Stochastic Image Segmentation by Typical Cuts

Yoram Gdalyahu Daphna Weinshall Michael Werman
Institute of Computer Science, The Hebrew University, 91904 Jerusalem, Israel
e-mail: {yoram,daphna,werman}@cs.huji.ac.il

Abstract

We present a stochastic clustering algorithm which uses pairwise similarity of elements, based on a new graph theoretical algorithm for the sampling of cuts in graphs. The stochastic nature of our method makes it robust against noise, including accidental edges and small spurious clusters. We demonstrate the robustness and superiority of our method for image segmentation on a few synthetic examples where other recently proposed methods (such as normalized-cut) fail. In addition, the complexity of our method is lower. We describe experiments with real images showing good segmentation results.

1 Introduction

There are many situations where one has pairwise similarity measurements between a set of elements and one wants to partition the set into *meaningful* parts (clusters). Image segmentation and partitioning an image data base into different types of pictures are two examples, where the source of pairwise similarity measurements might be local brightness similarity or color histogram similarity; in both problems clustering of the data is sought. This paper presents a new general approach to clustering which is robust, hierarchical, computationally efficient, and produces nice results.

Clustering algorithms can be divided into two categories: those that require a vectorial representation of the data, and those which use only pairwise representation. In the former case, every data item must be represented as a vector in a real normed space, while in the second case only pairwise relations of similarity or dissimilarity are used. The pairwise information can be represented by a weighted (and perhaps incomplete) graph $G(V, E)$: the nodes V represent data items, and the positive weight w_{ij} of an edge (i, j) represent the amount of similarity or dissimilarity between items i and j. In the rest of this paper w_{ij} represents a similarity value.

In this paper we only discuss direct methods for pairwise clustering, which do not involve the embedding of data items in a vector space. One such strategy is to use a similarity threshold θ, remove edges with weight less than θ, and identify the connected components that remain as clusters. A transformation of weights may precede the thresholding, to reflect transitive similarity relations. One recent example is the physically motivated transformation proposed in [1], which uses a granular magnet model and replaces weights by "spin correlations". Our algorithm is similar to this model.

A second pairwise clustering strategy is used by agglomerative algorithms [2], which start with the trivial partition of N points into N clusters of size one, and continue by subsequently merging pairs of clusters. At every step the two clusters which are most similar are merged together, until the similarity of the closest clusters is lower than some threshold. Different similarity measures between clusters distinguish between different agglomerative algorithms. In particular, the single linkage algorithm defines the similarity between clusters as the maximal similarity between two of their members, and the complete linkage algorithm uses the minimal value.

A third strategy of pairwise clustering uses the notion of cuts in a graph. A cut (A, B) in a graph $G(V, E)$ is a partition of V into two disjoint sets A and B. The capacity of the cut is the sum of weights of all edges that cross the cut, namely: $c(A, B) = \sum_{i \in A, j \in B} w_{ij}$. Among all the cuts that separate two marked vertices, the *minimal cut* is the one which has minimal capacity. The minimal cut clustering algorithm [9] divides the graph into components using a cascade of minimal cuts.

In a related approach, the normalized cut algorithm [8] uses the association of A (sum of weights incident on A) and the association of B to normalize the capacity $c(A, B)$. In contrast with the easy min-cut problem, the problem of finding a minimal normalized cut is NP-hard, but with certain approximations it reduces to a generalized eigenvalue problem [8]. This brings us to the family of algorithms that use spectral methods for clustering. One such approach uses the first eigenvector of the similarity matrix [7] to separate figure from ground.

Other pairwise clustering methods exist [5]. However, the three categories of methods above are of special importance to us, since our current work provides a common framework for all of them. Specifically, our new algorithm may be viewed as a randomized version of an agglomerative clustering procedure, and in the same time it generalizes the minimal cut algorithm. It is also strongly related to the physically motivated granular magnet model algorithm.

