On Clustering Using Random Walks

David Harel and Yehuda Koren

Dept. of Computer Science and Applied Mathematics
The Weizmann Institute of Science, Rehovot, Israel
{harel,yehuda}@wisdom.weizmann.ac.il

Abstract. We propose a novel approach to clustering, based on deterministic analysis of random walks on the weighted graph associated with the clustering problem. The method is centered around what we shall call *separating operators*, which are applied repeatedly to sharpen the distinction between the weights of inter-cluster edges (the so-called separators), and those of intra-cluster edges. These operators can be used as a stand-alone for some problems, but become particularly powerful when embedded in a classical multi-scale framework and/or enhanced by other known techniques, such as agglomerative clustering. The resulting algorithms are simple, fast and general, and appear to have many useful applications.

1 Introduction

Clustering is a classical problem, applicable to a wide variety of areas. It calls for discovering natural groups in data sets, and identifying abstract structures that might reside there. Clustering methods have been used in computer vision [11,2], VLSI design [4], data mining [3], web page clustering , and gene expression analysis.

Prior literature on the clustering problem is huge, see e.g., [7]. 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.

In this paper, we present a new set of clustering algorithms, based on deterministic exploration of random walks on the weighted graph associated with the data to be clustered. We use the similarity matrix of the data set, so no explicit representation of the coordinates of the data-points is needed. 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 for some kinds of problems, but their power becomes more apparent when embedded in a classical multi-scale framework and when enhanced by other known techniques, such as agglomerative or hierarchical clustering.

The resulting algorithms are simple, fast and general. As to the quality of the clustering, we exhibit encouraging results of applying these algorithms to several

R. Hariharan, M. Mukund, and V. Vinay (Eds.): FSTTCS 2001, LNCS 2245, pp. 18–41, 2001.

recently published data sets. However, in order to be able to better assess its usefulness, we are in the process of experimenting in other areas of application too.

2 Basic Notions

We use standard graph-theoretic notions. Specifically, let $G(V, E, w)$ be a weighted graph, which should be viewed as modeling a relation E over a set V of entities. Assume, without loss of generality, that the set of nodes V is $\{1, \ldots, n\}$. 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.

Given a weighted graph $G(V, E, w)$, the associated *transition matrix*, denoted by M^G, is the $n \times n$ matrix in which, if i and j are connected, the (i, j)'th entry is simply p_{ij}. Hence, we have

$$M^G_{ij} = \begin{cases} p_{ij} & \langle i, j \rangle \in E \\ 0 & \text{otherwise} \end{cases}$$

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. Thus, $P^k_{visit}(i)$ is the i-th row in the matrix $(M^G)^k$, the k'th power of M^G.

The *stationary distribution* of G is a vector $p \in \mathbb{R}^n$ such that $p \cdot M^G = p$. An important property of the stationary distribution is that if G is non-bipartite, then $P^k_{visit}(i)$ tends to the stationary distribution as k goes to ∞, regardless of the choice of i.

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:

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

The values of ρ_i are calculated by solving these equations [1], 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 N data points are specified either in term of their coordinates in a d-dimensional space or by means of an $n \times n$ *similarity matrix*, whose elements s_{ij} measure the similarity of data points i and j.

Our algorithms use the similarity matrix only, and thus can deal with cases where pairwise similarity is the only information available about the data. Specifically, we address the problem of clustering the weighted graph $G(V, E, w)$. Most often we prefer to model the data using sparse graphs, which contain only a small subset of the edges of the complete graph, 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 expresses the arrangement of the data, thus helping the clustering process.

A preferred quality of a clustering algorithm is its ability to determine the number k of natural clusters. In reality, however, most clustering algorithms require this number to be an input, which means that they may break up or combine natural clusters, or even create clusters when no natural ones exist in the data.

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.

Various cost functions, sometimes called objective functions, have been proposed in order to measure the quality of a given clustering. Viewing the clustering problem as an optimization problem of such an objective function formalizes the problem to some extent. However, we are not aware of any function that optimally captures the notion of a 'good' cluster, since for any function one can exhibit cases for which it fails. Furthermore, not surprisingly, no polynomial-time algorithm for optimizing such cost functions is known. In fact, a main role of cost

[1] Notice that when multiplying each row i with d_i, the weighted degree of the respected node, the system is represented with a symmetric positive-definite matrix, which is easier to be solved

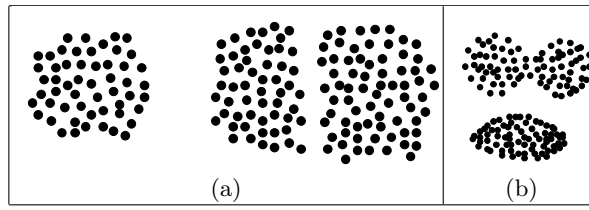


Fig. 1. Inherent ambiguity in clustering: How many clusters are there here?

functions for clustering is to obtain some intuition about the desired properties of a good clustering, and to serve as an objective metric for distinguishing a good clustering from a bad one.

