

# C<sup>2</sup>P: Clustering based on Closest Pairs

Alexandros Nanopoulos

Dept. Informatics  
Aristotle Univ.  
Thessaloniki, Greece  
alex@delab.csd.auth.gr

Yannis Theodoridis

Data & Knowl. Eng. Group  
Computer Technology Inst.  
Patras, Greece  
ytheod@cti.gr

Yannis Manolopoulos\*

Dept. Computer Science  
Univ. of Cyprus  
Nicosia, Cyprus  
manolopo@ucy.ac.cy

## Abstract

In this paper we present C<sup>2</sup>P, a new clustering algorithm for large spatial databases, which exploits spatial access methods for the determination of closest pairs. Several extensions are presented for scalable clustering in large databases that contain clusters of various shapes and outliers. Due to its characteristics, the proposed algorithm attains the advantages of hierarchical clustering and graph-theoretic algorithms providing both efficiency and quality of clustering result. The superiority of C<sup>2</sup>P is verified both with analytical and experimental results.

## 1 Introduction

Clustering is the organization of a collection of data into groups with respect to a distance or, equivalently, a similarity measure. Its objective is to assign to the same cluster data that are more close (similar) to each other than they are to data of different clusters. In data mining, clustering is used for the discovery of the distribution of data and the detection of patterns, e.g., finding groups of customers with similar buying behavior.

The application of clustering in spatial databases presents important characteristics. Spatial databases

---

\* On sabbatical leave from the Department of Informatics, Aristotle University, Thessaloniki, Greece, 54006.

*Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.*

usually contain very large numbers of points. Thus, algorithms for clustering in spatial databases do not assume that the entire database can be held in main memory. Therefore, additionally to the good quality of clustering, their scalability to the size of the database is of the same importance.

### 1.1 Related work

CLARANS [NH94] is a partitional clustering algorithm which, however, requires multiple database scans. The efficiency of CLARANS is improved by using focusing techniques proposed in [EKX95]. Range queries and the R\*-tree index structure are used to focus only on related portions of the database. DBSCAN [EKX96] is a density-based clustering algorithm for spatial databases. DBSCAN is based on parameters that are difficult to determine, whereas DBCLASS [XEKS98] does not present this requirement (with the cost of a reduction in clustering performance). Since both algorithms may follow a dense “bridge” of points that connects different clusters, they are sensitive to the “chaining-effect”, i.e., the incorrect merging of clusters (left part of Figure 1). In [BBBK00] the exploitation of the similarity join queries, using the R\*-tree and the X-tree index structures, is proposed to overcome the necessity for repetitive range queries required by DBSCAN. The STING algorithm [WYM97] is based on a grid-like data structure and the experimental results in [WYM97] indicate that STING outperforms DBSCAN.

The previous algorithms operate directly on the entire database. On the other hand, hierarchical clustering algorithms for data mining in spatial databases follow different approaches, which are based on data preprocessing. Algorithm BIRCH [ZRL96] is based on a pre-clustering scheme with the use of a specialized spatial index structure, the CF-tree, to determine a set of sub-clusters that are much fewer than the original data points. For the main clustering phase it resorts to a standard hierarchical algorithm for the set of the centers of the sub-clusters. In [GRS98], the

CURE algorithm is based on sampling and partitioning as an initial phase of data preprocessing. For the main clustering phase it uses a variation of standard hierarchical algorithms, which is facilitated by the k-d-tree [Same90] and the nearest-neighbor query. As described in [GRS98], CURE outperforms existing hierarchical algorithms, including BIRCH, both with respect to quality of result and efficiency.

OPTICS [ABKS99] is based on the concepts of density-based clustering (DBSCAN) and identifies the structure of clusters. Extensions to OPTICS are proposed in [BKS00] and, recently, in [BKKS01]. WaveCluster [SCZ99] uses the properties of the wavelet transformation (low pass filtering, multi-resolution examination). Clustering is done by merging components of the transformed space in the quantized grid. This process resembles density-based clustering algorithms. Hence, dense narrow bands of points between clusters, not consisting of just isolated points and thus not removable by the low pass filtering at the level where clustering is done, can render this procedure sensitive to the “chaining-effect”. Chameleon [KHK99] finds clusters with complicated shapes. It is based on graph partitioning, which presents significant computational requirements. Its scalability to large databases has not been examined in [KHK99]. Other related work includes algorithms in [AY00, HK99]. However, those approaches are not directly comparable with those discussed earlier, due to the peculiarities of high-dimensional space [HK99, AY00].

## 1.2 Contribution

In this paper, we present C<sup>2</sup>P, a new clustering algorithm for large spatial databases. C<sup>2</sup>P is based on the *Closest Pair* query (CPQ), presented in [CMTV00, HS98], and its variations [CMTV01]. The main objective of C<sup>2</sup>P is efficiency and scalability. We show that for large inputs its time complexity is  $O(n \log n)$  ( $n$  is the number of input points). Experimental results verify that C<sup>2</sup>P achieves both very good scalability and clustering quality. In summary, the main contributions of this paper are:

- A novel clustering algorithm, C<sup>2</sup>P, which combines efficiency and quality of clustering result.
- The recognition of the importance of *closest-pair* queries in clustering and the exploitation of the corresponding spatial access methods for clustering in spatial databases.

The rest of the paper is organized as follows. Section 2 gives necessary background information. The overview of the proposed method is described in Section 3, whereas the detailed algorithm is presented in Section 4. Section 5 contains the experimental results. Finally, conclusions are given in Section 6.

## 2 Background

Given  $n$  data points and  $k$  required clusters, hierarchical agglomerative clustering algorithms start with  $n$  clusters and iteratively merge the closest pair of clusters until  $k$  clusters are remaining. Several different measures for the distance between two clusters  $C_i$  and  $C_j$  have been proposed, e.g.:

$$\begin{aligned} D_{min}(C_i, C_j) &= \min_{\forall x \in C_i, x' \in C_j} \|x - x'\| \\ D_{mean}(C_i, C_j) &= \|m_i - m_j\|, m_x \text{ center of } C_x \end{aligned}$$

$D_{min}$  is used by the single-link hierarchical clustering algorithm, and joins the two clusters which contain the closest pair of points. It can detect elongated or concentric clusters, but it is sensitive to the “chaining-effect”. Each cluster is represented by all its points.  $D_{mean}$  is appropriate for detecting spherical and compact clusters. Each cluster is represented only by its center point. In [MSD83] different schemes are proposed for cluster representation, where each cluster is represented by a subset of distant points within the cluster. CURE [GRS98] follows this approach and uses the  $D_{min}$  distance for the set of representative points. Moreover, CURE uses a heuristic of shrinking the representative points by a factor (called shrinking factor) towards the center of the cluster to dampen the effect of outliers.