Our method is unique in its stochastic nature while provenly maintaining low complexity. In addition, our method provides a "natural" hierarchy of representations which is automatically determined. At a single hierarchical level our method performs as well as the aforementioned methods in "easy" cases, while keeping the good performance in "difficult" cases. In particular, it is more robust against noise and pathological configurations:

(i) A *minimal cut* algorithm is intuitively reasonable since it optimizes so that as much of the similarity weight remains within the parts of the clusters, and as little as possible is "wasted" between the clusters. However, it tends to fail when there is no clean separation into 2 parts, or when there are many small spurious parts due, e.g., to noise. Our stochastic approach avoids these problems and behaves more robustly.

(ii) The *single linkage* algorithm deals well with chained data, where items in a cluster are connected by transitive relations. Unfortunately the deterministic construction of chains can be harmful in the presence of noise, where a few points can make a "bridge" between two large clusters and merge them together. Our algorithm inherits the ability to cluster chained data; at the same time it is robust against such noisy bridges as long as the probability to select all the edges in the bridge remains small.

2 Stochastic pairwise clustering

Our randomized clustering algorithm (a preliminary version of which is described in [3]) is constructed of two main steps:

1. Stochastic partitioning of the similarity graph into r parts (by randomized agglomeration). For each partition index r ($r = N \ldots 1$):
   (a) We sample a probability distribution defined over cuts in the similarity graph (Section 2.1.1);
   (b) Using this sample and for every pair of nodes, we compute the marginal probability that they remain in the same cluster in a random cut, and replace the weight of the edge between the two nodes by this probability (Section 2.1.2);
   (c) We form clusters (by "typical cut") using connected components and threshold of 0.5 (Section 2.1.3).
2. Selection of proper r values, which reflect "interesting" structure in our problem (Section 2.2.

2.1 Stochastic partitioning

2.1.1 Efficient sampling of cuts in graphs

On the way to bi-partite cuts we consider multi-way cuts. An *r-way cut* is a partition of G into r components. The capacity of an r-way cut is the sum of weights of all edges that connect different components. In the rest of this paper we may refer to r-way cuts simply as "cuts".

For each r we induce a probability distribution on the set of *all* r-way cuts by assigning to each cut a probability, which decreases monotonically with its capacity. Hence, for example, the minimal cut is the most probable bi-partite cut in the graph, but other bi-partite cuts are also possible. The probability distribution over cuts is imposed by stochastic agglomeration. While a deterministic procedure, like single linkage, determines a single cut for each r, the stochastic procedure generates every r-way cut with probability that depends on its capacity.

More specifically, we use the contraction algorithm described in [6] which can be interpreted as stochastic version of the single linkage method. It starts with an N-way cut, corresponding to N clusters of size one, and continues by subsequently merging pairs of clusters. Thus at step $N - r$, the contraction algorithm generates a sample r-way cut.

At each step the contraction algorithm selects two clusters and merges them. The probability to select a pair of clusters is proportional to the sum of weights of all edges that connect the two clusters. It was shown in [6] that this mechanism induces a probability distribution over cuts, which decays rapidly with the cut capacity. Farther details can be found in the appendix.

The term "graph contraction" refers to a single path from N-way cut to 2-way cut. A single graph contraction generates for every r (r=N...1) a single sample of an r-way cut in the following way:

- select edge (i, j) with probability proportional to w_{ij}.
- replace nodes i and j by a single node $\{ij\}$.
- let the set of edges incident on $\{ij\}$ be the union of the sets of edges incident on i and j, but remove self loops formed by edges originally connecting i to j.
- repeat until 2 nodes remain.

It is shown in [6] that for general graphs a single graph contraction can be implemented in $O(N^2)$ time. In the appendix we develop a contraction scheme which lets us perform graph contraction on sparse graphs in $O(N \log N)$ time.