3.1 Clustering Methods

We now survey some clustering approaches. Instead of providing specific references to each method, we point the reader to the surveys in [7,8].

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* (that is suitable only for points in a metric space) 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. For example, consider the graph in Figure 2. A natural clustering decomposition of this graph is into two rectangular grids, the larger left-hand-side grid and the smaller right-hand-side grid. However, these methods will attach some nodes of the left-hand-side grid to the nodes of the right-hand-side grid, seeking to minimize the distance of each node to the center of its related cluster.

Hierarchical algorithms create a sequence of partitions in which each partition is nested into next partition in the sequence. *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 variants are the *Single-Link* and the *Complete-Link* algorithms. In Single-Link clustering similarity between clusters is measured as the similarity between the most similar pair of elements, one from each of the clusters, while in Complete-Link clustering the similarity is measured using the least similar pair of elements.

The Complete-Link algorithm tends to break up a relatively large (though natural) cluster into two (unnatural) clusters, and will face similar difficulties to the partitional algorithms discussed above. The Single-Link algorithm has a different problem — the "chaining effect": it can be easily fooled by outliers, merging two clusters that are connected by a narrow string of points. For example, when activated on the graph of Figure 2, it will fail, since the distance between the left-hand-side grid and the right-hand-side grid, is equal to the distance between any two adjacent subgraphs.

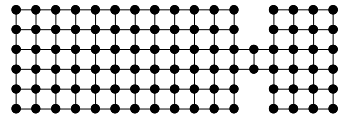


Fig. 2. A natural clustering decomposition of this graph is to divide it into two clusters: the left-hand-side larger grid and the right-hand-side smaller grid. The two nodes connecting these grids are outliers. Our clustering method reveals this decomposition, unlike many traditional clustering methods that will not discover it.

4 Cluster Analysis by Random Walks

4.1 Cluster Quality

Our work is motivated by the following predicate, with which we would like to capture a certain notion of the quality of a cluster:

Definition 4.1 *A cluster C is* (d, α)-normal *iff for every $i, j \in C$ for which $dist(i, j) \leq d$, the probability that a random walk originating at i will reach j before it visits some node outside C, is at least α.*

The role of α is obvious. The reason for limiting the distance between i and j to some d is that for clusters with a large enough diameter it may be easier to escape out of the cluster than to travel between distant nodes inside the cluster. This demonstrates the intuition that in a natural cluster, we need not necessarily seek a tight connection between *every* two nodes, but only between ones that are close enough. For example, consider Figure 2. Random walks starting at nodes in the right-hand-side of the of the large cluster, will probably visit close nodes of the other cluster, before visiting distant nodes of their own cluster.

In fact, the normality predicate can be seen to define the intuitive notion of discontinuities in the data. Such discontinuities indicate the boundaries of the clusters, and are created by sharp local changes in the data.

The normality predicate may label as good the clusters in different clustering decompositions of the same graph. This may be important in some cases,

like when we want to identify a hierarchical clustering decomposition. A disadvantage of this predicate is that when a cluster is not well separated from its neighborhood, the normality predicate may fail to declare the cluster as natural, even though its global structure might be very natural. For example consider Figure 3. A natural clustering decomposition of this graph is to separate the left-hand-side and the right-hand-side grids. However, the normality predicate will not necessarily label these two clusters normal, since there is a relatively wide portion connecting them.

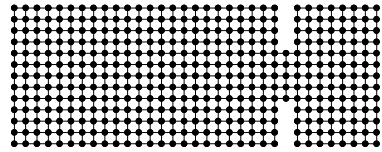


Fig. 3. Giving the normality predicate a hard time

Having said all this, we note that we do not have an efficient method for finding a clustering decomposition that adheres exactly to the normality predicate. However, the algorithms we have developed were conceived of to adhere to its spirit.

4.2 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 (when a discontinuity is revealed), or, rather, if the relationship between its two incident nodes is sufficiently intimate for them to be contained in a common cluster.

Definition 4.2 *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, in the context of the normality predicate, 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 local. Our philosophy, therefore, is that after identifying the separators by local considerations, we will deduce the global structure of the clustering decomposition by solving an easy global problem of finding connected components.

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 (when it exists). 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 4.3 *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:*

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

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

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

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

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

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

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

The key component in computing $NS(G)$ is the calculation of $P^{\leq k}_{visit}(v)$ and $P^{\leq k}_{visit}(u)$. If the graph G is of bounded degree, $P^{\leq k}_{visit}(u)$ 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, for bounded degree graphs $NS(G)$ can be computed in space $O(1)$ and time $\Theta(|E|)$, which in this case is just $\Theta(|V|) = \Theta(n)$.

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 4.4 *Let $G(V, E, w)$ be a graph, and let k be some constant. Denote by $P^{(k)}_{escape}(v, u)$ 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:*

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

We can now define separation by circular escape:

Definition 4.5 *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:*

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

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 $O(1)$.