In [Epps98] it is shown that a hierarchical clustering algorithm can have  $O(n^2 \log^2 n)$  time complexity and  $O(n)$  space complexity, independently of the cluster distance function. In [GRS98], CURE (which uses the  $D_{min}$  cluster distance function) is shown to have  $O(n^2 \log n)$  time complexity and  $O(n)$  space complexity (under the assumption of low dimensionality, the time complexity is reduced to  $O(n^2)$ ). Hence, hierarchical algorithms are not directly applicable for large spatial databases.

Sampling can be used to reduce the size of large databases [GRS98, PF00, J-HS98, KGKB01]. However, if clusters are not of uniform size and are not well separated, small samples may lead to incorrect result [GRS98, BKS00], as illustrated in the example at the right part of Figure 1.

To overcome this problem large samples have to be used, which possibly cannot be stored entirely in main memory. Two general approaches are proposed for this purpose [JMF99]: a) Incremental Clustering, which has been used in [Fish87] and [ZRL96] (BIRCH), but it is affected by the ordering of points. b) Divide-and-conquer Clustering, which is based on partitioning and has been used in [MK80] and [GRS98] (CURE). Quadratic time complexity is still required for each partition and for the final clustering [GRS98].

Clustering has also been posed as a graph-theoretic problem [AM70, RY81]. Based on the *Minimal Spanning Tree* (MST), clusters can be produced as its components by removing the longest edges. This procedure is followed in [Zahn71]. For spatial data, the

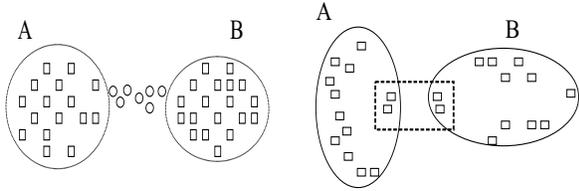


Figure 1: **Left:** “Chaining-effect”: Clusters  $A$  and  $B$  are connected with a “bridge” of points (circle-shaped). **Right:** Due to few samples in each cluster, the distance of points inside the rectangle (dash-line) is less than the distance from the other points of their clusters  $A$  and  $B$ , respectively. Thus, points across clusters are incorrectly merged.

required time complexity is determined by the Euclidean MST construction (the equivalent of MST for vector spaces), which is done in  $O(n \log n)$  time [PS85]. Therefore, graph-theoretic algorithms can be used for scalable clustering<sup>1</sup>. However, since all graph-theoretic algorithms that are based on MST are analogous to hierarchical clustering with the  $D_{min}$  distance [Epps98], they are sensitive to outliers and to the “chaining-effect”.

### 3 Clustering based on closest pairs

In this section we present the basic features of  $C^2P$ , the proposed clustering algorithm. As it follows from the description of the previous section, graph-theoretic (based on MST) and hierarchical clustering algorithms follow two different approaches, respectively:

1. *Static:* Cluster distances and representations do not change during the clustering procedure. The MST is computed once and the distances of the clusters, represented by the edges of the MST, are not updated. This is efficient with respect to time complexity ( $O(n \log n)$ ) but impacts the clustering quality.
2. *Fully dynamic:* Cluster distances and representations are updated after each merging of a pair of clusters. This is effective with respect to the clustering quality but the drawback is the high time complexity ( $O(n^2)$ ).

$C^2P$  consists of two main phases. The first phase efficiently determines a number of sub-clusters. Unlike hierarchical clustering algorithms, clusters, cluster distances and representations are not updated repeatedly after the merging of each pair of clusters. Unlike graph-theoretic algorithm, the representations are not static. The second phase performs the final clustering by using the sub-clusters of the first phase and a different cluster representation scheme. As a result,  $C^2P$

<sup>1</sup>Actually, the clusters of the single-link algorithm are sub-graphs of the MST [Epps98].

combines the advantages of both previous approaches, i.e., the low time complexity and the quality of clustering result. Since  $C^2P$  is based on the determination of closest pairs, in the following we first describe the corresponding algorithms on this subject and then we present in more detail the clustering procedure.

#### 3.1 Closest-pair queries

The *Closest-Pair* query (CPQ), which is proposed in [CMTV00, HS98], finds the closest pair of points from two datasets indexed with two R-tree data structures. In [CMTV01], two specializations of CPQ are proposed. The first is the *Self Closest-Pair* query (Self-CPQ), which finds the closest pair of points in a single dataset. The second is the *Self-Semi Closest-Pair* query (Self-Semi-CPQ), which, for one dataset, finds for each point its nearest neighbor point (equivalent to the all-nearest-neighbor query). In [CMTV01] several algorithms are presented for the Self-CPQ and the Self-Semi-CPQ. Here we follow the *Simple Recursive* versions of the two corresponding algorithms, assuming that the points are indexed with one R-tree and that the distance measure is the Euclidean.

Given  $n$  points, the execution time for Self-CPQ or Self-Semi-CPQ is the sum of time for creating the R-tree index and the time required by the corresponding *Simple Recursive* algorithm. The creation of the R-tree can be done with the bulk-loading algorithm of [KF93] in  $O(n \log n)$  time (due to the sorting of points). The expected search time for the *nearest-neighbor* query in an R-tree is  $O(\log n)$ , since the height of the R-tree is  $O(\log n)$  and the query searches a limited number of paths (a similar reasoning is followed in [ABKS99] for the cost of the determination of  $\epsilon$ -neighborhood). A naive algorithm for the execution of the aforementioned CP-queries can be based on the application of the nearest-neighbor query for each of the  $n$  points. The time complexity in this case is  $O(n \log n)$ . However, the execution time of the corresponding Simple Recursive algorithms in [CMTV01] require less time, compared to the naive algorithm, due to the branch-and-bound search procedure [CMTV01]. Hence, an upper bound for the overall time complexity of these queries is  $O(n \log n)$ . The space complexity in these cases (stemming from the R-tree) is linear to the number of points. It has to be noticed that different spatial index structures, e.g., the k-d-tree, can be used equivalently, maintaining the above bounds. We focus on the R-tree family, more particularly, on the R\*-tree structure, mainly because of its popularity (R-trees are found in commercial database systems, such as Oracle and Informix).

