Clustering Spatial Data Using Random Walks *

David Harel and Yehuda Koren †
Technical Report MCS01-08,
Dept. of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Rehovot, Israel.

Abstract

Discovering significant patterns that exist implicitly in huge spatial databases is an important computational task. A common approach to this problem is to use *cluster analysis*. However, traditional clustering methods have several shortcomings when addressing spatial data. We propose a novel approach to clustering spatial data, based on the deterministic analysis of random walks on a weighted graph generated from the data. Our approach has several advantages. First, it can decompose the data into arbitrarily shaped clusters of different sizes and densities. Second, it can overcome noise and outliers that may blur the natural decomposition of the data. Third, our method requires only $O(n \log n)$ time, and one of its variants needs only constant space.

1 Introduction

Spatial data are the data related to objects that occupy space. The data may be locations of objects in a geographical information system (GIS), medical databases, or vectors which describe parts in a CAD system [X$^+$97]. Advances in database technologies have resulted in huge amounts of spatial data, and knowledge discovery techniques become essential tools for successful analysis of spatial data bases. This paper deals with *clustering*, which is one of the central techniques in spatial data mining.

Clustering methods are used to discover natural groups in data sets, and to identify abstract structures that might reside there, without having any background knowledge of the characteristics of the data. They have been used in a variety of areas including: computer vision, VLSI design, data mining, web page clustering, and gene expression analysis.

Prior literature on the clustering problem is huge, see e.g., [JD88, JMF99]. However, to a large extent the problem remains elusive, and there is still a dire need for a clustering method that is natural and robust, yet very efficient in dealing with large data sets.

*A short version will appear in Proceedings of KDD-2001, The Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining

†E-mail: {harel,yehuda}@wisdom.weizmann.ac.il

The characteristics of spatial data pose several difficulties for clustering algorithms. Since we are seeking a natural decomposition, the clusters may have arbitrary shapes and non-uniform sizes. Moreover, different clusters may have different densities. Another issue is the existence of noise, which may interfere the clustering process and should be identified. Regarding complexity, the huge sizes of spatial databases imply the need for very efficient clustering algorithms (a quadratic running time may thus not be practical). Furthermore, it is impractical to assume that the entire database can reside in the main memory all at once.

In this paper, we present a new approach to clustering spatial data, based on deterministic exploration of random walks on a weighted graph associated with the data. The heart of the method is in what we shall be calling *separating operators*, which are applied to the graph iteratively. Their effect is to 'sharpen' the distinction between the weights of inter-cluster edges (those that ought to separate clusters) and intra-cluster edges (those that ought to remain inside a single cluster), by decreasing the former and increasing the latter. The operators can be used on their own or can be embedded in a classical agglomerative clustering framework.

The resulting algorithms are simple, fast and general, and seem to cope successfully with the unique difficulties common to spatial data. We exhibit encouraging results of applying these algorithms to several recently published data sets.

Section 2 describes some basic notions and Section 3 provides some background on the clustering problem. Section 4 describes how to model the data points using a weighted graph. Section 5, which is the heart of the paper, provides our basic random-walk-based separating operators and the resulting clustering algorithm. In section 6 we enhance our clustering method by combining it with classical agglomerative clustering. Section 7 demonstrates the results of our algorithm. In Section 8 we compare our work to some related clustering algorithms.

2 Basic notions

We will be using standard graph-theoretic notions. Specifically, let $G(V, E, w)$ be a weighted graph. The w is a weighting function $w : E \longrightarrow \mathbb{R}$, that measures the similarity between pairs of items (a higher value means more similar). Let $S \subseteq V$. The set of nodes that are connected to some node of S by a path with at most k edges is denoted by $V^k(S)$. The degree of G, denoted by $deg(G)$, is the maximal number of edges incident to some single node of G. The subgraph of G induced by S is denoted by $G(S)$. The edge between i and j is denoted by $\langle i, j\rangle$. Sometimes, when the context is clear, we will write simply $\langle i, j\rangle$ instead of $\langle i, j\rangle \in E$.

A *random walk* is a natural stochastic process on graphs. Given a graph and a start node, we select a neighbor of the node at random, and 'go there', after which we continue the random walk from the newly chosen node. The probability of a transition from node i to node j, is

$$p_{ij} = \frac{w(i,j)}{d_i}$$

where $d_i = \sum_{\langle i,k \rangle} w(i,k)$ is the *weighted degree* of node i.

Now, denote by $P^k_{visit}(i) \in \mathbb{R}^n$ the vector whose j-th component is the probability that a random walk originating at i will visit node j in its k-th step.

The *escape probability* from a source node s to a target node t, denoted by $P_{escape}(s,t)$, is defined as the probability that a random walk originating at s will reach t before returning to s. This probability can be computed as follows. For every $i \in V$, define a variable ρ_i satisfying:

$$\begin{aligned} &\rho_s = 0, \qquad \rho_t = 1, \quad \text{and} \\ &\rho_i = \sum_{\langle i,j \rangle} p_{ij} \cdot \rho_j \qquad \text{for } i \neq s,\ i \neq t \end{aligned}$$

The values of ρ_i are calculated by solving these equations, and then the desired escape probability is given by: $P_{escape}(s,t) = \sum_{\langle s,i \rangle} p_{si} \cdot \rho_i$

3 The Clustering Problem

The common definition of the clustering problem is as follows. Partition n given data points into k clusters, such that points within a cluster are more similar to each other than ones taken from different clusters.

The problem as described above is inherently ill-posed, since a set of points can be clustered naturally in many ways. For example, Figure 1(a) has three clusters, but one could argue that there are only two, since the two on the right hand side are close enough to be thought of as one. In Figure 1(b) one could argue for and against dividing the points in the top dense region into two highly connected natural clusters. A solution to such ambiguities is to use hierarchical clustering, which employs a parameter for controlling the desired resolution.

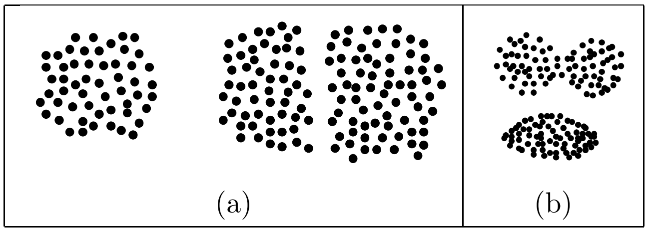


Figure 1: Inherent ambiguity in clustering: How many clusters are there here?

3.1 Clustering methods

We now survey some clustering approaches. We concentrate on distance based methods, which assume that there is a distance measurement between the data points, and instead of providing specific references to each method, we point the reader to [JD88, JMF99].

Clustering methods can be broadly classified into *hierarchical* and *partitional* approaches. Partitional clustering algorithms obtain a single partition of the data that optimizes a certain criterion. The most widely used criterion is minimizing the overall squared distance between each data point and the center of its related cluster. This tends to work well with isolated and compact clusters. The most common methods of this kind are the *k-Means* and the *k-Medoid* algorithms. An advantage of these algorithms is their robustness to outliers (nodes that cannot be classified into a natural cluster). Another advantage is their quick running time. Their major drawback is a tendency to produce spherically shaped clusters of similar sizes, which often prevents the finding of natural clusters.