4.3 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.

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 $O(|E|)$ 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.

We demonstrate this method by several examples. Consider Figure 4, which contains an almost uniformly weighted graph, taken from [12]. We experimented with both separating operators, each one with a four-fold iteration. The NS operator was used with $k = 3$ and $sim^k(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{dfn}}{=} f^k(\boldsymbol{x}, \boldsymbol{y})$ and the CE operator with $k = 2$, other choices work very similarly. 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 particularly interesting, and would probably 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 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

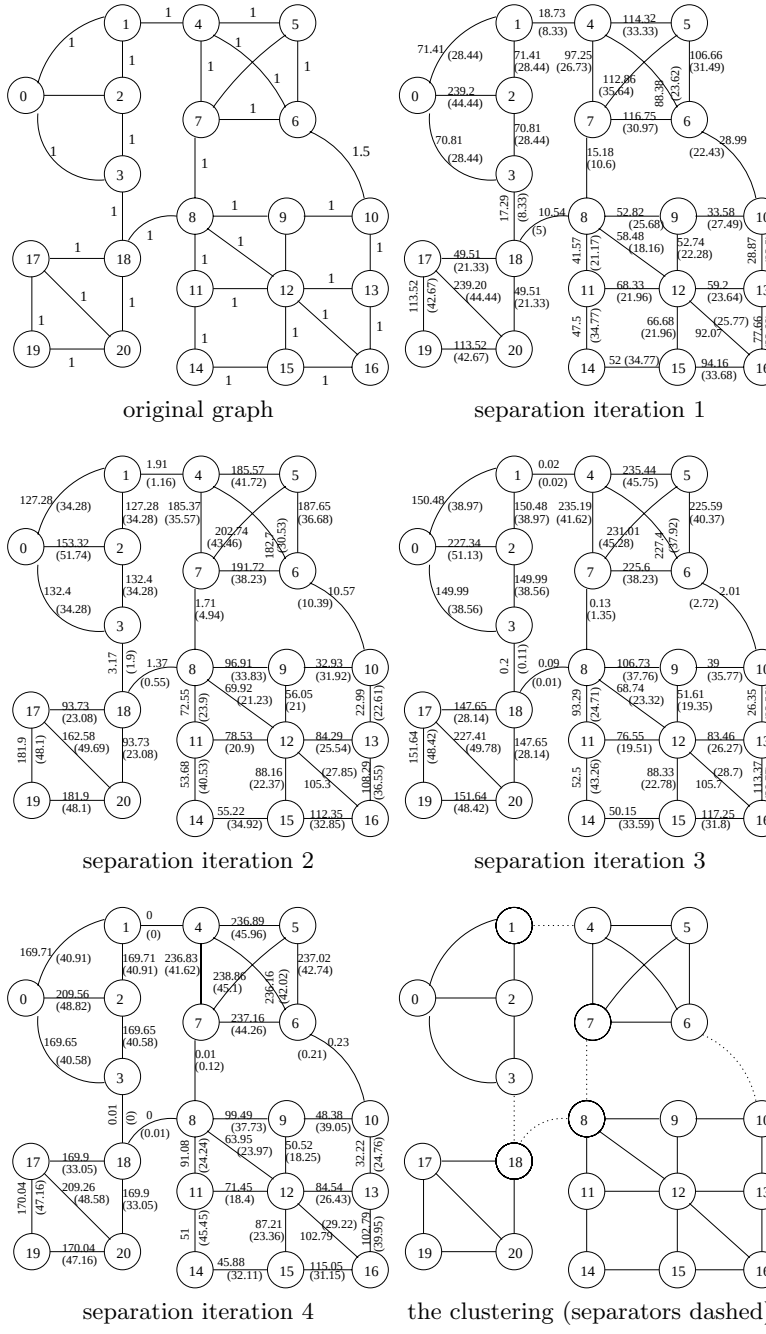


Fig. 4. Clustering using four-fold application of separation operators, which sharpen the edge weight distinction (NS values are on top and CE values, multiplied by 100, are in parentheses); example taken from [11]

the clustering methods that rely on edge-connectivity will fail here, since the edge-connectivity between every two nodes of the tree is one.

