^{20}$



(a)



(b)



(c)

Figure 8. Scalability, speed-up and network traffic on 45×10^3 Linux words.

is achieved by merging local Antipole information. Parameters describing local and subtrees clusters are stored in the network nodes. This allows efficient searching and cluster management. A mixture of randomized tournaments, BFR and BIRCH* clusters dynamic maintenance techniques is used. Experiments on both real and synthetic data show the good behavior of the proposed algorithms. Future work will focus on the tuning of the cluster radius threshold with respect to the number of processors and network topology. Finally, adding vertical fragmentation to deal with partial knowledge of attributes will be also considered.

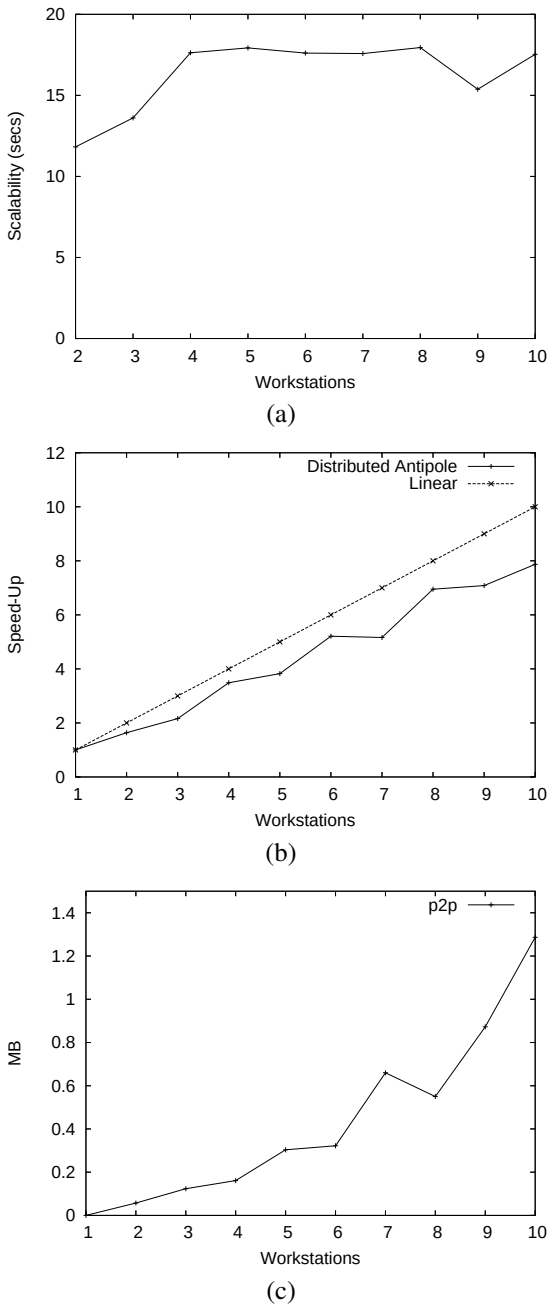


Figure 9. Scalability, speed-up and network traffic on 10^5 biosequences.

Acknowledgment

Support by MIUR Grant PRIN 2003 is acknowledged. Part of this work was developed for the project "PErvasive COmputing Supervisory Systems" (PECOSS) of Proteo S.p.A, supported by MIUR Grant N.11255 .

References

[1] P. Agarwal, J. Matousek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1:189–201, 1991.

[2] C. Aggarwal, J. L. Wolf, P. S. Yu, and M. Epelman. Using unbalanced trees for indexing multidimensional objects. *Knowledge and Information Systems*, 1(3):157–192, 1999.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *Proceedings of ACM SIGMOD*, pages 94–105, 1998.

[4] K. M. Alzoubi, P. J. Wan, and O. Frieder. Distributed heuristics for connected dominating sets in wireless ad hoc networks. *Journal of Communications and Networks*, 4(1), 2002.

[5] J. E. Kriegel H.-P. Pfeifle M. Towards effective and efficient distributed clustering. *Proc. Int. Workshop on Clustering Large Data Sets, 3rd Int. Conf. on Data Mining (ICDM 2003)*, pages 49–58, 2003.

[6] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proceedings of Combinatorial Pattern Matching, 5th Annual Symposium*, volume 807 of *Lecture Notes in Computer Sciences*, pages 198–212. Springer-Verlag, 1994.

[7] G. Barequet and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 82–91, 1999.