Hierarchical algorithms create a sequence of partitions in which each partition is nested into next partition in the sequence, forming a pyramid of partitions called a *dendrogram*. *Agglomerative clustering* is a well-known hierarchical clustering method that starts from the trivial partition of n points into n clusters of size 1 and continues by repeatedly merging pairs of clusters. At each step the two clusters that are most similar are merged, until the clustering is satisfactory. Different similarity measures between clusters result in different agglomerative algorithms. The most widely used measures for similarity between clusters are as follows:

- *maximal similarity* ("single link clustering"): similarity between clusters is measured as the similarity between the most similar pair of elements, one from each of the clusters.
- *minimal similarity* ("complete link clustering"): same as maximal similarity, but using the least similar pair of elements.
- *average similarity* ("group average clustering"): same as maximal similarity, but using the average similarity of pairs of elements.
- *total similarity*: similarity is measured as the total sum of similarities over all pairs of elements, one from each of the clusters. This sum can be normalized by dividing it by some power of the clusters' sizes, to avoid discrimination against small clusters.
- *mean similarity*: similarity is measured as the similarity between the means (i.e., the centers) of the clusters. This measure can be applied only when the data has actual coordinates, so that the mean can be computed.

All the above measures usually yield the same results if the clusters are well-separated and compact. However, if the clusters are close to one another (even by outliers), or if their shapes and sizes are not hyperspherical and uniform, the results of clustering can vary quite dramatically. The measures of maximal similarity, average similarity and mean similarity tend to break up a relatively large (though natural) cluster into two (unnatural) clusters. On the other hand, an unnormalized version of the total similarity measure has a tendency to contract large clusters whose similarity is high due to their sizes. The max similarity measure also has a problem — the "chaining effect": it can be easily fooled by outliers, merging two clusters that are connected by a narrow string of points.

There have been several attempts to strengthen agglomerative algorithms. However, in [KHK99] one finds an excellent explanation of the limitations of existing hierarchical schemes when used in common spatial data.

The method described in this paper can actually be classified as a hierarchical algorithm.

4 Modeling the Data using a Weighted Graph

Our method, as many other hierarchical clustering algorithms, deals with a graph representation of the data.

In the simplest case, the graph is complete, i.e., the nodes represent the points of the data, and each node is connected to every other one. The main information lies within the edge weights, for which we adopt a commonly used approach (see e.g., [BWD97]): the weight of the edge $\langle a, b \rangle$ is $\exp(-\frac{d(a,b)^2}{ave^2})$, where $d(a, b)$ is the Euclidean distance between a and b, and ave is the average Euclidean distance between two adjacent points in the graph. However, many methods prefer to model the data using sparse graphs, which contain only a small subset of the edges of the complete graph, mostly those corresponding to higher similarity values. Working with sparse graphs has several advantages. First, it reduces the time and space complexity, and second, the "structure" of adequate sparse graphs reflects spatial proximity and expresses the arrangement of the data, thus helping the clustering process.

Several sparse graph structures have been used for modeling data points, see, e.g., [JD88]. We mention here two commonly used ones: the *Delaunay triangulation* and the *k-mutual neighborhood graph*.

The Delaunay triangulation (DT) of a point set is the dual of the famous *Voronoi diagram*, which is a partition of the space into cells, one for each data point, so that the cell for data point a consists of that region of the space that is closer to a than to any other data point. An edge in the DT connects points a and b if and only if the Voronoi cells containing a and b share a common boundary. Hence, edges of the DT capture spatial proximity. Notice that since the DT of a planar point set is a planar graph, it contains at most $3n - 6$ edges. Regarding computation efficiency, many $O(n \log n)$ time and $O(n)$ space algorithms exist for computing the DT of a planar point set.

The k-mutual neighborhood graph contains all edges $\langle a, b\rangle$ for which a is one of the k nearest neighbors of b, and b is one of the k nearest neighbors of a. Regarding computation efficiency, the k-mutual neighborhood graph can be computed directly from the DT in time $O(n)$ (here we take k as a constant), by finding the k nearest neighbors of each point through a search on the DT. In general, the k-nearest neighbors of each point can be found in time $O(n \log n)$ and $O(n)$ space for any fixed arbitrary dimension [DE96]. When working with spatial databases, the k-nearest neighbors can be computed alternatively, using *region queries* and *nearest neighbor queries*. These queries are implemented using sophisticated spatial index structures, which consume on average $O(\log n)$ time. More importantly, they are very economical with respect to memory, see e.g. [X$^+$97]. Using this query-based technique, the k-mutual neighborhood graph can be computed in average time of $O(n \log n)$ and using a negligible amount of memory.

Figure 2 demonstrates a DT and a k-mutual neighborhood graph of a point set, containing two natural clusters. One should observe that nodes at the of k-mutual neighborhood lying near the boundaries of the clusters are adjacent to relatively longer edges than those in the interior. This demonstrates a case where an irregular phenomenon in the structure of the graph does not reflect an important structural phenomenon in the regular point set, which is undesirable. In the DT such a phenomenon does not occur. On the other hand, the DT contains many more edges that connect disjoint clusters than the k-mutual neighborhood graph. In this sense, the k-mutual neighborhood graph better captures the natural decomposition of the data. Hence, in many cases it may be best to use the intersection of these two graphs, i.e., to include only those edges that appear in both the DT and the k-mutual neighborhood graph, as is done in Figure 2(d). This intersection can be computed from the two graphs in time $O(n)$. Notice that on spatial databases that support region queries and nearest neighbor queries, the intersection graph can be computed with a negligible amount of memory, as follows: First build the k-mutual neighborhood graph as stated above, and then decide for each edge $\langle a, b\rangle$ whether it is included in the DT or not, by considering only points that are closer to a than b or closer to b than a. Since a and b are k-mutual neighbors, we expect that very few points will be considered.

5 Cluster Analysis by Random Walks

5.1 Separators and separating operators

Our approach to identifying natural clusters in a graph is to find ways to compute an 'intimacy relation' between the nodes incident to each of the graph's edges. In other words, we want to be able to decide for each edge if it should cross the boundaries of two clusters, or, rather, if

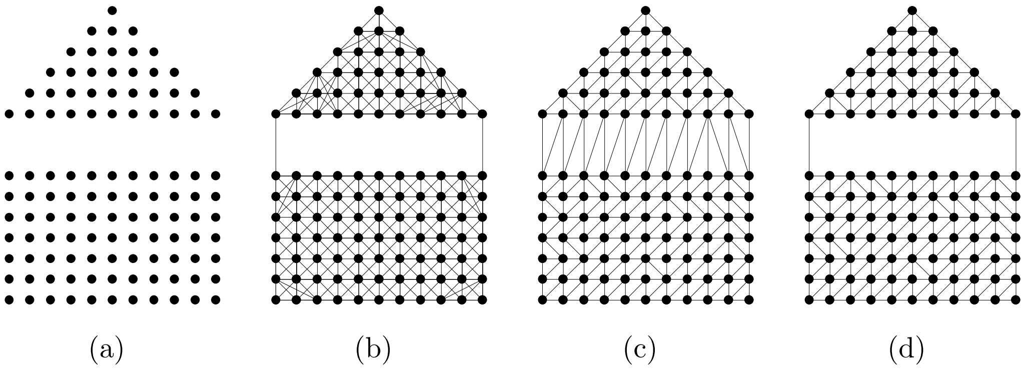


Figure 2: (a) the data set; (b) 10-mutual neighborhood graph; (c) Delaunay triangulation; (d) intersection of (b) and (c).

the relationship between its two incident nodes is sufficiently intimate for them to be contained in a common cluster.