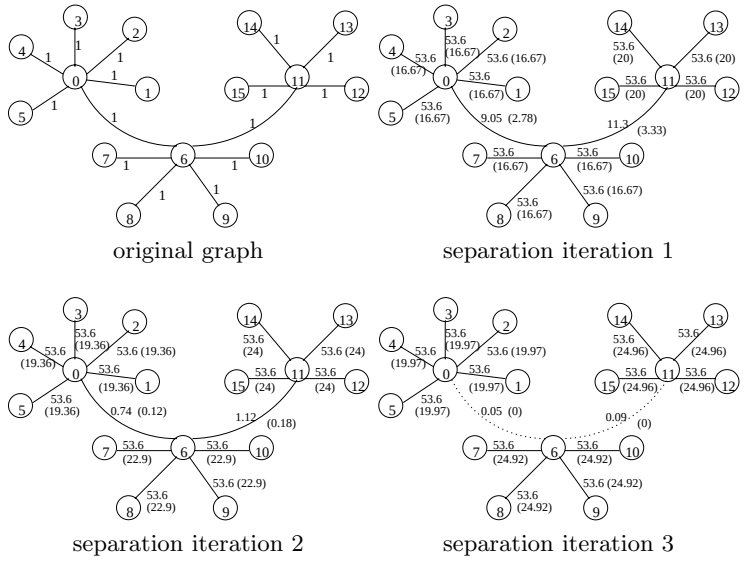
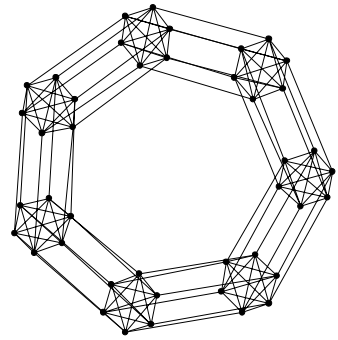


Fig. 5. Clustering a tree using three-fold application of separation (separators are dashed in iteration 3)

Figure 6 shows seven copies of the complete graph K_6 arranged cyclically. Each node is linked by 'internal' edges to the other five nodes in its own complete subgraph, and by 'external' edges to two nodes of the neighboring complete subgraphs. All edges have uniform weight. In the table, one can see how iterations of the separating operators diminish the weights of the 'external' edges, which are clearly found to be separators, decomposing the graph into seven clusters of complete subgraphs.

When applying the separating operators to the graph of Figure 2, the edges of the lowest sharpened weight are those outside the two grids, resulting in the decomposition of the graph into three clusters, as shown in Figure 7.

The graph in Figure 8(a) demonstrates the application of our method to a weighted graph. The weight of edges of this graph have been set up to decrease exponentially with their length. After a three-fold iteration of the CE separating operator with $k = 3$, and declaring the edges with weight below 0.097 as separators, the graph is decomposed into two clusters depicted in Figure 8(b). Slight changes to the value of k, or applying the NS separating operator, produce similar results, where several nodes on the chains connecting the upper and the lower clusters become independent clusters of outliers.



	1	2	3	4	5	6
CE	30.56 / 16.21	33.16 / 9.51	34.51 / 4.74	35.31 / 1.65	35.77 / 0.26	35.96 / 0
NS	191.38 / 12.08	279.17 / 0.33	287.14 / 0.01	287.3 /0	287.3 /0	287.3 / 0

Fig. 6. Clustering a cycle of complete graphs. Edges are of two kinds: internal edges that link two nodes of the same complete subgraph and external edges linking nodes of different subgraphs. The table shows the sharpened weights of internal/external edges after each of six iterations of separation.

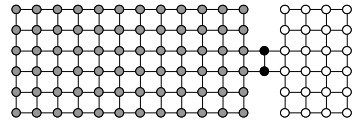
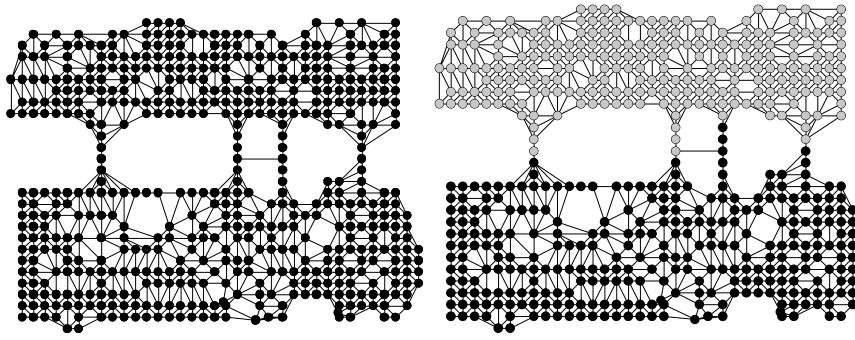


Fig. 7. Clustering the graph of Figure 2. The 3 clusters are denoted by different colors.



(a) (b)

Fig. 8. (a) A weighted graph (edge weights are decaying exponentially with their length); (b) decomposition of the graph into two clusters.

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.
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.

4.4 Clustering Spatial Point-Sets

We now illustrate the ability of our method to cluster "correctly" 2D sets of points, in a number of typical cases, some of which have been shown to be problematic for agglomerative methods [9]. (More extensive examples are given in Subsection 5.1.) For a short version of this paper that deals with clustering spatial data, see [5].

We have used 10-mutual neighborhood graphs for modeling the points. 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 edge weights, we adopt a commonly used approach: 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.

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 9 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.