2.1.2 Transformation of weights

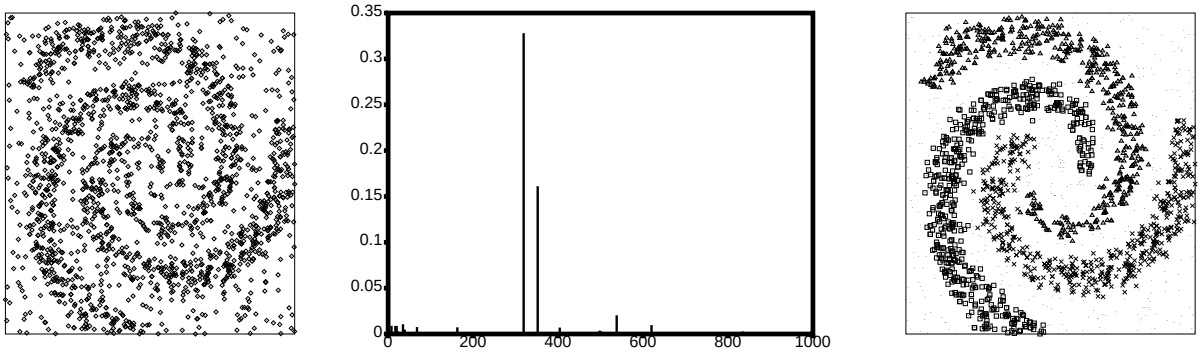
Using the probability distribution induced on r-way cuts, we compute the marginal probability p^r_{ij} that the edge between elements i and j does not cross a random r-way cut (or in other words, that elements i and j are in the same cluster of a random r-way cut). For every integer r between 1 and N we obtain a new graph by performing the following *weight transformation*: $w_{ij} \to p^r_{ij}$.

We can estimate p^r_{ij} by repeating the graph contraction M times and averaging these binary indicators (a better way is described below). Using the Chernoff inequality it can be shown[1] that if $M \geq (2 \ln 2 + 4 \ln N - 2 \ln \delta)/\epsilon^2$ then each p^r_{ij} is estimated, with probability larger than $1 - \delta$, within ϵ from its true value. Since a single graph contraction is of complexity $O(N^2)$ for complete graphs and $O(N \log N)$ for sparse graphs, it turns out that the complexity of estimating the marginal probabilities is $O(N^2 \log N)$ in the complete case, and $O(N \log^2 N)$ in the (more relevant) sparse case. Since in practice we only need to know whether p^r_{ij} is larger or smaller than 0.5, a more space-efficient method is described below.

2.1.3 Cluster formation

For every integer r between 1 and N we define the *typical cut* $(A_1, A_2, \ldots, A_s)$ as the partition of G into s components, such that for every $i \in A_\alpha$, $j \in A_\beta$ ($\alpha \neq \beta$, $\alpha, \beta = 1 \ldots s$) we have $p^r_{ij} < 0.5$. To find the typical cut for every integer r between 1 and N we first remove all the edges whose transformed weight p^r_{ij} is smaller than 0.5. We then compute the connected components in the remaining graph.

[1] Thanks to Ido Bregman for pointing this out.



(a) (b) (c)

Figure 1. Clustering of points in the Euclidean plane. (a) The 2000 data points; the coordinates of the points are not available for the clustering algorithm, which uses only the matrix of pairwise distances. (b) The graph of $\Delta T(r)$, computed for *every* integer r between 1 and N. Two peaks are clearly observed at r=320 and r=353. (c) The typical cut at r=354. The three large components are indicated by different symbols (crosses, triangles and squares) and the isolated background points are marked by small dots. At r=353 (not shown) the two spirals marked by crosses and triangles merge and form one cluster. At r=320 (also not shown) the three spirals form one cluster, and the background points remain isolated.