#### 3.2 Overview of clustering procedure

##### 3.2.1 First phase

The first phase of  $C^2P$  has as input  $n$  points and produces  $m$  sub-clusters, and it is iterative. Initially,

Self-Semi-CPQ finds  $n$  pairs of points  $(p, p')$  such that  $dist(p, p') = \min_{x \neq p} \{dist(p, x)\}$ , for each  $p$ . The pairs form a graph, where each point corresponds to a vertex and the weight of an edge  $(p, p')$  is equal to the distance between  $p$  and  $p'$ . We store the graph with the *adjacency lists* representation, that maintains for each vertex  $p'$  all the vertices  $p$  that are adjacent to it. More than one vertices may be adjacent to a vertex  $p'$ , since  $p'$  may be the closest point of more than one points. Similarly, there may exist parallel edges, called cycles of length two, between a pair of vertices, or cycles of larger length (due to ties)<sup>2</sup>. Consequently, the graph may have less than  $n - 1$  distinct edges. A graph with fewer than  $n - 1$  distinct edges cannot be connected, thus consisting of  $c > 1$  connected components.

The  $c$  connected components of the graph can be found with a Depth-First Search in the graph. Each component comprises a sub-cluster. In case  $c$  is equal to  $m$  (required number of sub-clusters), the first phase terminates. Otherwise, it proceeds to the next iteration. It finds the center of each sub-cluster, that forms its representation. Then, the same procedure is iteratively applied to the set of the  $c$  center points.

The connected components of the graph maintain the proximity of points. The Minimal Spanning Tree (graph-theoretic algorithm) also connects points which are close, but it is not allowed to contain cycles, hence close points may not be connected if they form a cycle. For C<sup>2</sup>P, since the points of each component maintain proximity, they are merged in a single step and not in multiple, as in the case of hierarchical algorithms, where only two clusters are merged at each step. Due to the representation of the clusters with their centers, C<sup>2</sup>P uses both the  $D_{min}$  (within components) and  $D_{mean}$  (for centers) cluster distance measures.

### 3.2.2 Second phase

The first phase of C<sup>2</sup>P has the objective of efficiently producing a number of sub-clusters which capture the shape of final clusters. Therefore, it represents clusters with their center points. The second phase uses a different cluster representation scheme to produce the finer final clustering. Also, the second phase merges two clusters (not multiple as the first) at each step in order to better control the clustering procedure. Therefore, the second phase is a specialization of the first, i.e., the latter can be modified in: a) finding different points to represent the cluster instead the center point, b) finding at each iteration only the closest pair of clusters that will be merged, instead of finding for each cluster the one closest to it.

When two clusters are merged, the  $r$  points (among all their points) which are closest to the center are selected (modification a.) and used as representatives of the new cluster. This way, variations in the shape

<sup>2</sup>Since the graph has  $n$  edges, it always contains a cycle of length larger than or equal to two.

and size of the clusters are captured more effectively [GRS98]<sup>3</sup>. The finding of the closest pair of clusters is done with Self-CPQ (modification b.). The original Self-CPQ is modified slightly to find pairs belonging to different clusters. The second phase terminates when the required number of clusters is reached and the representative points are found for each cluster (the final assignment of original points to clusters is described in the next section). From the above description it follows that the second phase operates in a manner analogous to hierarchical agglomerative clustering.

An example of the two-phase procedure of C<sup>2</sup>P is illustrated in Figure 2.

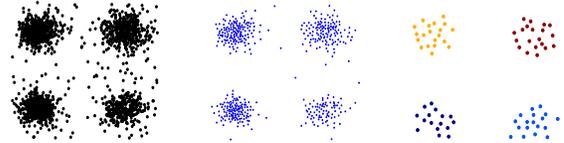


Figure 2: **Left:** An example dataset (4 clusters) **Center:** First phase. The set of center points of the sub-clusters. **Right:** Second phase. The set of representatives for each cluster (depicted with different colours).

## 4 The C<sup>2</sup>P clustering algorithm

First, the basic version of C<sup>2</sup>P is presented, consisting of the algorithms for the two phases. The first phase is implemented with the C<sup>2</sup>P-C algorithm (denoting the representation with the center point) and the second with the C<sup>2</sup>P-R algorithm (denoting the use of  $r$  representative points). In the sequel, extensions are given for the handling of outliers and for clustering very large spatial databases.

### 4.1 C<sup>2</sup>P-C

Figure 3 illustrates the C<sup>2</sup>P-C algorithm, based on the description of the previous section.

```

Algorithm C2P-C(D, m) {
1) c = |D|, P=D
2) while(c > m) {
3)   CP = Self-Semi-CPQ(P)
4)   G = AdjListsGraph (CP)
5)   P = ConnectedComponents_DFS(G, m)
6)   c = |P|
7) }
8) return P
}

```

Figure 3: Basic algorithm: first phase

The inputs to C<sup>2</sup>P-C are the database of points  $D$  and the number of required sub-clusters  $m$ . Variable  $c$  denotes the current number of clusters (initially set to

<sup>3</sup>Representation with scattered points that are shrink towards the center can be easily used as in [GRS98], but it requires the specification of the shrinking factor.