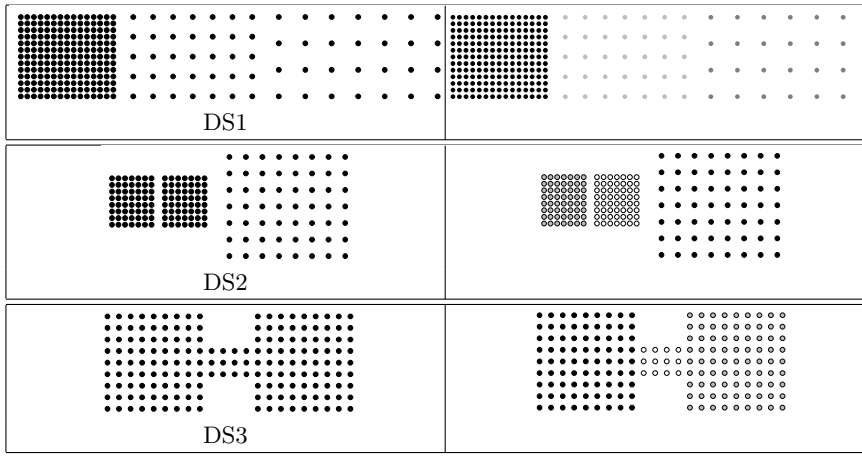


Fig. 9. Clustering of several data sets. Different clusters are indicated by different colors.

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 10.

5 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 preprocessing sharpens the edge weights, adding structural knowledge to them, and greatly enhances the agglomerative algorithms, as it can effectively prevent bad local merging that works against 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

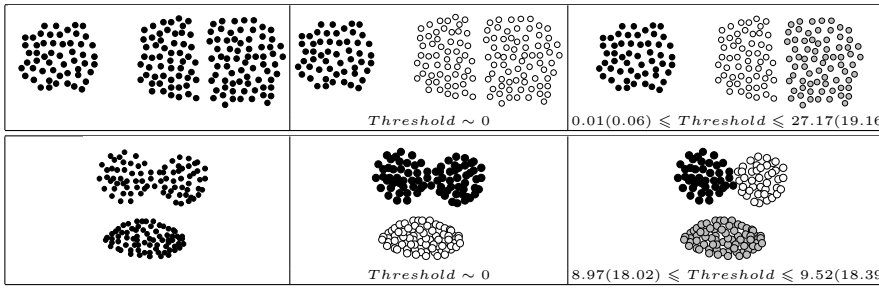


Fig. 10. Clustering at multiple resolutions using different thresholds. When values of CE are different from values of NS, the CE values are given in parentheses. CE values are multiplied by 100.

weight of the edge between t and the contracted node uniquely distinguishes between different variants of the agglomerative procedure. For example, when using 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 equivalent to a 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, which 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. It does not directly addresses the question of which are the meaningful decompositions inside the dendrogram.

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 [2], a rather effective way of measuring the importance of a level is by evaluating how sharp is the change that this level introduces into the clustering decomposition. Since changes that are involved with small clusters do not have a large impact, 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.

5.1 Examples

In this section we show the results of running our algorithm on several data sets from the literature. 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 4 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. The reader is encouraged to see [6], in order to view the figures of this section in color.

Figure 11 shows the results of the algorithm on data sets taken from [9]. 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 12 shows the result of the algorithm applied to a data set from [1]. 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 12(c).

In Figure 13, which was produced by adding points to a data set given in [1], 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 14, which in a way is the most difficult one we have included, is taken from [2]. We have modeled the data exactly the same way described in [2], by putting an edge between every two points whose distance is below some threshold. Using this model, [2] 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 5 to reveal the most meaningful levels in the dendrogram. Figure 15 demonstrates its capability with respect to the data set DS4 (shown in Figure 11). 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.

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

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

6 Multi-scale Clustering

In this section we embed our separation operators inside a classical multi-scale scheme. The resulting algorithm subsumes the agglomerative variant presented in Section 5.

A multi-scale treatment of a graph-related problem handles the graph in a global manner, by constructing a *coarse abstraction* thereof. The abstraction is a new graph that contains considerably fewer nodes than the original, while preserving some of its crucial properties. Dealing with a global property for the coarse graph may be easier, since it contains much less information, and hopefully still has the desired property. A *multi-scale* representation of the graph consist of various coarse abstractions that allow us to view the graph on different scales, that differ in the level of abstraction they represent. For example, see [10,11].