In order to compute a partition at every r level, we notice that p_{ij}^r is monotonic in r by construction. Thus it is sufficient to know for every i-j pair which r satisfies $p_{ij}^r = 0.5$ (denoted r_{ij}); we then remove the edge between elements i and j for every $r < r_{ij}$.

r_{ij} is found by repeating the contraction algorithm M times. In each iteration there exists a single r at which the edge between elements i and j is marked and the points are merged. Denote by r_m the level r which joins i and j in the m-th iteration ($m = 1 \ldots M$). The median r' of the sequence $\{r_1, r_2 \ldots r_M\}$ is the sample estimate for the level r_{ij} that satisfies $p_{ij}^r = 0.5$. We use an on-line technique (not described here) to estimate the median r' using small constant memory.

2.2 Hierarchical clustering

The stochastic partitioning of the similarity graph, described above, gives a cascade of nested partitions parameterized by an integer r between 1 and N. The question that remains is to define and choose "good" values of r, for which a "meaningful" clustering is obtained as part of a hierarchy of a few selected partitions.

We define the following function of the partition at each r:

$$T(r) = \frac{2}{N(N-1)} \sum_{i>j} N_i N_j$$

where N_i, N_j denote the number of elements in clusters i, j respectively. $T(r)$, therefore, measures how many edges cross over between different clusters in the r-partition, relative to the total number of edges in the complete graph.

Partitions which correspond to subsequent r values are typically similar to each other, or even identical, in the sense that only a few nodes (if any) change the component to which they belong. Consequently, $T(r)$ typically shows a very moderate increase. However, abrupt changes in $T(r)$ occur between different hierarchical levels of clustering, when two or more clusters are merged.

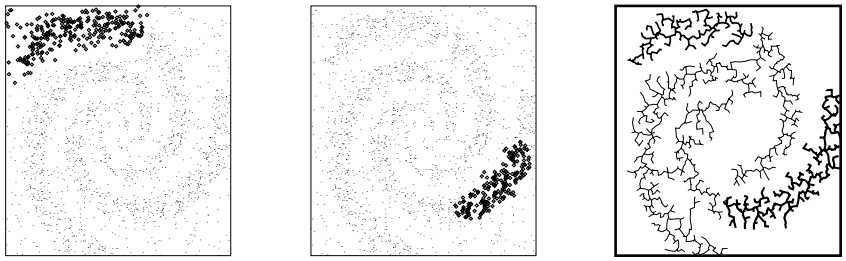
We propose to look at changes in the value of $T(r)$ between subsequent r values. For every integer r, the graph of $\Delta T(r)$ shows the (normalized) number of edges that are added to the typical cut in the transition from r-1 to r. Our heuristic scheme targets large peaks in this graph to identify "good" values of r where "meaningful" changes in the data organization occur.

Thus after computing the stochastic partitions at each r, we compute $\Delta T(r)$ and identify peaks in this function (which are typically few and rather obvious). Sometimes, especially when the dataset is small, the r values which correspond to large significant changes in $T(r)$ cannot be reliably distinguished from random fluctuations. In such cases we may use a variation on cross-validation, to rule out arbitrary peaks which do not correspond to anything real in the data. The validation scheme will be described in a forthcoming paper.

3 Examples

We start with an illustrative synthetic point set example in $\mathcal{R}^2$. The information which is made available to the clustering algorithm includes only the matrix of pairwise Euclidean distances d_{ij} between points. The weights w_{ij}, which reflect similarities between points, decay with increasing distances. We use the same functional form as in [1, 7, 8], namely $w_{ij} = \exp(-d_{ij}^2/a^2)$, where a is the average distance to the n-th nearest neighbor (we used n=10, but the results are not sensitive to this choice).

Figure 1(a) shows 2000 data points in $\mathcal{R}^2$. The number of edges in the complete similarity graph is therefore 4 million. Although this number is still manageable, we observe that the vast majority of the edges have negligible weight. Note that the numerical range of w_{ij} is [0,1]. By eliminating all the edges whose weight is smaller then 0.01, we are left with a sparse graph containing only about 46000 edges, and the sparse implementation described in the appendix can be applied.