[8] S. Basagni. Distributed clustering for ad hoc networks. *Proc. Int'l Symp. Parallel Architectures, Algorithms, and Networks*, 1999.

[9] S. Battiato, D. Cantone, D. Catalano, G. Cincotti, and M. Hofri. An efficient algorithm for the approximate median selection problem. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, volume 1767 of *Lecture Notes in Computer Sciences*, pages 226–238. Springer-Verlag, 2000.

[10] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

[11] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. *Proceedings of the 22Nd International Conference on Very Large Databases*, pages 28–39, 1996.

[12] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is Nearest Neighbor meaningful? *Proceedings of the 7Th Intl. Conference on Database Theory*, 1540:217–235, 1999.

[13] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transaction on Database Systems*, 24(3):361–404, 1999.

[14] S. Brin. Near neighbor search in large metric spaces. *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 574–584, 1995.

[15] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[16] I. Calantari and G. McDonald. A data structure and an algorithm for the nearest point problem. *IEEE Transaction on Software Engineering*, 9(5):631–634, 1983.

[17] D. Cantone, G. Cincotti, A. Ferro, and A. Pulvirenti. An efficient approximate algorithm for the 1-median problem in metric spaces. *SIAM Journal on Optimization*, 16(2):434–451, 2005.

[18] D. Cantone, A. Ferro, R. Giugno, G. L. Presti, and A. Pulvirenti. Multiple-winners randomized tournaments with consensus for optimization problems in generic metric spaces. *Prof. of 4th International Workshop on Efficient and Experimental Algorithms*, pages 265–276, 2005.

[19] D. Cantone, A. Ferro, A. Pulvirenti, D. R. Recupero, and D. Shasha. Antipole tree indexing to support range search and k-nearest neighbor search in metric spaces. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):535–550, 2005.

[20] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12(1-2):67–85, 2002.

[21] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. *Proceedings of SPIRE*, pages 75–86, 2000.

[22] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[23] P. Ciaccia and M. Patella. Bulk loading the M-tree. *Proceedings of the 9Th Australasian Database Conference (ADC)*, pages 15–26, 1998.

[24] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Proceedings of the 23th International Conference on Very Large Data Bases*, pages 426–435, 1997.

[25] K. Clarkson. Nearest neighbor queries in metric spaces. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 609–617, May 1997.

[26] I. S. Dhillon and D. S. Modh. A data-clustering algorithm on distributed memory multiprocessors. *Proc. Int. Conf. on Knowledge Discovery and Data Mining, SIGKDD*, 1999.

[27] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[28] P. B. U. Fayyad and C. Reina. Scaling clustering algorithms to large databases. *Proc. of KDD*, 1998.

[29] T. Feder and D. Greene. Optimal algorithms for approximate clustering. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 434–444, 1988.

[30] A. Ferro, G. Pigola, A. Pulvirenti, and D. Shasha. Fast clustering and minimum weight matching algorithms for very large mobile backbone wireless networks. *Int. J. Found. Comput. Sci.*, 14(2):223–236, 2003.

[31] A. W.-C. Fu, P. M. Chan, Y.-L. Cheung, and Y. Moon. Dynamic VP-tree indexing for n-nearest neighbor search given pair-wise distances. *The VLDB Journal*, 9(2):154–173, 2000.

[32] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. *Proceedings of the IEEE 15th International Conference on Data Engineering*, pages 502–511, 1999.

[33] M. Gerla and J. T. C. Tsai. Multicluster, mobile, multimedia radio network. *ACM-Baltzer Journal of Wireless Networks*, 1(3):255–265, 1995.

[34] J. Ghosh and S. Merugu. Distributed clustering with limited knowledge sharing. *Proc. of the 5th International Conference on Advances in Pattern Recognition*, pages 48–53, 2003.

[35] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[36] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[37] T. Gonzalez. Covering a set of points in multidimensional space. *Information Processing Letters*, 40:181–188, 1991.

[38] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. *Proceedings of ACM SIGMOD*, pages 73–84, 1998.

[39] A. Guttman. R-trees: a dynamic index structure for spatial searching. *Proceedings of ACM SIGMOD*, pages 47–57, 1984.

[40] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, 2001.

[41] S. Har-Peled. A practical approach for computing the diameter of a point set. *Proceedings of the 17th Symposium on Computational Geometry*, pages 177–186, 2001.

[42] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[43] E. Hoel and H. Samet. Data-parallel spatial join algorithms. *Proc. of the 23rd Intl. Conf. on Parallel Processing*, 3:227–234, August 1994.

