Self organization in vision: stochastic clustering for image segmentation, perceptual grouping, and image database organization

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, and show how it can be used to address various problems in computer vision, including the low-level image segmentation, mid-level perceptual grouping, and high-level image database organization. The clustering problem is viewed as a graph partitioning problem, where nodes represent data elements and the weights of the edges represent pairwise similarities. We generate samples of cuts in this graph, by using Karger's contraction algorithm, and compute an "average" cut which provides the basis for our solution to the clustering problem. The stochastic nature of our method makes it robust against noise, including accidental edges and small spurious clusters. The complexity of our algorithm is very low: $O(|E| \log^2 N)$ for N objects, $|E|$ similarity relations and a fixed accuracy level. In addition, and without additional computational cost, our algorithm provides a hierarchy of nested partitions. We demonstrate the superiority of our method for image segmentation on a few synthetic and real images, B&W and color. Our other examples include the concatenation of edges in a cluttered scene (perceptual grouping), and the organization of an image database for the purpose of multi-view *3D* object recognition.

1 Introduction

A wide range of tasks in computer vision may be viewed as unsupervised partitioning of data. Image segmentation, grouping of edge elements and image database organization, are problems at different levels of visual information processing (low, middle and high level vision, respectively). These tasks have different application objectives, and they handle very different data entities (pixels, edgels, images). Nevertheless, they all come to serve a common goal, which is the partitioning of the visual entities into "coherent" parts.

"Good" data partitioning, or *clustering*, is a vague concept – there is no universal measure for the coherence of a cluster. Intuitively, coherence should reflect intra-cluster homogeneity and inter-cluster separability. Objects in a homogeneous cluster resemble each other more than they resemble objects in other clusters. By grouping objects into parts, or by organizing the data in a hierarchy of parts, cluster analysis attempts to recover the inherent structure within a given dataset. Data clustering is a fundamental problem in unsupervised learning, with a strong connection to cognitive science [29, 40, 36].

The goal of this work is to use cluster analysis as a unifying principle for a wide range of problems from low, middle and high level vision. In our approach we distinguish between two stages of processing. The first stage is task dependent, and defines the affinity, or similarity, between the visual entities. The affinity is a function of the relevant attributes. Low level attributes might be the spatial location, intensity level, color composition or filter response of a pixel in the image. Mid level attributes, in the case of edge elements, may be spatial location, orientation or curvature, and the affinity associated with them may reflect properties such as proximity, symmetry, co-circularity and good continuity. High level attributes may be the entire shape of an object in the scene or the color distribution of all the pixels in an image.

The second stage in this approach follows the unifying principle and applies cluster analysis to organize visual objects (pixels, edgels, images) into coherent groups. These groups reflect internal structure among the entities, where (roughly speaking) the affinity within groups is larger than the affinity between groups. Therefore, a cluster of pixels in the image, sharing similar locations and colors, is expected to account for an object or a part of an object in the scene. A cluster of edge elements is expected to exhibit a meaningful aggregation into a complete edge, and a cluster of images in a database is expected to be related by a common topic.

In this work we present a novel stochastic pairwise clustering algorithm. Our algorithm is hierarchical, efficient, robust and model-free. The robustness of our method is achieved by averaging over all the possible interpretations of the data, giving more weight to data partitions that are associated with lower cost. This idea is adopted from clustering algorithms that are inspired by statistical mechanics, and in particular our method is related to [1]. Like the algorithms that apply stochastic simulation of a certain dynamics, we generate a sample of configurations, and count the number of times that two nodes are in the same cluster. This leads to the definition of a powerful collective similarity measure, which is the pairing probability of every two nodes. Our algorithm can be analyzed analytically, and for sparse graphs and a fixed accuracy level it runs in $O(N \log^2 N)$ time, where N is the number of data-points.

The rest of this paper is organized as follows. The problems of image segmentation, perceptual grouping of edge elements and image database organization are reviewed in Section 1.1, while some relevant literature on pairwise clustering is reviewed in Section 1.2. The novel clustering algorithm is described in Section 2, with additional details provided in Section 4. Our results are summarized in Section 3; a concluding discussion follows in Section 5.