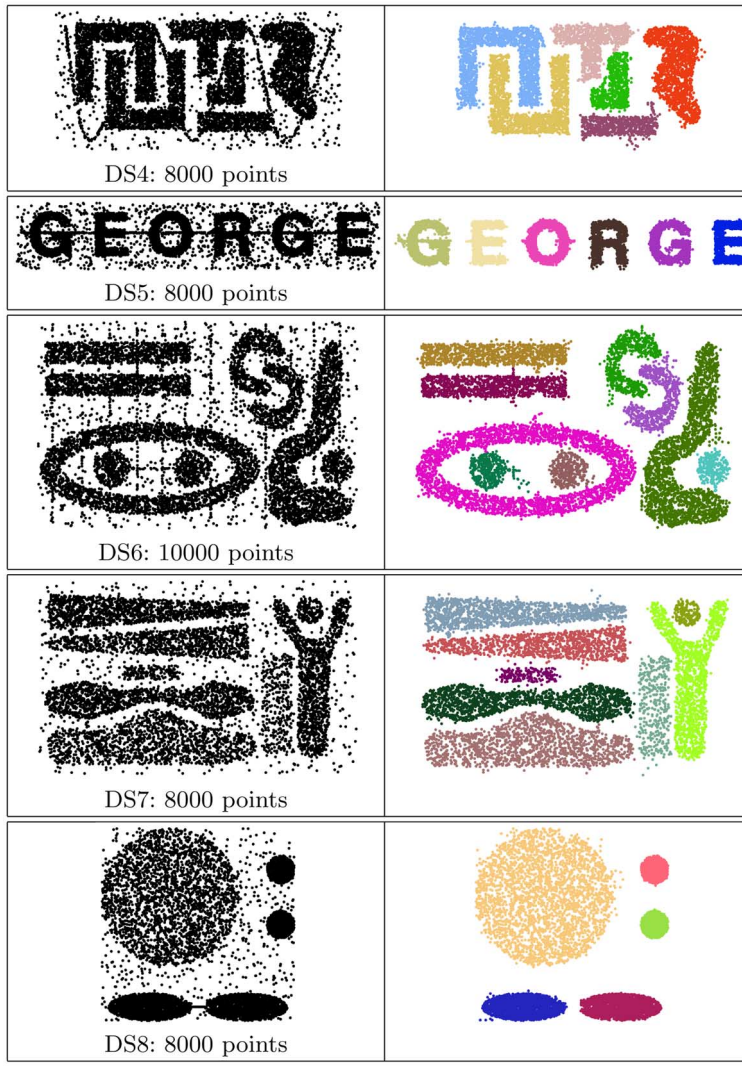


Fig. 11. Data sets taken from [9] (see [6] for clearer color versions of this figure and of Figs. 12–15).

6.1 The General Scheme

In our context, we find that the multi-scale technique is often called for in order to identify clusters whose naturalness stems from the graph's global structure, and which would be very difficult to identify using only local considerations. Such clusters are not well separated from their surroundings. For example, there might be wide 'channels of leaked information' between such a cluster and others, disrupting separation. If we were able to construct a coarse graph in which a wide

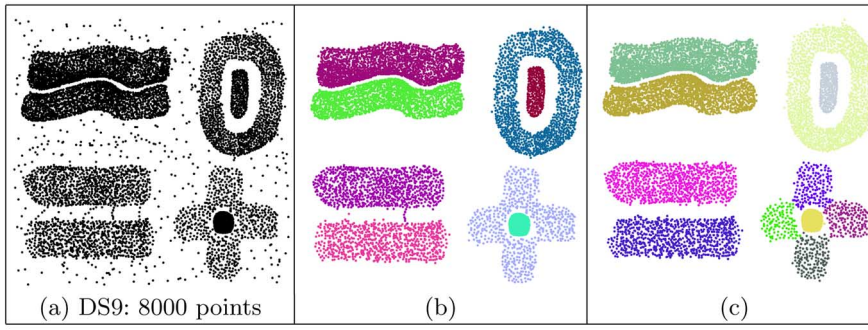


Fig. 12. Two different clusterings of a data set taken from [1]

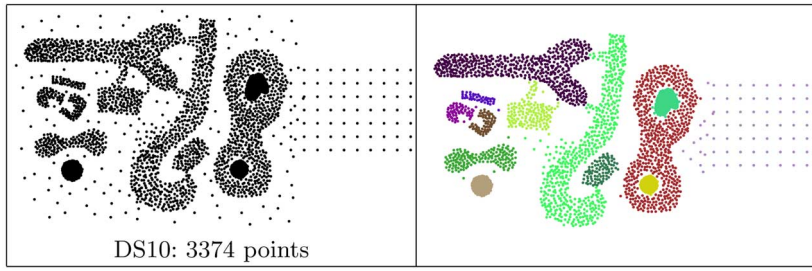


Fig. 13. A data set with clusters of different densities

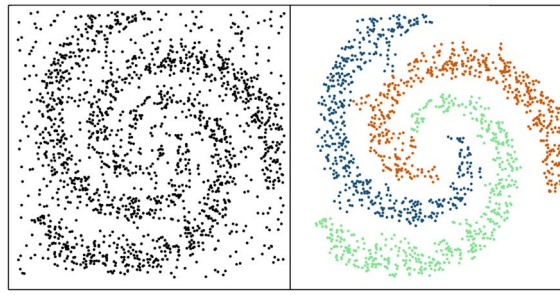


Fig. 14. A 2012 points data set taken from [2]

channel of connection is replaced by a single separating edge, we would be able to overcome the difficulty.