$|D|$ ).  $P$  denotes the current set of points to be cluster (initially set to  $D$ ).  $C^2P-C$  iterates until the current number of clusters,  $c$ , is equal to  $m$ . At each iteration, Self-Semi-CPQ finds the set of all closest pairs, denoted as  $CP$  (step 3). The AdjListsGraph procedure forms the graph  $G$  from the pairs  $CP$  (step 4). ConnectedComponents\_DFS finds the connected components of  $G$  (step 5). It also finds for each component its center point. The set of all center points is returned in variable  $P$  (the points to be clustered next). Variable  $c$  is updated to the current number of clusters (step 6). At the last iteration, if less than  $m$  clusters are going to be produced (within step 5), then edges with the largest weights are removed and split the corresponding components, until exactly  $m$  clusters are produced. The result of  $C^2P-C$  is contained in variable  $P$ , consisting of the centers of the  $m$  sub-clusters.

**Proposition 1** *For  $n$  points,  $C^2P-C$  has  $O(n \log n)$  time complexity and  $O(n)$  space complexity.*

**Proof.** For  $n$  points, step 3 has  $O(n \log n)$  time complexity (Section 3.1). The formation of the adjacency lists at step 4 has  $O(n)$  time complexity, since each point is appended at the end of the corresponding list and there are  $n$  edges. The DFS traversal and the determination of the connected components at step 5 has time complexity  $O(n)$  for graphs that are represented with adjacency lists [Sedg83]. Also, the centers of the clusters are found in  $O(n)$  time. It follows that the time complexity is dominated by step 3.

At the best case, the first iteration produces  $m$  clusters and  $C^2P-C$  terminates. Thus, in this case, the time complexity of the algorithm is  $O(n \log n)$ . At the worst case, the maximum number of clusters than can occur after step 5 for the  $n$  points is  $\frac{n}{2}$ . This happens when only parallel edges are formed. In this case, the following iteration has to cluster  $\frac{n}{2}$  points and thus, the time complexity  $C_n$  for the  $n$  points can be expressed as  $C_n = O(n \log n) + C_{n/2}$ . This recursive equation has solution  $C_n = O(n \log n)$  [SF96]. Thus, the overall time complexity of  $C^2P-C$  is  $O(n \log n)$ .

As mentioned in Section 3.1, the space complexity of step 3 is  $O(n)$ . The adjacency lists of the graph at step 4 have also  $O(n)$  time complexity, because the graph has  $n$  edges. After each iteration, only the cluster centers are maintained. Therefore, the space complexity of  $C^2P-C$  is  $O(n)$ .  $\square$

## 4.2 $C^2P-R$

Figure 4 illustrates  $C^2P-R$ . The inputs are:  $SC$ , the set of sub-clusters produced by  $C^2P-C$  ( $|SC| = m$ ), and  $k$ , the required number of clusters.  $P$  denotes the points that are clustered at each step (initially equal to  $SC$ ). Variable  $c$  denotes the number of clusters (initially is set to  $m$ ).  $C^2P-R$  finds the clusters,  $C_1, C_2$  that contain the closest pair of representative points

(step 3).  $C_1$  and  $C_2$  are merged with procedure Merge and the set  $RP$  of the representative points of the new cluster is found (the old ones are removed step 5). The current number of clusters is reduced by one (step 7).

```

Algorithm C2P-R(SC, k) {
1) c = |SC|, P=SC
2) while(c > k) {
3)   Self-CPQ(P, &C1, &C2)
4)   RP = Merge(C1, C2, r)
5)   P -= representatives of C1, C2
6)   P += RP
7)   c--;
8) }
9) return P
}

Procedure Merge(C1, C2, r) {
1) R1 = representatives of C1
2) R2 = representatives of C2
3) m = Center(R1, R2)
4) RP = r-NearestNeighbor(m)
5) return RP
}

```

Figure 4: Basic algorithm: second phase

$C^2P-R$  operates in a manner analogous to hierarchical clustering algorithms, therefore, its time complexity is equivalent to theirs.

**Corollary 1** *Let  $m$  be the number of center points of the sub-clusters produced by the first phase.  $C^2P-R$  has  $O(m^2 \log m)$  time complexity.*

**Proof:** The Self-CPQ at each step of  $C^2P-R$  requires time  $O(m \log m)$ . This determines the time complexity of the step (since only the representatives are stored for each cluster, the complexity of the Merge procedure is constant, i.e., it does not depend on  $m$ ). Each step is executed at most  $m - 1$  times. It follows that the overall time complexity is  $O(m^2 \log m)$ .  $\square$

The number,  $m$ , of sub-clusters produced by the first phase determines the “switching” point between the two phases. It has to be considered that at the second phase each of the  $k$  clusters will be represented with  $r$  points at most. Therefore,  $m$  can be set to  $r \cdot k$  (i.e., it does not depend on the number,  $n$ , of input points of the first phase). The value of  $r$  depends on the shape of clusters and can be specified following the approaches in [GRS98, MSD83].

From Proposition 1 and Corollary 1, it follows that the overall time complexity of  $C^2P$  is  $O(n \log n + m^2 \log m)$ . For large sample sizes, since  $m \ll n$  and  $m$  does not depend on  $n$ , it follows that the complexity of  $C^2P$  is determined from the  $O(n \log n)$  complexity of the first phase. Hence,  $C^2P$  is scalable to inputs of large sizes.

## 4.3 Elimination of outliers

Outliers are called the points that do not belong to any particular cluster. They tend to belong to isolated por-

tions of the space, thus forming small neighborhoods of points that are far apart from existing clusters.

The basic version of  $C^2P$  is extended by following a simple heuristic for the elimination of outliers, which is applied during the first phase. At step 3 of  $C^2P-C$ , the mean distance  $\bar{D}$  of points to their closest point is found. Then, in step 5, points with distance larger than a cut-off distance are considered as outliers and are removed. The cut-off distance is determined as a multiple of  $\bar{D}$  (e.g.,  $3 \cdot \bar{D}$ ). Additionally, if a component that is found at step 5 of  $C^2P-C$  has a very small number of points (e.g., 2), then it is removed. However,  $C^2P-C$  does not apply the latter heuristic during its early iterations because, initially, points tend to form small clusters.