Definition 5.1 *Let the graph $G(V, E)$ be clustered by $C = (C_1, \ldots, C_k)$. An edge $\langle u, v\rangle \in E$ is called a* separating edge for C, *or a* separator *for short, if $u \in C_i, v \in C_j$ for $i \neq j$.*

Any set of edges $F \subset E$ gives rise to an induced clustering C_F, obtained by simply taking the clusters to be the connected components of the graph $G(V, E - F)$. The set F will then contain precisely the separating edges of C_F. Another way of putting this is that if we can indeed decide which are the separators of a natural clustering of G, we are done, since we will simply take the clustering to be C_F for the discovered set F of separators.

We have decided to concentrate on discovering a set of separating edges, since the decision as to whether an edge should be separating involves only relatively local considerations. Globally speaking, there might not be much difference between two neighboring nodes, and the reasons for placing two neighbors in different clusters will most often be some sharp local transition of some characteristic of the cluster, which global considerations may decay, see e.g., Figure 3.

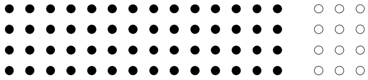


Figure 3: Two natural clusters. The only reason for separating the black points from the white (hollow) ones lies in local considerations. Most global approaches that seek for an intimate relation between every two points in the cluster, such as k-means, will probably fail here and might attach points from the right hand side of the black cluster to the white cluster.

The strategy we propose for identifying separators is to use an iterative process of *separation*. Separation reweights edges by local considerations in such a way that the weight of an edge connecting 'intimately related' nodes is increased, and for others it is decreased. This is a kind of *sharpening pass*, in which the edges are reweighted to sharpen the distinction between (eventual) separating and non-separating edges. When the separating operation is iterated several times, a sort of 'zero-one' phenomenon emerges, whereby the weight of an edge that should be a separator notably diminishes.

We now offer two methods for performing the edge separation, both based on deterministic analysis of random walks.

NS: Separation by neighborhood similarity

A helpful property of the vector $P^k_{visit}(i)$ is that it provides the level of nearness or intimacy between the node i and every other node, based on the structure of the graph. Actually, $P^k_{visit}(i)$ generalizes the concept of weighted neighborhoods, since $P^1_{visit}(i)$ is exactly the weighted neighborhood of i. Also $P^{\infty}_{visit}(i)$ does not depend on i and is equal to the *stationary distribution* of the graph. Hence, the value of $P^k_{visit}(i)$ is not very interesting for overly large values of k. We will actually be using the term $P^{\leq k}_{visit}(\cdot)$, which is defined to be $\sum_{i=1}^{k} P^i_{visit}(v)$.

Now, in order to estimate the closeness of two nodes v and u, we fix some small k (e.g., $k = 3$) and compare $P^{\leq k}_{visit}(v)$ and $P^{\leq k}_{visit}(u)$. The smaller the difference the greater the intimacy between u and v. The reason we use $P^{\leq k}_{visit}$ here and not P^k_{visit} is that for a bipartite subgraph the values of P^k_{visit} can be very different, since the two random walks originating from u and v cannot visit the same node at the same time. However, if we are willing to sum some steps of the two walks, we may find that they visit roughly the same nodes.

We now define the separating operator itself:

Definition 5.2 *Let $G(V, E, w)$ be a weighted graph and k be some small constant. The* separation of G by neighborhood similarity*, denoted by $NS(G)$, is defined to be:*

$$\begin{aligned} &NS(G) \stackrel{dfn}{=} G_s(V, E, w_s), \\ &\textit{where} \quad \forall \langle v, u \rangle \in E, \quad w_s(u, v) = sim^k(P^{\leq k}_{visit}(v), P^{\leq k}_{visit}(u)) \end{aligned}$$

Here, $sim^k(\vec{x}, \vec{y})$ is some similarity measure of the vectors $\vec{x}$ and $\vec{y}$, whose value increases as $\vec{x}$ and $\vec{y}$ become more similar. A suitable choice is:

$$f^k(\vec{x}, \vec{y}) \stackrel{\text{dfn}}{=} \exp(2k - \|\vec{x} - \vec{y}\|_{L_1}) - 1$$

The norm L_1 is defined in the standard way: For $\vec{a}, \vec{b} \in \mathbb{R}^n$, $\|\vec{a} - \vec{b}\|_{L_1} = \sum_{i=1}^{n} |a_i - b_i|$

Another suitable choice is the cosine, or the correlation, of $\vec{x}$ and $\vec{y}$, defined as:

$$cos(\vec{x}, \vec{y}) = \frac{(\vec{x}, \vec{y})}{\sqrt{(\vec{x}, \vec{x})} \cdot \sqrt{(\vec{y}, \vec{y})}}$$

where $(\cdot, \cdot)$ denotes the inner-product.

The key component in computing $NS(G)$ is the calculation of $P_{visit}^{\leq k}(v)$ and $P_{visit}^{\leq k}(u)$. Since the graphs that we want to cluster are of bounded degree, $P_{visit}^{\leq k}(u)$ can be computed in time and space $O(deg(G)^k)$, which is independent of the size of G and can be treated as a constant. Hence, $NS(G)$ can be computed in space $O(1)$ and time $\Theta(|E|)$, which in this case is just $\Theta(n)$.

CE: Separation by circular escape

An alternative method for capturing the extent of intimacy between nodes u and v, is by the probability that a random walk that starts at v visits u exactly once before returning to v for the first time. (This notion is symmetric, since the event obtained by exchanging the roles of v and u has the same probability.) If v and u are in different natural clusters, the probability of such an event will be low, since a random walk that visits v will likely return to v before reaching u (and the same with u and v exchanged).