1.1 Segmentation, perceptual grouping, and image retrieval

This section puts our framework in the context of other computer vision work. Classical techniques for image segmentation include gray level thresholding, region growing and recursive splitting (henceforth to be called split-and-merge), relaxation by the Markov Random Field approach or by neural networks, and more [13, 26, 27]. Recently, several authors considered image segmentation as a direct application of pairwise clustering. This includes the use of objective functions which are minimized by graph theoretic methods [43, 37, 20], deterministic annealing [17], multi-grid techniques [33], and other methods [2]. Orthogonal to these approaches, which try to minimize a certain cost function, there are methods which are built upon local and greedy aggregation [8, 3] or upon spectral decomposition of the similarity matrix [28, 41].

The phrase perceptual grouping is usually associated with mid level vision, and more specifically with the grouping of edge elements. Classical methods for perceptual grouping of edge elements frequently assign a saliency value to each edge, representing to what extent it is part of a salient shape or background noise [35]. The problem is also treated using stochastic completion fields [42, 38], cost minimization [16] and spectral decomposition of the affinity matrix [31, 15].

A measure of similarity between images is essential for organization and retrieval of images. One Possibility is to use a global measure, such as the color histogram of the entire image. In our work we use the shape of extracted objects; we extend existing methods by using our own shape dissimilarity measure [10], which enables the organization of an image database into shape categories. One application of such an organization is to present the content of the database in a concise form, which can be digested by the user. In image retrieval, given a query image, we will use the database organization to associate the query with the cluster of most similar images.

Clustering of pixels, edge elements and images is associated with different levels of image understanding. It is important to distinguish our goal from the closely related vector quantization approach, where a concise representation of the data is sought, regardless of the actual meaning and significance of the clusters. Vector quantization is concerned with information encoding by means of a finite size codebook, usually for the purpose of compressed archiving or transmission. Although the objective of understanding and compression is different, they are tightly related since compact description of the data reflects abstraction, which is a form of understanding.

1.2 Pairwise (graph based) clustering: review

We next review some clustering methods that are formulated as graph partitioning [6, 21]. Here the data is represented by a weighted (and perhaps incomplete) graph, including a set of nodes representing data items, a set of edges connecting them, and a function that assigns weight to each edge representing the affinity between the adjacent data items.

The agglomeration heuristics for pairwise clustering start from 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 that are most similar are merged together, until the similarity of the closest clusters is lower than some threshold. In graph terminology, an agglomeration step is called contraction, where two nodes of the graph are unified. 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 very general class of agglomerative algorithms is formalized in [25].

A few partitioning criteria are based on the notion of cuts in a graph, where the capacity of the cut is the sum of weights of all edges that cross the cut. The *minimal cut clustering algorithm* seeks the partition which minimizes this cost. For bi-partitions, which split the data into two parts, this optimization problem is extensively studied and solved in polynomial time. However, clustering applications frequently involve very large graphs, and the exact solution becomes impractical. In our proposed clustering algorithm we apply a new sampling tool, originally developed as the core of a probabilistic minimal cut algorithm [22]. This tool is known as the contraction algorithm, and in fact it is a randomized version of the single linkage method for clustering. Hence, in addition to improving efficiency, we make a formal connection between cut based algorithms and agglomerative algorithms.

Spectral methods identify good partitions via the eigenvectors of the similarity matrix, or other matrices derived from it. The similarity matrix is the matrix whose (i, j) element contains the weight of the directed edge from node i to node j. In general spectral methods are not associated with any global cost function and can be thought of as useful heuristics. The factorization method [28] uses the principal eigenvector v of the similarity matrix, treating it as an indicator function: a threshold θ is chosen, after which each node i is assigned to one part if $v[i] > \theta$ and to the other part otherwise; this method was applied to figure/ground segmentation and perceptual grouping. Similar applications to perceptual grouping and hypertext retrieval were proposed earlier in [31, 24]. See [9] for the relation with dynamical systems and stochastic graph traversal.