#### 4.4 Clustering large databases

$C^2P$  uses sampling for the reduction of the databases size (Section 2). The results of [J-HS98] indicate that: a) with an increase in the database size, the clustering quality is preserved for constant sample size, b) an increase in random sample size does not pay off beyond a point, i.e. the quality in the result does not increase, whereas the execution time is increased significantly. In [GRS98] it is proven that the size of the random sample does not depend on the size of the database. This way, as verified in [GRS98], the quality of result is comparable to the case of applying clustering on the complete database, whereas the efficiency is significantly improved (sampling may also reduce the impact of outliers). Similar to [GRS98], and for fair comparison (c.f., Section 5),  $C^2P$  uses the reservoir random sampling algorithm [Vitt85]. However, [PF00, KGKB01] present algorithms for density-biased sampling for the case of skewed distribution in cluster sizes. We point out the examination of density-biased sampling as a direction of future work. For the determination of random sample sizes we refer to [GRS98, J-HS98]. It should be mentioned that when  $C^2P$  is applied to a sample, the time complexity of  $O(n \log n)$  refers to the size of sample,  $n$ . Regarding the total number of points,  $N$ , in the database, reservoir sampling examines only a small fraction of them [Vitt85].

##### 4.4.1 Divide-and-conquer method

For large random samples that cannot fit entirely in main memory (Section 2) we next present a divide-and-conquer procedure,  $C^2P-P$  (denoting partitions), as an extension to  $C^2P-C$ .  $C^2P-P$  (illustrated in Figure 5) divides the sample into a number of partitions and clusters them separately. In a second phase, the final clustering is produced. The contents of each partition can be stored entirely in main memory.

The inputs of  $C^2P-P$  are the set  $S$  of points to be clustered (i.e., the sample), the required number of sub-clusters  $m$  and the number of partitions  $p$ .  $C$  de-

```

 $C^2P-P(S, m, p)$  {
1) create partitions  $P_i$  of  $S$ ,  $i$  in  $[1, p]$ 
2)  $C = \{\}$  /*the empty set*/
3) foreach  $P_i$  {
4)    $C += C^2P-C(P_i, m')$ 
5) }
6)  $C^2P-C(C, m)$ 
}

```

Figure 5: Divide-and-conquer algorithm

notes the set of partial clusters (initially is empty). For each partition  $P_i$ ,  $C^2P-C$  is applied (step 4) for a required number of clusters  $m' > m$ . The result consists of the center points of the partial clusters found in partition, and it is added to set  $C$ . Then (step 6),  $C^2P-C$  is applied to the points of  $C$  ( $|C| = p \cdot m'$ ) and  $m$  clusters are produced. It is assumed that outlier elimination is applied only during the execution of  $C^2P-C$  (steps 4 and 6). The value of  $m'$  is set so as: a) wrong cluster merging will not occur during partial clustering, b) the  $|C|$  points can be held in main memory. Section 5 will elaborate further on these issues.

##### 4.4.2 Labeling the database contents

After having found the clusters in a given random sample,  $C^2P$  has to determine the cluster that each point of the initial database belongs to. Points are assigned to the closest cluster. The distance of a point  $p$  from a cluster  $C$  is equal to the minimum distance of  $p$  from the representative points of  $C$ . As it is illustrated in Figure 4,  $C^2P-R$  returns the representative points of the discovered clusters. These points can be organized in an R-tree data structure to facilitate the searching of the nearest representative point.

## 5 Performance results

This section contains the experimental results on the performance of  $C^2P$ . We examine both the quality of the clustering result and the efficiency. We compare  $C^2P$  with CURE and graph-theoretic algorithm. Based on the experimental results of [GRS98], CURE is an efficient hierarchical agglomerative clustering algorithm, scalable to large databases and, at the same time, outperforms existing algorithms (e.g., BIRCH) with respect to the clustering quality. As used in [GRS98], we derive the MST based graph-theoretic algorithm from CURE, by setting as representatives all the points in a cluster and shrinking factor to 0.

$C^2P$  and CURE were developed in  $C$ , using the same components. For CURE we used the improved procedure for merging clusters, described in [GRS98]. Since we are interested in the relative comparison of the algorithms, we used a main-memory R-tree data structure for both algorithms and the method in [KF93] for its bulk-loading. In [GRS98], points are stored in a k-d-tree. However, the clustering result is

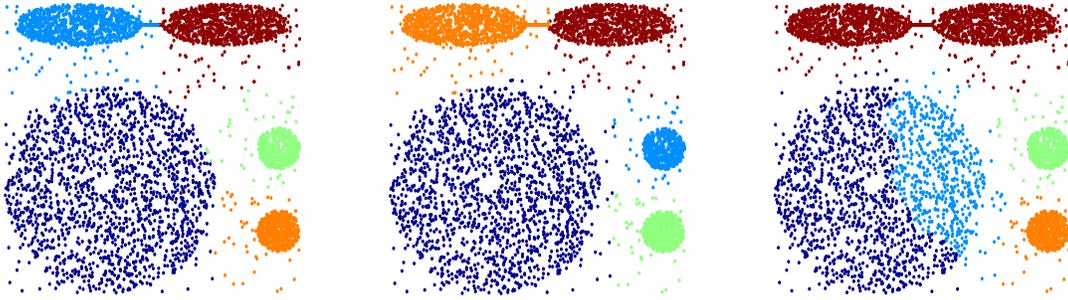


Figure 6: DS1: **Left:** Result of  $C^2P$ . **Center:** Result of CURE. **Right:** Result of graph-theoretic algorithm

not affected by the type of the data structure (see Section 3.1) and from our results it was realized that the time performance of R-tree is not inferior to that of k-d-tree. The experiments were run using a Pentium II computer at 500 MHz with 256 MB of RAM under Windows NT.

The datasets that were used (due to space restrictions we present only two characteristic ones), are denoted as DS1 (100,000 two dimensional points in 5 clusters – see Figure 6) and DS2 (100,000 points in 100 clusters placed on a grid – see Figure 7). DS1 was used in [GRS98]. Its characteristics can test the quality of clustering result with respect to the “chaining-effect”, the outliers and clusters with different sizes and shapes. DS2 was used both in [ZRL96, GRS98]. Next, we present the results regarding the three algorithms on the clustering quality and in the sequel we present the results on efficiency. The depicted images were produced in color (available in [NTM01]).

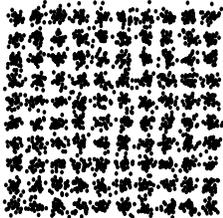


Figure 7: DS2: 100 spherical clusters placed on a grid.