For example, consider the graph in Figure 3. As mentioned earlier, the natural clustering decomposition of this graph does not obey the predicate for cluster normality introduced in Section 4.1, due to the broad five-edge connection between the two larger parts of the graph. Hence, our separating operators, which were developed in the spirit of this predicate, will have a hard time identifying the natural decomposition of this graph. (The operators do manage to natu-

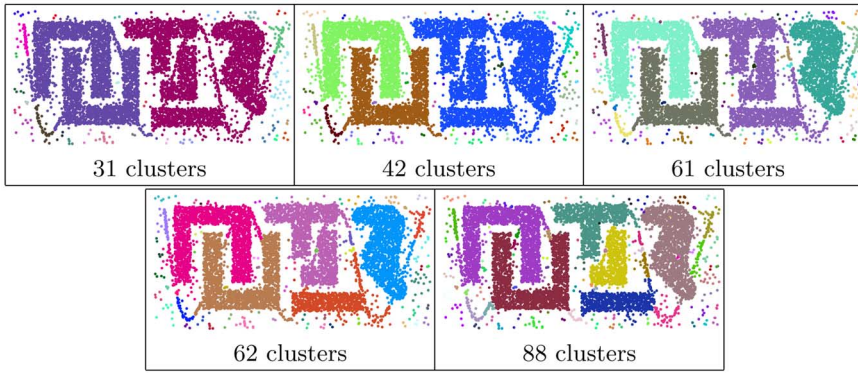


Fig. 15. A hierarchy containing five decompositions of DS4 corresponding to the five levels with the highest prominency rank.

rally decompose this graph if they applied with a relatively large value of k, i.e., $k > 5$.) The multi-scale solution we propose below overcomes this situation by constructing a coarse representation similar to that of Figure 2, in which the natural decomposition is correctly identified. We can then use the decomposition of the coarse graph to cluster the original one, as long as we have a good way of establishing a correspondence between the left and right grids of the two graphs, respectively.

Here, now, is a high-level outline of the multi-scale clustering algorithm.

MS-Clustering $(G(V, E, w))$

1. compute iterated sharpened weights of G's edges.
2. *if* G is small enough *then*
   return a clustering decomposition of G.
3. construct $G^C(V^C, E^C, w^C)$, a coarser abstraction of G, such that $|V^C| = \alpha \cdot |V|$, where $0 < \alpha < 1$.
4. call **MS-Clustering**$(G^C(V^C, E^C, w^C))$.
5. obtain a clustering of G by projecting the clustering of G^C onto G.
6. improve the clustering of G using a greedy smoothing procedure.
7. *end*

6.2 Structure Preserving Coarsening

Clearly, the key step in the algorithm is the computation of a coarse graph, which we now set out to describe. A common approach to coarsening graphs is to use a series of *edge-contractions*. In a single operation of edge-contraction we pick some edge $\langle v, u \rangle$, and combine the two nodes v and u ('fine nodes') into a single super-node $v \cup u$ ('coarse-node'). In order to preserve the connectivity information in the coarse graph, we take the set of edges of $v \cup u$ to be the union

of the sets of edges of v and u. If v and u have a common neighbor t, the weight of the edge $\langle v \cup u, t \rangle$ is taken to be $w(v,t) + w(u,t)$.

A coarse graph is only useful to us if it retains the information related to the natural clustering decomposition of the original graph. Hence, we seek what we call a *structure preserving coarsening* in which large-enough natural clusters of the original graph are preserved by the coarsening. A key condition for this is that a coarse node does not contain two fine nodes that are associated with different natural clusters; or, equivalently, that we do not contract a separating edge.

To achieve this, we select the edges to be contracted by considering the sharpened weights of the edges — those obtained by using our separating operators — and contract only edges with high sharpened weights. We would like to eliminate the tendency of such a procedure to contract pairs of large nodes whose connectivity is high due to their sizes. Accordingly, we normalize the sharpened weights by dividing them by some power of the sizes of the relevant nodes.

The hope is that the kind of wide connections between natural clusters that appear in Figure 3 will show up as sets of separators. This is based on the fact that connections between sets of nodes that are related to the same cluster should be stronger than connections between sets that are related (even partially) to different clusters.

After finding the clustering decomposition of the coarse graph, we deduce the related clustering decomposition of the original graph by a simple projection based on the inclusion relation between fine and coarse nodes. The projected clustering might need refining: When the wide connections indeed exist, it may be hard to find the 'right' boundary of a natural cluster, and some local mistakes could occur during coarsening. We eliminate this problem by adding a *smoothing* phase (line 6 of the algorithm), in which we carry out an iterative greedy process of exchanging nodes between the clusters. The exchanges are done in such a way that each node joins the cluster that minimizes some global cost-function (we have chosen the multi-way cut between all the clusters). This kind of smoothing is similar to what is often done in graph partitioning; see, e.g., [10].