[44] P. Indyk. Sublinear time algorithms for metric space problems. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 428–434, 1999.

[45] E. Januzaj, H. P. Kriegel, and M. Pfeifle. Scalable density-based distributed clustering. *Proc. 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa*, pages 231–244, 2004.

[46] P. Jouve and N. Nicoloyannis. A new method for combining partitions, applications for distributed clustering. *Proc. in International Workshop on Paralell and Distributed Machine Learning and Data Mining*, 2003.

[47] I. Kamel and C.Faloutsos. Parallel r-trees. *Proceedings of the ACM SIGMOD*, pages 195–204, 1992.

[48] M. Klusch, S. Lodi, and G. Moro. Distributed clustering based on sampling local density estimates. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 485–490, 2003.

[49] N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. *EDBT '96: Proceedings of the 5th International Conference on Extending Database Technology*, pages 592–614, 1996.

[50] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[51] A. B. McDonald and T. Znati. A mobility based framework for adaptive clustering in wireless ad-hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8), 1999.

[52] J. McQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Symposium on Math, Statistics, and Probability*, pages 281–297, 1967.

[53] A. Mondal, M. Kitsuregawa, B. C. Ooi, and K. Tan. R-tree-based data migration and self-tuning strategies in shared-nothing spatial database. *Proc. of ACM GIS*, pages 28–33, 2001.

[54] L. Mutenda and M. Kitsuregawa. Parallel r-tree spatial join for a shared-nothing architecture. *Proc. of DANTE*, pages 423–430, 1999.

[55] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11:28–46, 2002.

[56] R. Ng and J. Han. Clarans: a method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[57] H. Noltemeier, K. Verbarg, and C. Zirkelbach. Monotonous bisector* trees - a tool for efficient partitioning of complex scenes of geometric objects. In *Data Structure and Efficient Algorithms*, volume 594 of *Lecture Notes in Computer Sciences*, pages 186–203. Springer-Verlag, 1992.

[58] C. F. Olson. Parallel algorithms for hierarchical clustering. *Parrallel Computing*, 21(8):1313 – 1325, 1995.

[59] C. D. Pietro, A. Ferro, G. Pigola, A. Pulvirenti, M. Purrello, M. Ragusa, and D. Shasha. Anticlustal: Multiple sequence alignment by antipole clustering. *Data Mining in Bioinformatics*, pages 43–57, 2005.

[60] C. D. Pietro, V. D. Pietro, G. Emmanuele, A. Ferro, T. Maugeri, E. Modica, G. Pigola, A. Pulvirenti, M. Purrello, M. Ragusa, M. Scalia, D. Shasha, S. Travali, and V. Zimmitti. Anticlustal: Multiple sequence alignment by antipole clustering and linear approximate 1-median computation. *2nd IEEE Computer Society Bioinformatics Conference (CSB 2003), 11-14 August 2003, Stanford, CA, USA*, pages 326–336, 2003.

[61] C. Procopiuc. *Geometric Techniques for Clustering Theory and Practice*. PhD thesis, Duke University, 2001.

[62] B. Schnitzer and S. Leutenegger. Master-client r-trees: A new parallel r-tree architecture. *Proc. of the 23rd Intl. Conf. on Parallel Processing*, pages 68–77, 1999.

[63] M. Shapiro. The choice of reference points in best-match file searching. *commACM*, 20(5):339–343, 1997.

[64] D. Shasha and T.-L. Wang. New techniques for best-match retrieval. *ACM Transaction on Information Systems*, 8(2):140–158, 1990.

[65] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A wavelet based clustering approach for spatial data in very large databases. *The VLDB Journal*, 8(3-4):289–304, 2000.

[66] S. Sumanasekara and M. Ramakrishna. CHILMA: An efficient high dimensional indexing structure for image databases. *Proceedings of the First IEEE Pacific-Rim Conference on Multimedia*, pages 76–79, 2000.

[67] C. Traina Jr, A. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. *Proceedings of the 7th International Conference on Extending Database Technology*, 1777:51–65, 2000.

[68] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[69] J. Vaidya and C. Clifton. Privacy-preserving k-means clustering over vertically partitioned data. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 206–215, 2003.

[70] P. Wendel, M. Ghanem, and Y. Guo. Scalable clustering on the data grid. *Fourth All Hands Meeting*, 2005.

[71] X. Xu, J. Jger, and H.-P. Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3:263–290, 1999.

[72] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, January 1993.

[73] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 103–114, 1996.