The probability of this event is given by:

$$P_{escape}(v, u) \cdot P_{escape}(u, v)$$

Seeking efficient computation, and on the reasonable assumption that data relevant to the intimacy of v and u lies in a relatively small neighborhood around v and u, we can constrain our attention to a limited neighborhood, by the following:

Definition 5.3 *Let $G(V, E, w)$ be a graph, and let k be some constant. Denote by $P_{escape}^{(k)}(v, u)$ the probability $P_{escape}(v, u)$, but computed using random walks on the subgraph $G(V^k(\{v, u\}))$ instead of on the original graph G. The* circular escape probability *of v and u is defined to be:*

$$CE^k(v, u) \stackrel{dfn}{=} P_{escape}^{(k)}(v, u) \cdot P_{escape}^{(k)}(u, v).$$

We can now define separation by circular escape:

Definition 5.4 *Let $G(V, E, w)$ be a weighted graph, and let k be some small constant. The* separation of G by circular escape*, denoted by $CE(G)$, is defined to be:*

$$\begin{aligned} &CE(G) \stackrel{dfn}{=} G_s(V, E, w_s) \\ &\textit{where} \quad \forall \langle v, u \rangle \in E, \quad w_s(u, v) = CE^k(v, u) \end{aligned}$$

For graphs with bounded degree, the size of $G(V^k(v, u))$ is independent of the size of G, so that $CE^k(v, u)$ can be computed essentially in constant time and space. Hence, as with $NS(G)$, the separating operator $CE(G)$ can be computed in time $\Theta(|E|) = \Theta(n)$ and space $O(1)$.

Our experiments show that in general the CE operator yields better results than the NS operator. However, the computation of the CE operator is clearly more complicated in terms of numerical precision, as well as running time.

5.2 Clustering by separation

The idea of separating operators is to uncover and bring to the surface a closeness between nodes that exists implicitly in the structure of the graph. Separating operators increase the weights of intra-cluster edges and decrease those of inter-cluster ones. Iterating the separating operators sharpens the distinction further. After a small number of iterations we expect the difference between the weights of the two kinds of edges to differ sufficiently to be readily apparent, because the weights of separators are expected to diminish significantly.

We demonstrate this method by two simple graph-based examples. Consider Figure 4, which contains an almost uniformly weighted graph, taken from [SN99]. We experimented with both separating operators, each one with a four-fold iteration. The NS operator was used with $k = 3$ and $sim^k(\vec{x}, \vec{y}) \stackrel{\text{dfn}}{=} f^k(\vec{x}, \vec{y})$ and the CE operator with $k = 2$. The results of both runs appear along the edges in the figure (with those of CE appearing, multiplied by 100, in parentheses). As can be seen, the separation iterations cause the weights of edges $\langle 3, 18\rangle$, $\langle 7, 8\rangle$, $\langle 6, 10\rangle$, $\langle 1, 4\rangle$, and $\langle 8, 18\rangle$ to become significantly smaller than those of the other edges; in fact, they tend to zero in a clear way. We conclude that these edges are separators, thus obtaining the natural clustering of the graph by removing them and taking each connected component of the resulting graph to be a cluster, as indicated in the lower right hand drawing of the figure.

Notice that the first activation of the separating operator already shows differences in the intimacy that later lead to the clustering, but the results are not quite sharp enough to make a clear identification of separators. Take edge $\langle 6, 10\rangle$, for example. We intentionally initialized it to be of weight 1.5 — higher than the other edges — and after the first separation its weight is still too high to be labeled a separator. It is still higher than that of the non-separating edge $\langle 10, 13\rangle$. However, the next few iterations of the separating operator cause its weight to decrease rapidly, sharpening the distinction, and its being a separator becomes obvious.

The success in separating nodes 6 and 10 is remarkable, and would not be possible by many clustering methods. This demonstrates how our separation operators integrate structural properties of the graph, and succeed in separating these nodes, despite of the fact that the edge joining them has the highest similarity value in the graph.

Figure 5 shows the algorithms applied to a tree, using three-fold separation. The results clearly establish edges $\langle 0, 6\rangle$ and $\langle 6, 11\rangle$ as separators. Notice that the clustering methods that rely on edge-connectivity (like [BWD97, GWM99]), will fail here, since the edge-connectivity between every two nodes of the tree is 1.

The main virtues of clustering by separation are:

1. Applying a separation operator to an edge in a bounded-degree graph takes constant time and space, resulting in very good time complexity for large bounded-degree graphs.

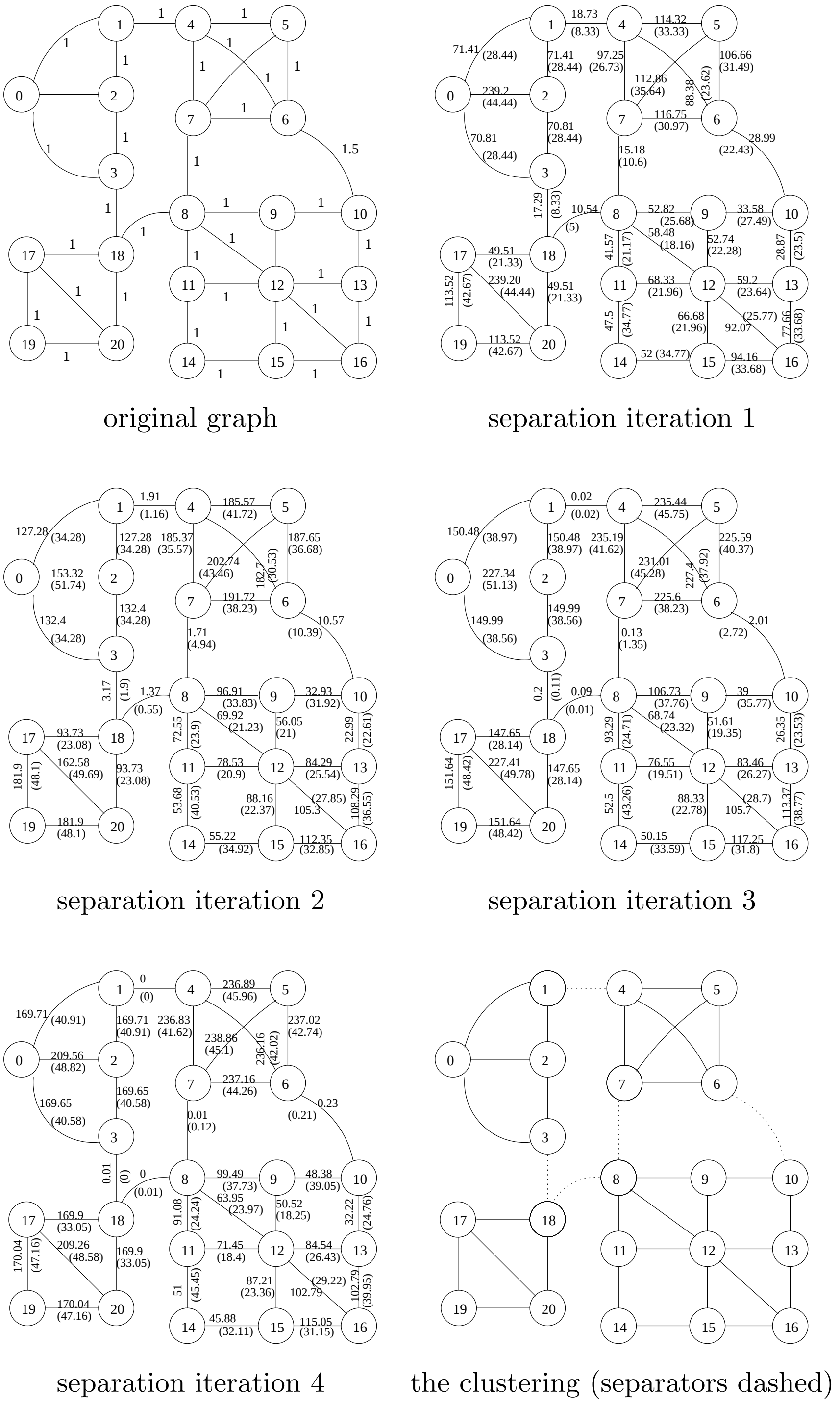


Figure 4: Clustering using four-fold application of separation operators, which sharpen the edge weight distinction (NS values are on top and CE values, multiplied by 100, are in parentheses); example taken from [SN99]