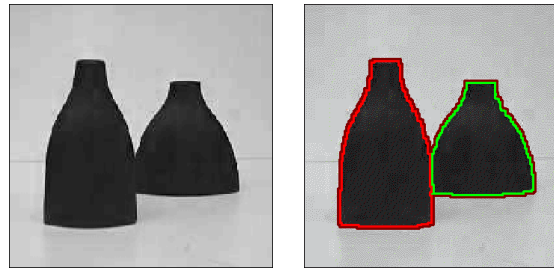
6.3 Relationship to Agglomerative Clustering

The edge contraction operation in our multi-scale clustering method is essentially the same as the merging of two clusters in agglomerative algorithms.

The main difference between the multi-scale method and the agglomerative variant introduced in Section 5, is in the use of the separating operators on the coarse graphs, not only on the original fine graph: at certain levels of the process we sharpen the edge weights by these operators. This way, the operators act upon more global properties, which can be identified on the coarser graphs. Another advantage of the multi-scale algorithm, is its utilization of the smoothing process, which can undo erroneous merges.

We have found this multi-scale algorithm superior for the task of image segmentation. A common approach to image segmentation is to represent the image by a weighted graph whose nodes are the pixels of the image. There are edges

connecting each pixel with its four immediate neighbors, and the weight of an edge is determined by the similarity of the intensity levels of the incident pixels. Figure 16 shows the ability of the multi-scale algorithm to accurately separate the two vases, in spite of the large connectivity between them. We are still in the process of investigating the use of our ideas in image segmentation, and we expect to present additional results.



(a) (b)

Fig. 16. (a) original 350×350 image (taken from [10]); (b) segmentation: each vase forms its own segment

7 Related Work

Random walks were first used in cluster analysis in [4]. 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.

A recent algorithm that uses deterministic analysis of random walks for cluster analysis is that of [13]. The approach there is quite different from ours. Also, its time and space complexity appear to be $\Omega(n^3)$ and $\Theta(n^2)$, respectively, even for bounded degree graphs.

A recently published graph-based approach to clustering, aimed at overcoming the limitations of agglomerative methods, is [9]. It is hard for us to assess its quality since we do not have its implementation. However, the running time of [9], which is $O(nm + n \log n + m^2 \log m)$ for $m \sim 0.03n$, is slower than ours.

Finally, we mention [3], in which an agglomerative clustering algorithm is described that merges the two clusters with the (normalized) greatest number of common neighbors. To our best 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 this work, in the sense that it considers weights of edges and adds considerations related to larger neighborhoods.

8 Conclusion

The process of using random-walk-based separating operators for clustering seems to have a number of major advantages. One advantage is in the quality of the resulting clustering. Our algorithms can reveal clusters of any shape without a special tendency towards spherically shaped clusters or ones of similar sizes (unlike many clustering algorithms that tradeoff these features for being robust against outliers). At the same time, the decisions the algorithms make are based on the relevant structure of the graph, making them essentially immune to outliers and noise.

Another advantage is the running time. Our separating operators can be applied in linear time when the graphs are of bounded degree, and their running time in general is very fast. We have been able to cluster 10,000-node planar graphs in less than two seconds, on a 700MHz Pentium III PC.

In addition to developing the algorithms themselves, we have also attempted to define a rather general criterion for the naturalness of a cluster (Section 4.1). We hope to use this criterion in the future as the basis of improved algorithms, and to better study the connections between it and random-walk-based separation.

Finally, we believe that the structure preserving coarsening introduced in Section 6 can be used to improve other algorithms that perform coarsening on structured graphs, e.g., multi- scale graph drawing algorithms and multi-level graph partitioning algorithms [10].

References

1. 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.
2. 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.
3. 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.
4. 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.
5. D. Harel and Y. Koren, "Clustering Spatial Data using Random Walks", *Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining* (KDD-2001), ACM, pp. 281–286, 2000.
6. D. Harel and Y. Koren, "Clustering Spatial Data Using Random Walks", Technical Report MCS01-08, Dept. of Computer Science and Applied Mathematics, The Weizmann Institute of Science, 2001. Available at: `www.wisdom.weizmann.ac.il/reports.html`
7. A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Englewood Cliffs, New Jersy, 1988.

8. A. K. Jain, M.N. Murty and P.J. Flynn, "Data Clustering: A Review", *ACM Computing Surveys*, **31** (1999), 264–323.
9. G. Karypis, E. Han, and V. Kumar, "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling", *IEEE Computer*, **32** (1999), 68–75.
10. G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing* **20**:1 (1999), 359–392.
11. E. Sharon, A. Brandt and R. Basri, "Fast Multiscale Image Segmentation", *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pp. 70–77, 2000.
12. B. Stein and O. Niggemann, "On the Nature of Structure and its Identification", *Proceedings 25th Workshop on Graph-Theoretic Concepts in Computer Science*, LNCS 1665, pp. 122–134, Springer Verlag, 1999.
13. N. Tishby and N. Slonim, "Data Clustering by Markovian relaxation and the Information Bottleneck Method", *Advances in Neural Information Processing Systems 13*, 2000.