(a) (b) (c)

Figure 2. The performance of other algorithms applied on the data of Figure 1. The same exponential transformation from distances to weights was used (see text). (a) The best normalized cut [8] partition. (b) The result obtained by the factorization method [7]. (c) Deterministic single linkage. Unlike our randomized algorithm, the deterministic single linkage algorithm is sensitive to "bridges" that connect large clusters. Here, the procedure is halted manually when 3 large clusters exists, and just before two of them merge together. The desired structure is already irrevocably missing.

Good results can be obtained by using as few as 200 iterations of graph contraction. Here, in order to reduce statistical noise even more, we use 1000 iterations (0.57 sec per iteration on Pentium II 450 MHz). The resulting impulse graph of $\Delta T(r)$ (see Section 2.2) is shown in Figure 1(b).

The two obvious peaks at r=320 and r=353 mark the meaningful hierarchical levels in the data organization. At r=354 the typical cut splits the data into three large clusters (the 3 spirals) and many isolated points (the background). This is shown in Figure 1(c). At r=353 two spirals merge together, hence there is a large change in the number of edges that cross the typical cut, and a large peak in $\Delta T(r)$ appears. At r=320 the three spirals form one cluster (separated from the background points), as is identified by the first peak in $\Delta T(r)$. See Figure 2 for comparisons with other methods, all of which fail here; our algorithm clearly produces better results.

Our Second and third examples deal with the segmentation of brightness images. Each pixel of the image is a data item, and the similarity w_{ij} between pixels depends on proximity and intensity values. Again, we use the same functional form as in [7, 8], namely $w_{ij} = \exp(-d_{ij}^2/a^2) \times \exp(-dI_{ij}^2/b^2)$. The first factor measures similarity by proximity and the second factor measures similarity by intensity, where dI_{ij} is the intensity difference between pixel i and pixel j. As in [7, 8], the parameters a and b are determined manually. To reduce the number of edges and get a sparse graph, we eliminate edges whose weight is below some threshold and consider short range neighborhoods (see figure captions for details).

Figure 3(a) shows a gray level image of a baseball game. We use the same image used by Shi and Malik [8] to be able to compare the two methods. Figure 3(b) shows the graph of $\Delta T(r)$ that we obtain for this image. It is clear that there are only a few candidate solutions to consider. The peaks in this graph mark the detection of large objects in the scene. The corresponding segmentation levels are shown in Figures 3(c-f).

After forming the typical cut, some pixels (usually on object boundaries) are left isolated or form tiny clusters (say, of size < 30, which is less than one thousandth of the number of pixels in the image). It is a matter of choice whether these pixels should be assigned to one of the larger clusters or left "unlabeled". We prefer the first alternative, hence we merge each tiny cluster with its nearest large cluster according to the transformed weights, p_{ij}^r.

It appears that the results shown in Figure 3 are better that those reported in [8][2]. In particular, our algorithm segments fine details (like the palms of the hands of the lower player) while keeping the whole background as one part for a very long time; likewise, we have a level in which the circle-like object in the upper left corner is separated from the background.

Our next example follows the recent work of Perona and Freeman [7], who considered segmentation of "structured" foreground from "unstructured" background. Figure 4(a) shows a synthetic image that is generated according to the parameters reported in [7]. The significant difference between this example and the previous one is that here the background pixels are not similar to each other in terms of brightness level, and hence according to standard clustering criteria they cannot form a single group.

As shown in [7], the normalized cut algorithm [8] breaks down under these circumstances, since it is based on bi-partitions. On the other hand, our algorithm is based of multi-way cuts and has no difficulty dealing with such cases. The background pixels in this case form isolated or tiny clusters, which can be ignored, and the foreground pixels form large and significant clusters.