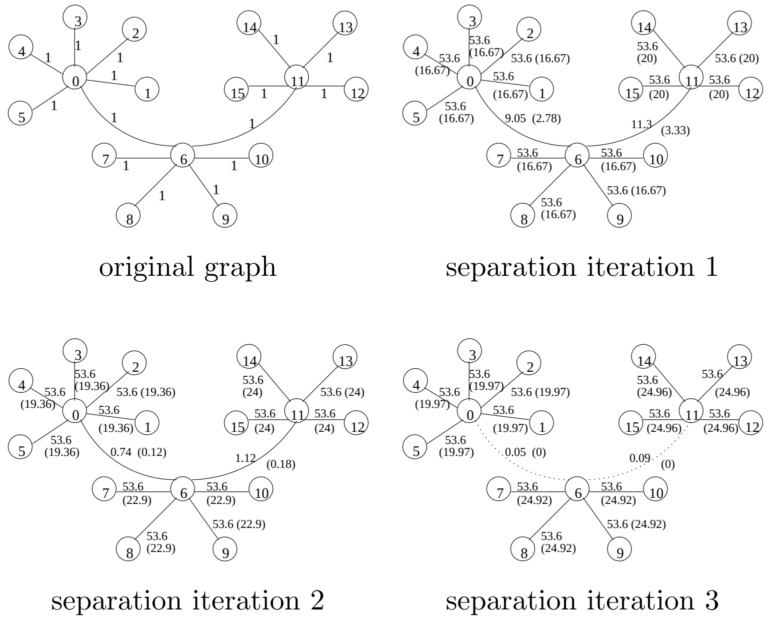


Figure 5: Clustering a tree using three-fold application of separation (separators are dashed in iteration 3)

2. Edges are weighted based on the relevant graph structure, thus overcoming phenomena like random noise and outliers, which are not reflected directly in the structure.

3. Iterating the separating operators causes information from distant parts of the graph to 'flow in', reaching the areas where separating decisions are to be made.

Notice that the differences between consecutive iterations of separation diminish as the process continues, and they appear to tend to a fixpoint. This behavior requires further investigation.

5.3 Clustering spatial points

We now illustrate the ability of our method to cluster "correctly" 2D points, in a number of typical cases, some of which have been shown to be problematic for agglomerative methods [KHK99]. (More extensive examples are given in Section 7.) We show only examples in 2D, although the method works well in higher dimensions too, because two dimensions are easier to visualize and evaluate.

We have used 10-mutual neighborhood graphs for modeling the points (intersection with the Delaunay triangulation gives similar results). The results are achieved using 3 iterations of either CE or NS, with $k = 3$. For NS, we took the function $sim(\cdot,\cdot)$ to be $f(\cdot,\cdot)$. In general, other choices work equally well.

The partition of the edges into separators and non-separators is based on a threshold value, such that all the edges whose weight is below this value are declared as separators. Without loss of generality, we may restrict ourselves to the $O(n)$ edge weights as candidates for being thresholds. The actual threshold value (or several, if a hierarchy of decompositions is called for), is found by some statistical test, e.g., inspecting the edge-weight frequency histogram, where the frequency of the separators' weights is usually smaller, since most of the edges are inside the clusters, and have higher weights than those of the separators.

Figure 6 shows the clustering decomposition of three data sets using our algorithm.

The data set DS1 shows the inherent capability of our algorithms to cluster at different resolutions at once, i.e., to detect several groups with different intra-group densities. This ability is beyond the capabilities of many clustering algorithms that can show the denser clusters only after breaking up the sparser clusters. Data set DS2 demonstrates the ability of our algorithm to separate the two left hand side clusters, despite the fact that the distance between these clusters is smaller than the distance between points inside the right hand side cluster.

The data set DS3 exhibits the capability of our algorithm to take into account the structural properties of the data set, which is the only clue for separating these evenly spaced points.

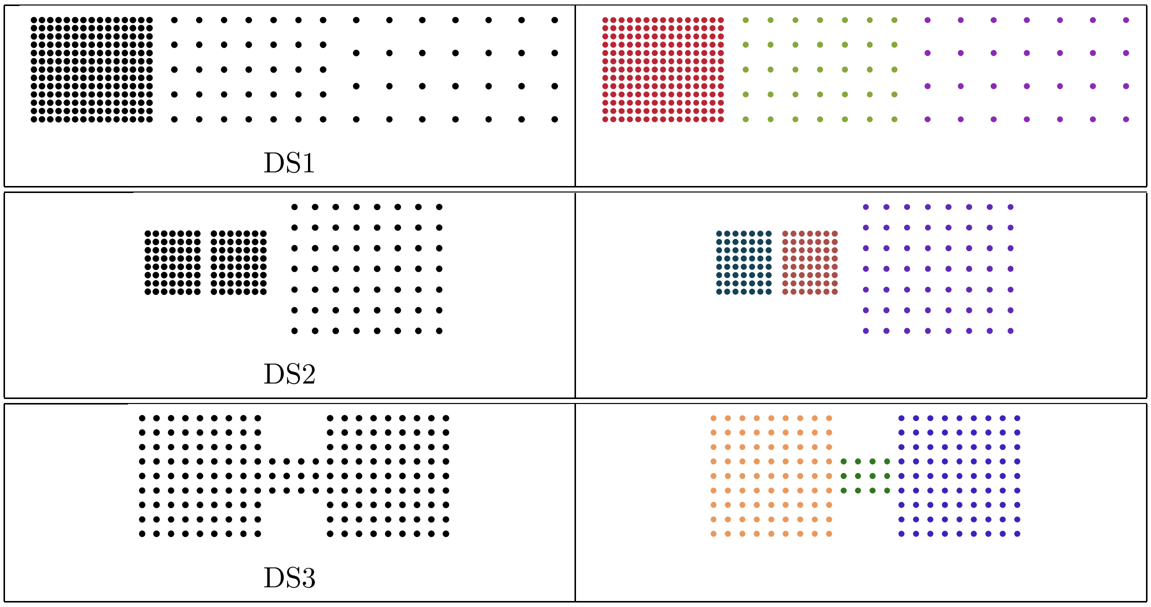


Figure 6: Clustering of several data sets. Different clusters are indicated by different colors.