The use of non principal eigenvectors can sometimes add more power to the algorithm. Assuming that the similarity matrix A is symmetric, let V denote the $n \times k$ matrix whose columns are the k largest eigenvectors of A. As shown in [41], the algorithm described in [5] effectively uses the matrix $Q = VV^T$ for motion grouping. If $\hat{V}$ denotes the matrix V after its rows are normalized to length 1, then [32] defines $\hat{Q} = \hat{V}\hat{V}^T$; for "well separated" data and with a proper choice of k, it is claimed that the value of $\hat{Q}[i, j]$ is close to either 0 or 1, depending on whether the nodes i and j belong to the same cluster or not.

A related method, applied so far to image segmentation, texture segmentation and motion segmentation, is the normalized cut algorithm [37]. Given a symmetric similarity matrix A, the normalized cut algorithm uses the eigenvectors of its Laplacian matrix; in particular, the eigenvector corresponding to the second smallest eigenvalue is used to partition the graph into two parts via thresholding. Unlike the other heuristically spectral methods, this algorithm can be shown to provide a continuous approximation to a discrete optimization problem [4]; it minimizes a specific partitioning criterion, the capacity of the bi-partition normalized by the association of the two parts.

2 The typical cut algorithm

This section presents the general approach and the principles of our clustering method; a full description and analysis is postponed to Section 4, while the code itself is available from [44]. After defining the terminology in Section 2.1, we describe the algorithm in Section 2.2. To clarify how the algorithm works and to demonstrate some of its properties, an illustrative example is discussed in details in Section 2.3. Issues of complexity, pre-processing and related work are discussed in Sections 2.4, 2.5 and 2.6.

2.1 Notations and Definitions

Our clustering algorithm uses pairwise similarities, which are represented as a weighted graph $G(V, E)$: the nodes V represent data items, and the positive weight w_{ij} of an edge (i, j) represents the similarity between nodes (data items) i and j. The graph $G(V, E)$ may be incomplete, due to missing data or due to edge dilution (whose purpose is to increase efficiency). The weights w_{ij} may violate metric properties, and in general they may reflect either similarity or dissimilarity values. In the current work, however, we assume that the weights reflect symmetric similarity relations (hence $w_{ij}=w_{ji}$, and $w_{ij}=0$ for i and j that are completely dissimilar). We do not assume that the similarity weights obey the triangular inequality, and self similarities w_{ii} are not defined.

A cut (V_1, V_2) in a graph $G(V, E)$ is a partition of V into two disjoint sets V_1 and V_2. The capacity of the cut is the sum of weights of all edges that cross the cut, namely: $c(V_1, V_2) = \sum_{i \in V_1, j \in V_2} w_{ij}$. A *minimal cut* has the minimal capacity. We use the term "cut" also for the generalized case of multi-way cuts. A partition of V into r disjoint sets $(V_1, \ldots, V_r)$ is called r-way cut, and in accordance its capacity is defined as $\sum_{i \in V_\alpha, j \in V_\beta, \alpha \neq \beta} w_{ij}$. Every one of the r components may be referred to as a "side" of the cut.

Let the nodes which belong to each side V_α $(\alpha = 1 \ldots r)$ be grouped together into one *meta-node*, and discard all the edges which form self loops within meta-nodes (namely, discard the inner edges of each component, which connect two inner nodes belonging to the same component). The graph which is thus obtained has exactly r meta-nodes, and it is a multi-graph since meta-nodes may be connected to each other by more than one edge. Actually, if G is a complete graph, then the number of edges connecting the meta-nodes representing the components V_α and V_β is exactly $|V_\alpha||V_\beta|$.

The grouping procedure described above yields a *contracted graph* which has r meta-nodes, denoted G'_r. Note that this notation does not characterize the contracted graph, since there are many ways to group the nodes of G into r disjoint sets. However, any contracted graph G'_r represents an r-way cut in the original graph. The edges of G'_r are the edges which cross the corresponding r-way cut in the original graph.

2.2 Outline of algorithm

This section provides a simplified concise description of the algorithm, ignoring some of the implementation issues arising from space and time complexity considerations. The algorithm is divided into two stages, described in pseudo-code in Figures 1,2 and explained below.

Generating typical cuts: For a given value of r $(r = 1 \ldots N)$ our algorithm generates a sample of M possible r-way cuts, and uses this sample to estimate the probability p^r_{ij} that (i, j) is not a crossing edge of a random r-way cut. The pseudo-code in Figure 1 counts, for every pair of nodes $i, j \in V$ and for every integer r between 1 and N, the number of r-way cuts (out of M) in which the two nodes are on the same side. These accumulators are divided by M to estimate for every two nodes the probability p^r_{ij} that they are on the same side.