In Figure 4(b) we show the impulse graph $\Delta T(r)$ obtained for this example. The largest peak at r=654 signals that the typical cut computed at this r-level undergoes a significant change: at r=655 the pixels of each rectangle are grouped together, and the background pixels form tiny groups which are

[2] In addition to the fact that our method appears to work when the Ncut algorithm fails (Figure 2), the complexity of our algorithm appears to be lower: while our running time is $O(N \log^2 N)$, the complexity of Ncut is $O(MN)$, where M is the maximal number of matrix-vector multiplications allowed in the Lanczos procedure. The number M depends on many factors, and Shi and Malik observed that it is typically less then $O(\sqrt{N})$.

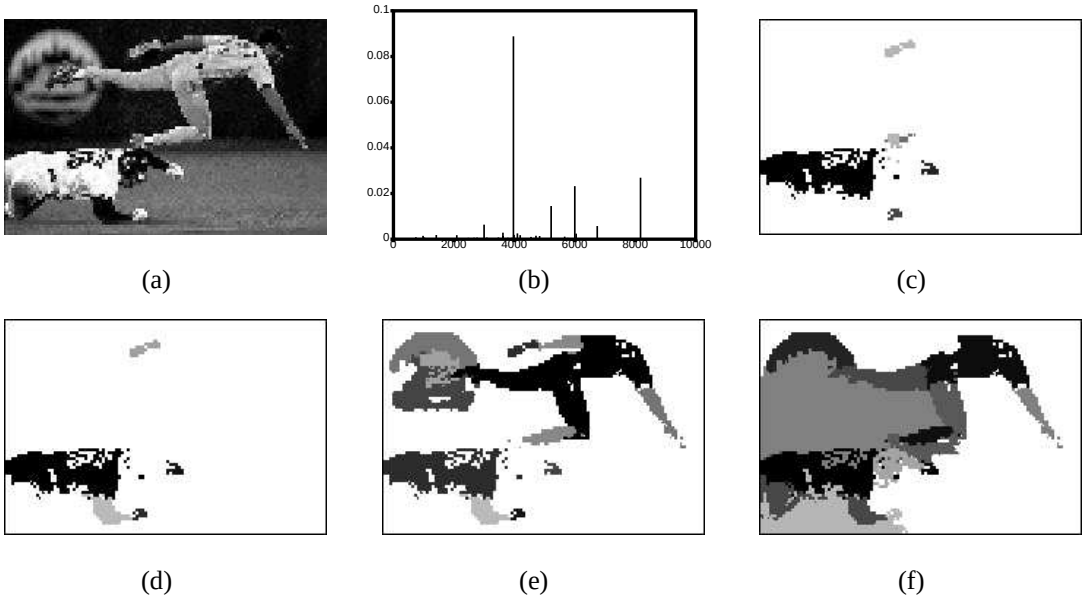


Figure 3. Brightness image segmentation. (a) The original image of size 147 X 221. (b) The graph of $\Delta T(r)$. (c)-(f) The segmentation results which correspond to the four largest peaks of $\Delta T(r)$ at r=3968, 5216, 5998 and 8174. At low r values, most of the image pixels form one giant cluster (and points on object boundaries are left isolated). The large peak at r=3968 marks the splitting of one player body from this cluster, as shown in (c). The smaller peak at r=5216 marks the splitting of the player hand as is shown in (d). The segmentation which corresponds to r=5998 is shown in (e), and perceptually it is the most desirable partition. The large peak at r=8174 is due to the splitting of the background into two parts (f). Parameter setting: Intensity range is [0,1], a=8, b=0.1, edges whose weight w_{ij} is below 0.01 were eliminated, and only the four nearest neighbors plus four random neighbors were included for each pixel.