### 5.1 Clustering Quality

First, we examined the quality of the result for DS1. The random sample contained 2,500 points (the same size that was used in [GRS98]) and no partitioning was applied. We used the default values for the parameters of CURE as given in [GRS98], i.e., 10 representative points per cluster and 0.3 shrinking factor. For  $C^2P$ , the first phase produced 100 sub-clusters (i.e.,  $m = 100$ ) and 20 points were used as representatives (i.e.,  $r = 20$ ). The outlier cut-off distance was set to three times the average (set as default value). Figure 6 presents the clustering results. As it is illustrated, both  $C^2P$  and CURE can correctly discover the clusters in DS1 (the different coloring is due to

the different order that clusters were identified). This shows that  $C^2P$  is not affected by the “chaining-effect”, the differences in the cluster sizes and the existence of outliers. On the other hand, the graph theoretic algorithm is affected by the “chaining-effect” and merges the two elongated clusters. Also, it splits the large spherical cluster. Analogous results were obtained for DS2, and are presented in [NTM01].

### 5.2 Tuning the parameters of $C^2P$

The main parameters of  $C^2P$  are the number,  $m$ , of the sub-clusters of the first phase (the “switching” point) and the number,  $r$ , of representatives of the second phase. As described, the size of the random sample can be determined following the approaches in [GRS98, J-HS98], thus we do not perform a separate examination here. Partitioning is examined in the following section. We use the DS1 data set that contains 5 clusters (i.e.,  $k = 5$ ) and the size of the random samples was set to 5,000 points.

#### Number of representatives $r$

We set  $m$  to 100 and varied the number of representatives,  $r$ , in the range from 1 (equivalent to the case where only the center is used for representation) to 30. For values less than 20,  $C^2P$  did not always produce the correct result (especially for very small values of  $r$ ). The left part of Figure 8 illustrates the representatives per cluster when  $r$  was set to 5. As it is shown, the large cluster, in the lower left corner, is split. More precisely, the representative points were assigned to different clusters, thus the cluster, after the labeling phase, is finally split (the final result is not shown here). Additionally, the two small clusters on the right were merged, since the corresponding representatives were assigned to the same cluster. For larger values of  $r$ , the correct result is produced. The right part of Figure 8 illustrates the result when  $r$  was set to 20. As it is shown, the representative points were correctly assigned to clusters and the result after labeling (not shown here) was correct. Generally, the value of  $r$  is determined by the shape of the clusters,

with larger clusters requiring more representatives to capture their shape.

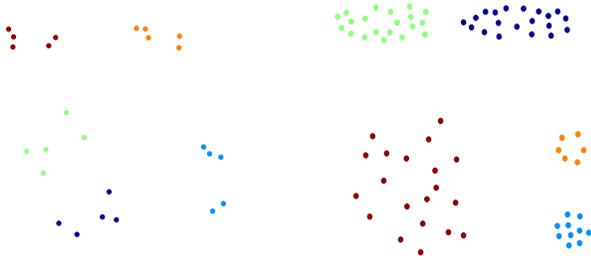


Figure 8: Number,  $r$ , of the representatives in the second phase. **Left:**  $r = 5$ . **Right:**  $r = 20$ .

### Number of sub-clusters $m$

Based on the previously described results, we selected  $r$  equal to 20. Since  $k$  is 5, from the description of Section 4.1.2, the number of sub-clusters of the first phase has to be set to 100 (i.e.,  $20 \cdot 5$ ). We varied the values of  $m$  in the range from 5 to 200. The left part of Figure 9 illustrates the result for  $m = 10$ , where it can be noticed that an incorrect clustering is produced (the large cluster was split and the two small clusters, on the right, were merged). Similar results were obtained for values less than 100. For larger values, C<sup>2</sup>P produced the correct result. The right part of Figure 9 illustrates the result for  $m = 100$ . Thus, although  $m$  is much smaller than the size,  $n$ , of the sample (in this case  $n = 5,000$ ), the sub-clusters of the first phase can effectively capture the shape of the final clusters.

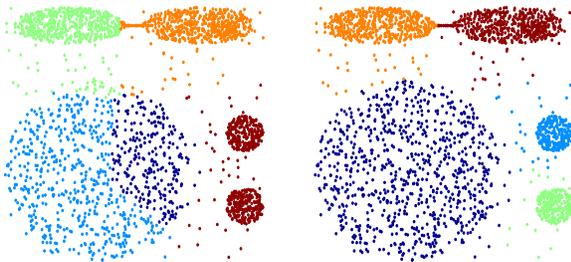


Figure 9: Number,  $m$ , of the sub-clusters of the first phase. **Left:**  $m = 10$ . **Right:**  $m = 100$ .

### 5.3 Efficiency

First, we measured the execution time with respect to the sample size. We did not consider the graph-theoretic algorithm for the measurement of efficiency since, as presented previously, it did not produce the correct clustering result. The result for DS1 is depicted in Figure 10. CURE used the default parameters given in [GRS98] (10 representatives, 0.3 shrinking factor). For C<sup>2</sup>P,  $m$  was set to 100 and  $r$  to 20. No partitioning

was used. The result for DS2 is depicted in Figure 11, where for both algorithms one representative per cluster is used [GRS98]. As it is illustrated, C<sup>2</sup>P clearly outperforms CURE both for small and large sample sizes. Also, these results verify the analytical time complexity described in Section 4 and indicate that C<sup>2</sup>P is scalable to inputs with large number of points, in the case of data sets that require large samples (see Section 2).

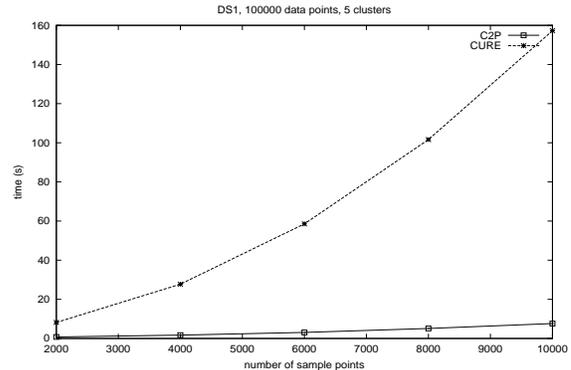


Figure 10: DS1: Exec. time (in seconds) w.r.t. sample size.