When there is a hierarchy of suitable decompositions, our method can reveal it by using a different threshold for each level of the hierarchy. For example, consider the two data sets of Figure 1. For each of these we have used two different thresholds, to achieve two decompositions. The results are given in Figure 7.

It is noteworthy that the general methodology of revealing the graph structure by exploration of random walks is fundamental and robust enough to be applied to the clustering of spatial

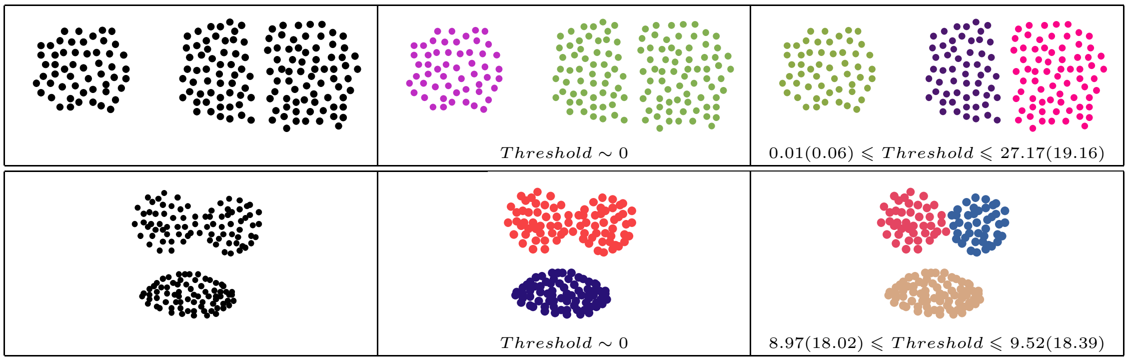


Figure 7: Clustering at multiple resolutions using different thresholds. When values of CE are different from values of NS, the CE values are given in parentheses. CE values are multiplied by 100.

data, even though our use of spatial properties of the data is minimum.

One should observe that the memory requirements of our clustering method are low, which is very important for huge spatial data bases that cannot fit in main memory. The only "global" operation in the process is that of computing connected components after removing separators. Since this operation does not require too many accesses to each single node, it can be performed quite efficiently without copying the entire data base into main memory.

6 Integration with Agglomerative Clustering

The separation operators can be used as a preprocessing stage before activating agglomerative clustering on the graph. (Without loss of generality, we think of agglomerative algorithms as working on graphs.) Such a preprocessing sharpens the edge weights, adding structural knowledge to them, and greatly enhances the agglomerative algorithms, as it can effectively prevent bad local merging opposing the graph structure.

Implementation of the agglomerative algorithm can be done using a dynamic graph structure. At each step we take the edge of the highest weight, merge ("contract") its two endpoints, and update all the adjacent edges. When contracting nodes u and v having a common neighbor t, the way we determine the weight of the edge between t and the contracted node uniquely distinguishes between different variants of the agglomerative procedure. For example, when using maximal similarity ("single link"), we take this weight as $\max\{w(v,t), w(u,t)\}$, while when using total similarity we fix the weight as $w(v,t) + w(u,t)$. For a bounded degree graph, which is our case, each such step can be carried out in time $O(\log n)$, using a binary heap.

It is interesting that the clustering method we have described in the previous section is in fact completely equivalent to a maximal similarity agglomerative ("single link") algorithm preceded by a separation operation. Hence we can view the integration of the separation operation with the agglomerative algorithm as a generalization of the method we have discussed in the previous section, that enables us to use any variant of the agglomerative algorithm.

We have found particularly effective the normalized total similarity variant, in which we measure the similarity between two clusters as the total sum of the weights of the original edges connecting these clusters. We would like to eliminate the tendency of such a procedure to contract pairs of nodes representing large clusters whose connectivity is high due to their sizes. Accordingly, we normalize the weights by dividing them by some power of the sizes of the relevant clusters. More precisely, we measure the similarity of two clusters C_1 and C_2 by:

$$\frac{w(C_1, C_2)}{\sqrt[d]{|C_1|} + \sqrt[d]{|C_2|}}$$

where $w(C_1, C_2)$ is the sum of original edge weights between C_1 and C_2, and d is the dimension of the space in which the points lie. We took $\sqrt[d]{|C_1|}$ and $\sqrt[d]{|C_2|}$ as an approximation of the size of the boundaries of the clusters C_1 and C_2, respectively.

The overall time complexity of the algorithm is $O(n \log n)$, which includes the time needed for constructing the graph and the time needed for performing n contractions using a binary heap. This equals the time complexity of the method described in the previous section (because of the graph construction stage). However, the space complexity is now worse. We need $\Theta(n)$ memory for efficiently handling the binary heap.

Selecting significant decompositions

An agglomerative clustering algorithm provides us with a dendrogram, which is a pyramid of nested clustering decompositions. The question of which are the meaningful decompositions inside the dendrogram, still remains.

Each level in the dendrogram is constructed from the level below, by merging two clusters. We associate with each level a grade that measures the importance of that level. Inspired by the work of [GWM99], a rather effective way of measuring the importance of a level is by evaluating how sharp is the change that this level introduces to the clustering decomposition. Since changes that are involved with small clusters do not have a large influence, we define the *prominency rank* of a level in the dendrogram, in which the clusters C_i and C_j of the level below were merged, as:

$$|C_i| \cdot |C_j|$$

We demonstrate the effectiveness of this measure in the next section.

7 Examples

In this section we show the results of running our algorithm on several data sets from the literature. We modeled these data sets using the intersection of the Delaunay triangulation and the 15-mutual neighborhood graph. For all the results we have used total similarity agglomerative clustering, preceded by 2 iterations of the NS separation operator with $k = 3$ and similarity function defined as $cos(\cdot, \cdot)$. Using the CE operator, changing the value of k, or increasing the number of iterations, do not have a significant effect on the results. Using the method described in Section 5 may change the results in few cases.

We implemented the algorithm in C++, running on a Pentium III 800MHz processor. The code for constructing the Delaunay triangulation is of Triangle, which is available from URL: http://www.cs.cmu.edu/˜quake/triangle.html.

Figure 8 shows the results of the algorithm on data sets taken from [KHK99]. These data sets contain clusters of different shapes, sizes and densities and also random noise. A nice property of our algorithm is that random noise gets to stay inside small clusters. After clustering the data, the algorithm treats all the relatively small clusters, whose sizes are below half of the average cluster size, as noise, and simply omits them showing only the larger clusters.

Figure 9 shows the result of the algorithm applied to a data set from [EL00]. We show two levels in the hierarchy, representing two possible decompositions. We are particularly happy with the algorithm's ability to break the cross shaped cluster into 4 highly connected clusters, as shown in Figure 9(c).