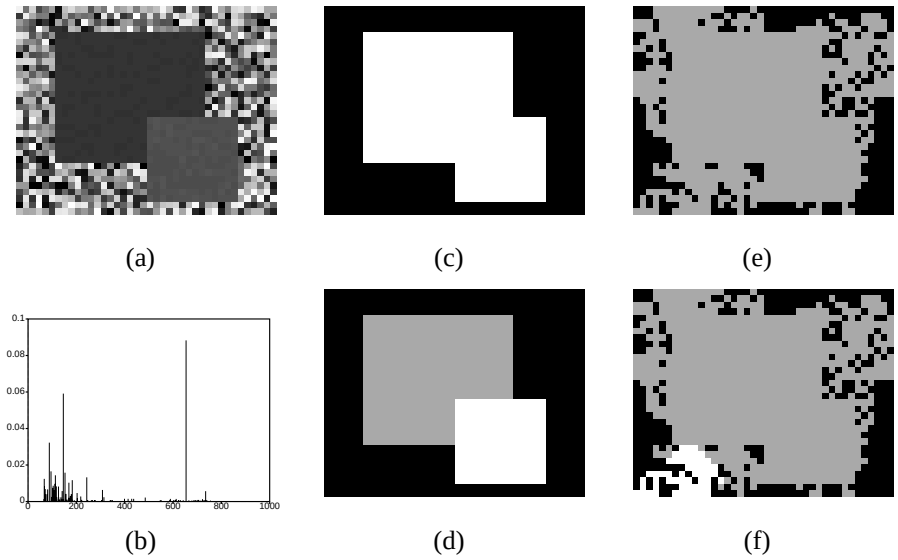


Figure 4. Separation of homogeneous objects from noisy background. (a) Synthetic 30 X 42 image, generated with the same parameters as in [7]: the brightness values are uniformly distributed between 0 and 1 for the background, between 0.2 and 0.21 for the larger rectangle, and between 0.3 and 0.31 for the smaller one. (b) the graph of $\Delta T(r)$. The largest peak appears at r=654, the second largest appears at r=146. (c,d) The segmentation results at r=654,655 that correspond to the largest peak of $\Delta T(r)$. Pixels in the Background form isolated or tiny clusters. (e,f) The segmentation results at r=146,147. These results are brought to show what kind of structure is captured by the bunch of peaks on the left of the graph in (b). Parameter setting: a=3, b=0.1 (like in [7]), only the 8 nearest neighbors of each pixel are considered.

ignored; at r=654 the two rectangles combine into one cluster. This transition is shown in Figures 4(c-d). As we go down to low r values, the clusters in the background grow and merge with the foreground set, as indicated by the bunch of peaks on the left of the graph. The highest one among them is at r=146, which corresponds to the transition from (e) to (f) in Figure 4. We expect that these peaks would be ruled out by a cross validation technique that we have recently developed.

Our results seem better than those reported by in [7]. In addition, our algorithm works in cases where the factorization method fails, as is demonstrated in Figure 2. Another example in which our method works and the factorization method fails includes two concentric rings of points in the plane. In general, the factorization method is not suitable for chained data, since it looks for block structure in the similarity matrix.

Appendix: algorithms and complexity of cut sampling

Since the number of cuts in a graph is exponentially large, one must ask whether the probability distribution over cuts is computable. Here the decaying rate of the cut probability plays an essential role. The induced probability is found to decay fast enough with the capacity, hence the distribution of cuts is dominated by the low capacity cuts. There exists a polynomial bound on the number of low capacity cuts in any graph [6], hence the problem becomes computable since a sample of polynomial size is sufficient to estimate the p_{ij}^r's.

The sampling tool that we use is the "contraction algorithm" [6] whose basic idea is outlined in Section 2.1.1. Its discovery led to an efficient probabilistic algorithm which finds the minimal cut, as the probability of the contraction algorithm to return the minimal r-way cut of any graph is at least $N^{-2(r-1)}$. Moreover, the probability of every cut to survive a contraction from N to r nodes is equal to the probability of not selecting an edge which cross this cut. Hence, the probability to return any cut of the graph decays with increasing capacity.