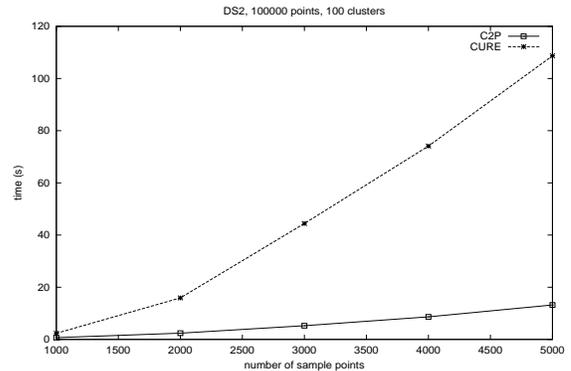


Figure 11: DS2: Exec. time (in seconds) w.r.t. sample size.

Next we examined the impact of the divide-and-conquer technique on the performance of both algorithms. More precisely, we applied C<sup>2</sup>P-P and the partitioning algorithm described in [GRS98]. We used DS1, with C<sup>2</sup>P and CURE having the same parameter setting as in the previous experiment. For both algorithms, the clustering in each partition stopped when the number of sub-clusters was 1/3 of the number of points in the partition (this fraction specifies parameter  $m'$  of C<sup>2</sup>P-P). This criterion is indicated in [GRS98] and we use it here for the sake of fair comparison. The left part of Figure 12 presents the execution times with respect to the sample size for five partitions. The right part of Figure 12 presents the execution times with respect to the number of partitions,

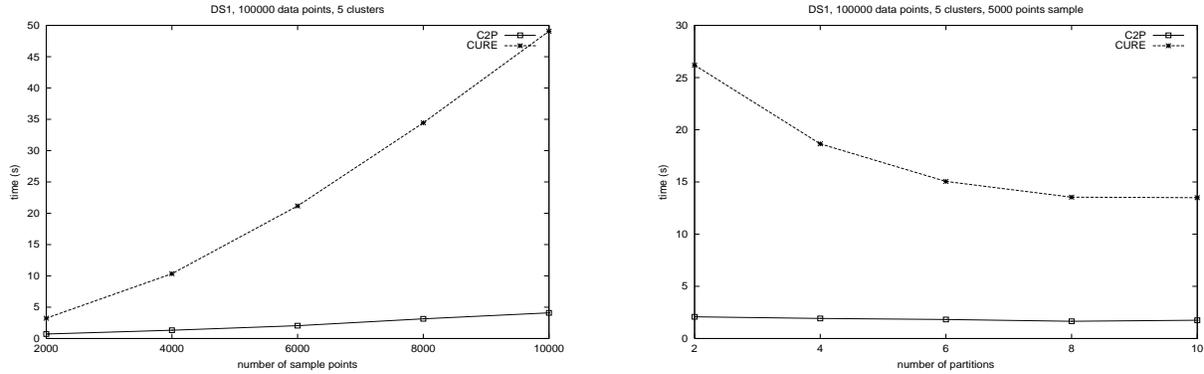


Figure 12: **Left:** Execution time (in seconds) w.r.t. sample size, for 5 partitions. **Right** Execution time (in seconds) w.r.t. number of partitions.

for a sample size of 5,000 points. As it is illustrated, both algorithms present a performance improvement, compared to the previous case where the divide-and-conquer method was not used. However, CURE is clearly outperformed by C<sup>2</sup>P, both for small and large numbers of partitions.

## 6 Conclusions

We considered the problem of scalable clustering in large databases, which at the same time maintains the quality of the result.

We have proposed a new clustering algorithm, C<sup>2</sup>P, which exploits index structures and the processing of closest pair queries in spatial databases. It combines the advantages of the hierarchical agglomerative and graph-theoretic clustering algorithms. Extensions are provided for large spatial databases and for outlier handling. We showed that the time complexity of C<sup>2</sup>P for large datasets is  $O(n \log n)$ , thus it scales well to large inputs. Experimental results indicate its efficiency. The main findings of this paper are summarized as follows:

- The development of a new clustering algorithm, which is scalable to large input sizes and maintains the quality of the clustering result.
- Extensions to the basic algorithm for handling outliers, clusters of various shapes and sizes, and large databases.
- Analytical and experimental results, which illustrate the superiority of C<sup>2</sup>P.

In future work will consider the density-based sampling [PF00, KGKB01] for cluster sizes that follow skew distribution.

## Acknowledgments

We would like to thank Mr. Antonio Corral for his help on the subject of the closest pair queries.

## References

- [ABKS99] M. Ankerst, M. Breunig, H-P. Kriegel, J. Sander: “OPTICS: Ordering Points To Identify the Clustering Structure”. *Proc. ACM Int. Conf. on Management of Data (SIGMOD’99)*, 1999, pp. 49-60.
- [AM70] J. Augustson, J. Minker: “An Analysis of some Graph Theoretical Clustering Techniques”. *Journal of the ACM*, Vol.17, 4, 571-588, 1970.
- [AY00] C. Aggarwal, P. Yu: “Finding Generalized Projected Clusters In High Dimensional Spaces”. *Proc. ACM Int. Conf. on Management of Data (SIGMOD’00)*, 2000, pp. 70-81.
- [BBBK00] C. Böhm, B. Braunmüller, M. Breunig, H-P. Kriegel: “High Performance Clustering Based on the Similarity Join”. *Proc. Int. Conf. on Information and Knowledge Management (CIKM’00)*, 2000.
- [BKKS01] M. Breunig, H.-P. Kriegel, P. Kroger, J. Sander: “Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering”. *Proc. ACM Int. Conf. on Management of Data (SIGMOD’01)*, 2001.
- [BKS00] M. Breunig, H.-P. Kriegel, J. Sander: “Fast Hierarchical Clustering Based on Compressed Data and OPTICS”. *Proc. European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD’00)*, 2000.
- [CMTV00] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos: “Closest Pair Queries in Spatial Databases”. *Proc. ACM Int. Conf. on Management of Data (SIGMOD’00)*, 2000, pp. 189-200.
- [CMTV01] A. Corral, Y. Manolopoulos, Y. Theodoridis, M. Vassilakopoulos: “Algorithms for Processing Closest Pair Queries in Spatial Databases”. Tech. Report (available at [www.de.csd.auth.gr/publications.html](http://www.de.csd.auth.gr/publications.html)), 2001.
- [Epps98] D. Eppstein: “Fast Hierarchical Clustering and Other Applications of Dynamic Closest