In Figure 10, which was produced by adding points to a data set given in [EL00], we show the noteworthy capability of our algorithm to identify clusters of different densities at the same level of the hierarchy. Notice that the intra-distance between the points inside the right hand side cluster, is larger than the inter-distance between several other clusters.

The data set in Figure 11, which in a way is the most difficult one we have included, is taken from [GWM99]. We have modeled the data exactly the same way described in [GWM99], by putting an edge between every two points whose distance is below some threshold. Using this model, [GWM99] shows the inability of two spectral methods and of the single-link algorithm to cluster this data set correctly.

Throughout all the examples given in this section we have used the prominency rank introduced in Section 6 to reveal the most meaningful levels in the dendrogram. Figure 12 demonstrates its capability with respect to the data set DS4 (shown in Figure 8). We have chosen the five levels with the highest prominency ranks, and for each level we show the level that precedes it. It can be seen that these five levels are exactly the five places where the six large natural clusters are merged. In this figure we have chosen not to hide the noise, so the reader can see the results of the algorithm before hiding the noise.

Table 1 gives the actual running times of the algorithm on the data sets given here. We should mention that our code is not optimized, and the running time can certainly be improved.

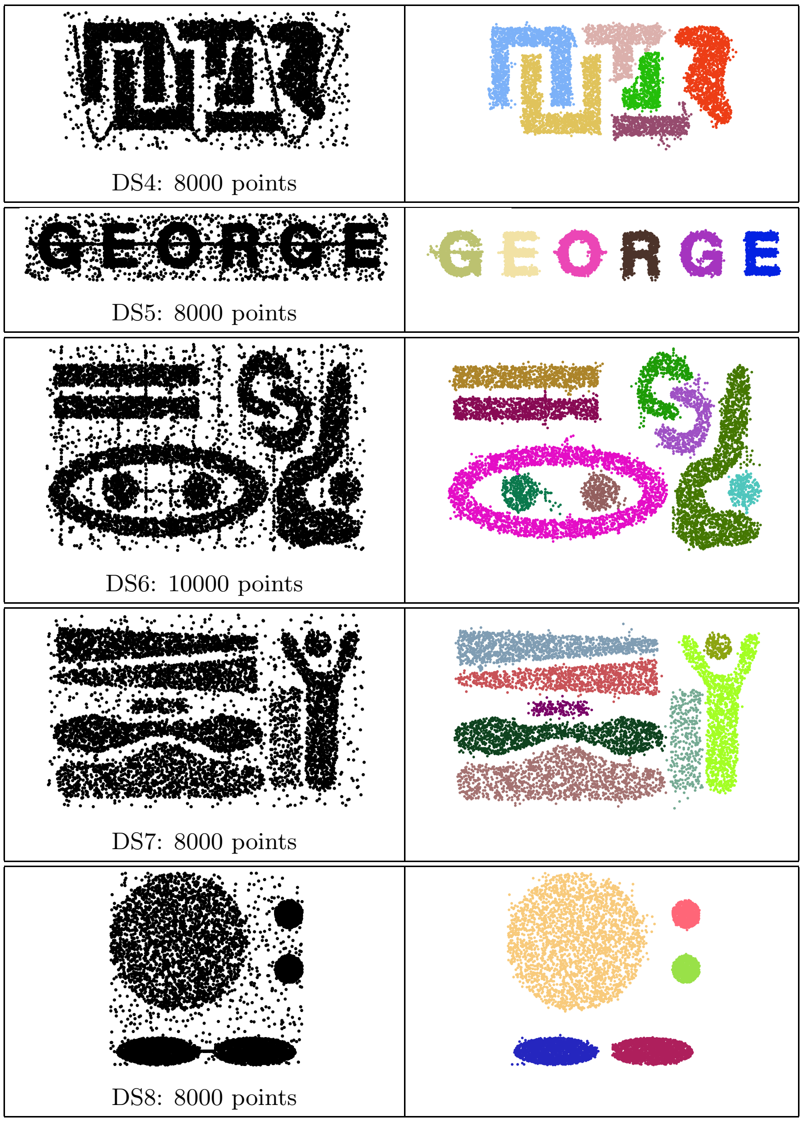
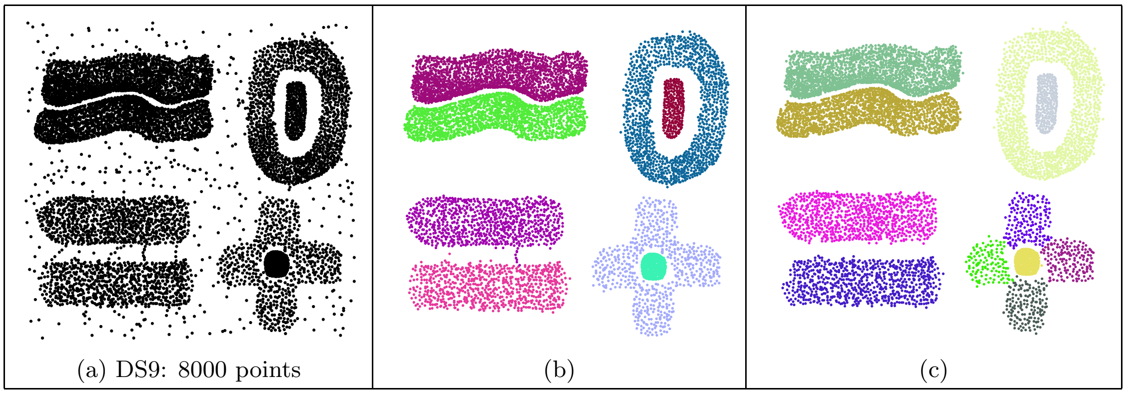


Figure 8: Data sets taken from [KHK99]

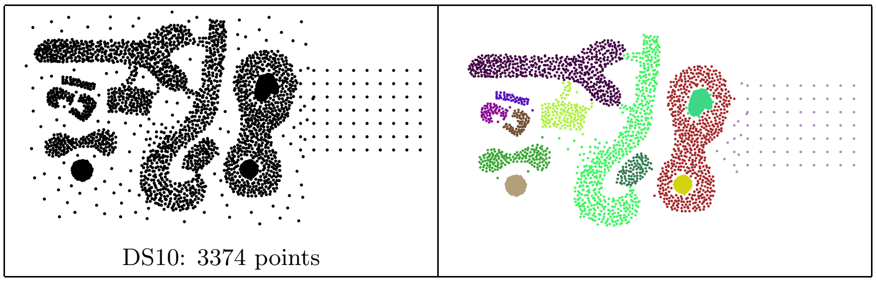
8 Related Work

Two recently published graph-based approaches to spatial data clustering, aimed at overcoming the limitations of prior methods, are [KHK99] and [EL00]. [EL00] has the advantage of being completely parameter-free but it might face problems in clustering complex data, for example



(a) DS9: 8000 points (b) (c)

Figure 9: Two different clusterings of a data set taken from [EL00]