The procedure which is outlined in Section 2.1.1 involves the selection of an edge (i,j) with probability proportional to w_{ij}. As shown in [6], this can be done in $O(N)$ time by a two step procedure. One keeps for every node i its (generalized) degree, i.e., the sum of weights of the edges incident on i, and one chooses at the first step a node i_0 out of N, with probability proportional to the node degree. At the second step the second node j is chosen, with probability proportional to $w_{i_0 j}$. Hence every step is a selection of one item out of N, which takes $O(N)$ time. Since a single graph contraction repeats the edge selection N times, the resulting complexity is $O(N^2)$.

The similarity graphs which appear in clustering problems, and in particular in image segmentation applications, are naturally very sparse. For image segmentation, it is reasonable to consider for each pixel only the neighboring pixels. We therefore develop a contraction scheme for sparse graphs that runs in $O(e \log e)$ time. Here e is the number of edges, which is $O(N)$ in sparse graphs.

We construct a binary tree whose e leaves represent the edges of the sparse graph. Each leaf contains the weight of the corresponding edge, and each inner node contains the sum of weights of its two sons. A selection of an edge is implemented by propagation from the root to one of the leaves, where at each inner node a probabilistic decision is made regarding the next node (the probability to select a son is proportional to its weight). Hence, the complexity of selecting an edge is $O(\log e)$, and since a single graph contraction involves N edge selections, the total complexity of selections is $O(N \log e)$.

The selected edge marks which two connected components are merged together. We find all the edges that connect these two components (see below), and starting from each leaf that represents one of them we propagate toward the root, subtracting the weight of the leaf from each node we path through. Since eventually every edge is reached and eliminated, a single graph contraction involves e such updates, each one of complexity $O(\log e)$. The total complexity of maintaining the data structure is therefore $O(e \log e)$, providing that we have an efficient way to find the edges which connect the two components being merged together.

Finding the edges which connect two components in the graph involves two stages. The first one is to identify the two components which are merged together after an edge (i,j) was selected. This is done using the same technique used by the Union-Find algorithm that partitions a set of items into equivalence classes. It can be shown that the total complexity of the $2N$ Find operations is $O(N\alpha(N))$, where $\alpha(N)$ is the inverse Ackermann's function, $\alpha(N) \leq 4$ for all integers N one is ever likely to encounter.

The second stage consists of finding the other edges which connect the two identified components. One possible way is to query all the edges which incident on the smaller component, checking whether their other vertex is in the larger component. For each vertex in the smaller component we may present this query no more than $O(\log N)$ times, since this is the maximal number of times that a vertex can be in the smaller one of two merged components. The total complexity of querying during one graph contraction is therefore bounded by $O(N \log N)$. We note that in our implementation we maintain a data structure which supports direct indexing for the connecting edges, but its description is beyond our scope here.

[1] Blatt M., Wiseman S. and Domany E., "Data clustering using a model granular magnet", Neural Computation 9, 1805-1842, 1997.

[2] Duda O. and Hart E., *"Pattern classification and scene analysis"*, Wiley-Interscience, New York, 1973.

[3] Gdalyahu Y., Weinshall D. and Werman M., "A Randomized Algorithm for Pairwise Clustering", Proc. NIPS, 1998.

[4] Hofmann T. and Buhmann J., "Pairwise data clustering by deterministic annealing", PAMI 19, 1-14, 1997.

[5] Jain A. and Dubes R., *"Algorithms for clustering data"*, Prentice Hall, NJ, 1988.

[6] Karger D., "A new approach to the minimum cut problem", Journal of the ACM, 43(4) 1996.

[7] Perona P. and Freeman, W., "A factorization approach to grouping", Proc. ECCV, 1998.

[8] Shi J. and Malik J., "Normalized cuts and image segmentation", Proc. CVPR, 731-737, 1997.

[9] Wu Z. and Leahy R., "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation", PAMI 15, 1101-1113, 1993.