- Pairs". *Proc. Symposium on Discrete Algorithms (SODA'98)*, 1998.
- [EKSX96] M. Ester, H.-P. Kriegel, J. Sander, X. Xu: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". *Proc. Int. Conf. on Knowledge Discovery and Data Mining (KDD'96)*, 1996.
- [EKX95] M. Ester, H.-P. Kriegel, X. Xu: "Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification", *Proc. Int. Symp. on Large Spatial Databases (SSD'95)*, 1995, pp. 67-82.
- [Fish87] D. Fisher: "Knowledge Acquisition via Incremental Conceptual Clustering". *Machine Learning*, Vol.2, 139-172, 1987.
- [GRS98] S. Guha, R. Rastogi, K. Shim: "CURE: An Efficient Clustering Algorithm for Large Databases". *Proc. ACM Int. Conf. on Management of Data (SIGMOD'98)*, 1998, pp. 73-84.
- [HK99] A. Hinneburg, D. Keim: "Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering". *Proc. Int. Conf. on Very Large Databases (VLDB'99)*, 1999, pp. 506-517.
- [HS98] G. Hjaltason, H. Samet: "Incremental Distance Join Algorithms for Spatial Databases". *Proc. ACM Int. Conf. on Management of Data (SIGMOD'98)*, 1998, pp. 237-248.
- [J-HS98] N. Jose-Hkajavi, K. Salem: "Two-phase Clustering of Large Datasets". *Technical Report CS-98-27*, Department of Computer Science, University of Waterloo, 1998.
- [JMF99] A. Jain, M. Murty, P. Flynn: "Data Clustering: A Review". *ACM Computing Surveys*, Vol.31, No.3, 264-323, 1999.
- [KF93] I. Kamel, C. Faloutsos: "On Packing R-trees". *Proc. Int. Conf. on Information and Knowledge Management (CIKM'93)*, 1993, pp. 490-499.
- [KGKB01] G. Kollios, D. Gunopoulos, N. Koudas, S. Berchtold: "An Efficient Approximation Scheme for Data Mining Tasks". *Proc. IEEE Int. Conf. on Data Engineering (ICDE'01)*, 2001, pp. 453-462.
- [KHK99] G. Karypis, E. Han, V. Kumar: "Chameleon: Hierarchical Clustering Using Dynamic Modeling". *IEEE Computer*, Vol.32, No.8, 68-75, 1999.
- [MK80] M. Murty, G. Krishna: "A Computationally Efficient Technique for Data Clustering". *Pattern Recognition*, Vol. 12, 153-158, 1980.
- [MSD83] R. Michalski, R. Stepp, E. Diday: "A Recent Advance in Data Analysis: Clustering Objects into Classes Characterized by Conjunctive Concepts". *Progress in Pattern Recognition*, Vol. 1, North-Holland Publishing, 1983.
- [NH94] R. Ng, J. Han: "Efficient and Effective Clustering Methods for Spatial Data Mining". *Proc. Int. Conf. on Very Large Databases (VLDB'94)*, 1994, pp. 144-155.
- [NTM01] A. Nanopoulos, Y. Theodoridis, Y. Manolopoulos: "C<sup>2</sup>P: Clustering based on Closest Pairs". Tech. Report (available at [www.de.csd.auth.gr/publications.html](http://www.de.csd.auth.gr/publications.html)), 2001.
- [PF00] C. Palmer, C. Faloutsos: "Density Biased Sampling: An Improved Method for Data Mining and Clustering", *Proc. ACM Int. Conf. on Management of Data (SIGMOD'00)*, 2000, pp. 82-92.
- [PS85] F. Preparata, M. Shamos: "Computational Geometry: an Introduction". *Springer-Verlag*, 1985.
- [RY81] V. Raghavan, C. Yu: "A Comparison of the Stability Characteristics of some Graph Theoretic Clustering Methods". *IEEE Trans. on Pattern Analysis Machine Intelligence*, Vol.3, 393-402, 1981.
- [Same90] H. Samet: "*The Design and Analysis of Spatial Data Structures*". Addison Wesley, 1990.
- [Sedg83] R. Sedgewick: "*Algorithms*". Addison Wesley, 1983.
- [SCZ99] G. Sheikholeslami, S. Chatterjee, A. Zhang: "WaveCluster: A Wavelet Based Clustering Approach for Spatial Data in Very Large Databases". *VLDB Journal*, Vol. 8(3-4), 289-304, 1999.
- [SF96] R. Sedgewick, P. Flajolet: "*An Introduction to the Analysis of Algorithms*", Addison Wesley, 1996.
- [Vitt85] J. Vitter: "Random Sampling with a Reservoir". *ACM Trans. on Mathematical Software*, 11(1), 37-57, 1985.
- [WYM97] W. Wang, J. Yang, R. Muntz: "STING: A Statistical Information Grid Approach to Spatial Data Mining". *Proc. Int. Conf. on Very Large Databases (VLDB'97)*, 1997, pp. 186-195.
- [XEKS98] X. Xu, M. Ester, H.-P. Kriegel, J. Sander: "A Nonparametric Clustering Algorithm for Knowledge Discovery in Large Spatial Databases". *Proc. Int. Conf. on Data Engineering (ICDE'98)*, 1998.
- [Zahn71] C. Zahn: "Graph-theoretical Methods for Detecting and Describing Gestalt Clusters". *IEEE Trans. Computers C-20*, 1971, pp. 68-86.
- [ZRL96] T. Zhang, R. Ramakrishnan, M. Linvy: "BIRCH: An Efficient Data Clustering Method for Very Large Databases". *Proc. ACM Int. Conf. on Management of Data (SIGMOD'96)*, 1996, pp. 103-114.