DS10: 3374 points

Figure 10: A data set with clusters of different densities

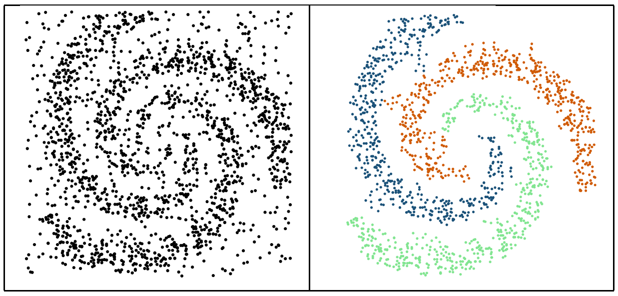


Figure 11: A 2012 points data set taken from [GWM99]

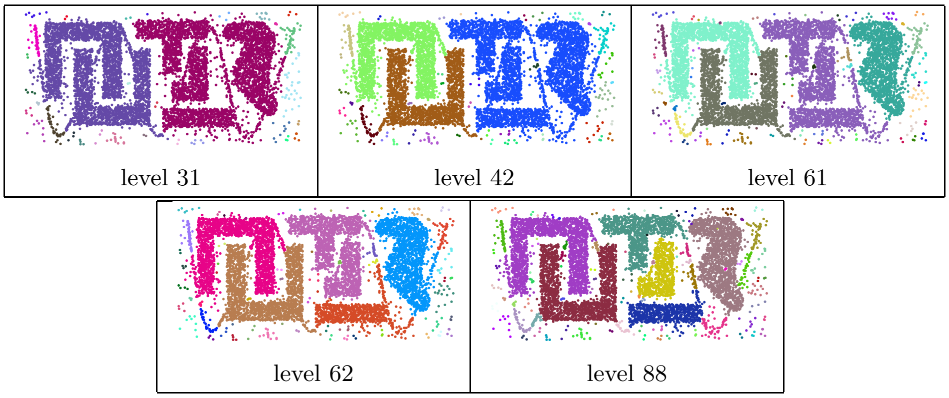


Figure 12: A hierarchy containing five decompositions of DS4 corresponding to the five levels with the highest prominency rank. (Level index indicates the number of clusters.)

Data Set	Size	Graph construction	Separation	Agglomeration	Overall	Ratio $\frac{Points}{Sec}$
DS4	8000	0.4	0.88	0.19	1.47	5434
DS5	8000	0.41	0.83	0.19	1.43	5587
DS6	10000	0.5	1.12	0.26	1.88	5311
DS7	8000	0.4	0.89	0.2	1.49	5358
DS8	8000	0.39	0.93	0.2	1.52	5256
DS9	8000	0.33	0.66	0.21	1.2	6656
DS10	3374	0.14	0.26	0.07	0.47	7178

Table 1: Running time (in seconds; non-optimized) of the various components of the clustering algorithm

the data set DS3, where it might have a difficulty in separating the clusters. Regarding [KHK99], it is hard for us to assess its quality since we do not have its implementation. However, the running time of [KHK99], which is $O(nm + n \log n + m^2 \log m)$ for $m \sim 0.03n$, is slower than ours.

Modeling the data points by a weighted graph is a very common approach. However, not many methods perform a deep analysis of the graph. Two methods that do carry out such an analysis are those of [BWD97], [GWM99]. They employ a Monte-Carlo-like simulation for exploring the graphs, which appears to be slower than our deterministic analysis of the random walks. These methods probably have difficulties in showing a single decomposition with clusters of highly differing densities as in, e.g., the data set DS10.

Random walks were first used in cluster analysis in [HK92]. However, the properties of the random walk there are not computed deterministically, but by a random algorithm that simulates a $\Theta(n^2)$ random walk. This results in time and space complexity of $\Theta(n^3)$ and $\Theta(n^2)$, respectively, even on bounded degree graphs.

Finally, we mention [GRS99], in which an agglomerative clustering algorithm is described that merges the two clusters with the (normalized) greatest number of common neighbors. To the best of our knowledge, this is the first agglomerative algorithm that considers properties related directly to the structure of the graph. Our work can be considered to be a rather extensive generalization of [GRS99], in the sense that it considers weights of edges and adds considerations related to larger neighborhoods.

9 Conclusion

We have introduced a novel algorithm for achieving a hierarchical clustering of spatial data, using random-walk-based separating operators. This approach seems to have several advantages.

First, it is robust in the presence of noise and outliers, and is flexible in handling data of different densities. It can reveal clusters of any shape without a special tendency towards spherically shaped clusters or ones of similar sizes. At the same time, the decisions the algorithm makes are based on the relevant structure of the associated graph, making it essentially immune to outliers and noise.

The second advantage is the running time and space requirements. The time complexity of our algorithm applied to n data points is $O(n \log n)$, and its practical running time, in general, is very fast. We have been able to cluster 10,000 points in less than two seconds. Regarding space complexity, one of the variants of the algorithm can be applied with a constant amount of space.

Since the algorithm does not rely on spatial knowledge, we plan to try it on other types of data. We have already used a variant of it for image segmentation with very encouraging results, and will report on it separately.

References

[BWD97] M. Blatt, S. Wiseman and E, Domany, "Data Clustering Using a Model Granular Magnet", *Neural Computation*, **9** (1997), 1805–1842.

[DE96] M. T. Dickerson and D. Eppstein, "Algorithms for proximity problems in higher dimensions", *Comp. Geom. Theory and Applications*, **5** (1996), 277–291.

[EL00] V. Estivill-Castro and I. Lee,"AUTOCLUST: Automatic Clustering via Boundary Extraction for Mining Massive Point-Data Sets", *5th International Conference on Geocomputation*, GeoComputation CD-ROM: GC049, ISBN 0-9533477-2-9.

[GRS99] S. Guha, R. Rastogi and K. Shim, "ROCK: A Robust Clustering Algorithm for Categorical Attributes", *Proceedings of the 15th International Conference on Data Engineering*, pp. 512–521, 1999.

[GWM99] Y. Gdalyahu, D. Weinshall and M. Werman, "Stochastic Image Segmentation by Typical Cuts", *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, 1999, pp. 588–601.

[HK92] L. Hagen and A. Kahng, "A New Approach to Effective Circuit Clustering", *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 422–427, 1992.

[JD88] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, New Jersy, 1988.

[JMF99] A. K. Jain, M.N. Murty and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys*, **31** (1999), 264–323.

[KHK99] G. Karypis, E. Han, and V. Kumar, "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling", *IEEE Computer*, **32** (1999), 68–75.

[SN99] B. Stein and O. Niggemann, "On the Nature of Structure and its Identification", *Proceedings 25th Inter. Workshop on Graph-Theoretic Concepts in Computer Science*, Lecture Notes in Computer Science, Vol. 1665, pp. 122–134, Springer Verlag, 1999.

[X+97] X. Xu , M. Ester, H.P. Kriegel and J. Sander, "Clustering and Knowledge Discovery in Spatial Databases", *Vistas in Astronomy*, **41** (1997), 397–403.