In this pseudo-code the procedure `CONTRACT` generates the r-way cut G'_r from the previously generated cut G'_{r+1}. The procedure `CONTRACT` selects two meta-nodes of G'_{r+1} and merges them into one meta-node of

```
procedure STAGE-1:
input:   weighted graph $G(V,E)$ with $N$ nodes.
output:  3D array $p$ of probabilities.

$s^r_{ij}$ ← 0 for $i,j,r = 1 \dots N$   (initialize counters)
for $m = 1 \dots M$:
   $G'_N \leftarrow G(V,E)$
   for $r = (N-1) \dots 1$:
      $G'_r$ ← CONTRACT($G'_{r+1}$)   (generate an $r$-way cut)
      for $i = 1 \dots N$:
         for $j = 1 \dots N$:
            if $i$ and $j$ belong to the same meta-node of $G'_r$, then
               $s^r_{ij}$ ← $s^r_{ij}$+1
            end-if
         end-loop
      end-loop
   end-loop
end-loop
$p^r_{ij}$ ← $s^r_{ij}/M$ for $i,j,r = 1 \dots N$   (compute empirical probabilities)
return array $p$.
```

Figure 1: Pseudo-code which transforms similarity weights into pairing probabilities.

G'_r, while discarding the edges which previously connected these two meta-nodes. The selection of the nodes to be unified is probabilistic: an edge (i,j) of G'_{r+1} is selected for contraction with probability proportional to its weight w_{ij}. Then, the two meta-nodes which are adjacent to the selected edge are merged.

The contraction procedure is the cornerstone of our method, since it defines the sample of M cuts, according to which the empirical probabilities p^r_{ij} are computed. Thus the contraction procedure is our sampling tool, typically assigning higher probability to cuts with lower capacity as is shown in [22] . In fact, [22] proves that the minimal cut can be found using this sampling method in polynomial time, even though the overall number of possible cuts is exponential. In summary, the contraction process induces a probability distribution over cuts, and under this distribution we estimate p^r_{ij} – the marginal probability that nodes i and j are on the same side of a random r-way cut.

The number of r-way cuts in a graph of N nodes is the Sterling number of the second kind, denoted $\tau(r,N)$. Let $\alpha(r) = 1 \dots \tau(r,N)$ be an index to the set of all r-way cuts in $G(V,E)$. Fix r and let P_α denote the probability that the contraction algorithm generates the cut α. For a fixed r value, $\sum_\alpha P_\alpha = 1$. Define an indicator variable e^α_{ij} to be 1 if the edge (i,j) crosses the cut α and 0 otherwise. It is readily seen that

$$\sum_{(i,j)\in E} w_{ij}(1-p^r_{ij}) = \sum_{(i,j)\in E} w_{ij} \sum_\alpha e^\alpha_{ij} P_\alpha = \sum_\alpha c_\alpha P_\alpha = \langle c(r) \rangle$$

where c_α is the capacity of cut α, and $\langle c(r) \rangle$ is the expected value of the r-way capacity. We can therefore interpret 1-p^r_{ij} as the probability that edge (i,j) is a crossing edge in an "average cut". We use this observation for the following definition.

For every integer r between 1 and N we define the *typical cut* $(A_1, A_2, \dots, A_{s(r)})$ as the partition of G into connected components, such that for every $i \in A_\alpha$, $j \in A_\beta$ $(\alpha \neq \beta, \alpha, \beta = 1 \dots s(r))$ 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, and we then compute the connected components in the remaining graph. Note that the number of parts, $s(r)$, in the typical cut can be different from r.

The N typical cuts corresponding to $r = 1 \dots N$ are the candidate solutions to our clustering problem. Although this is an extremely small number compared with the exponential number of possible partitions, we still need to select only a few interesting solutions out of the N candidates. 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.

Selecting meaningful partitions: We define the following function of the typical cut at level r:

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

where $N_k = |A_k|$ denotes the number of elements in the k-th cluster. $T(r)$, therefore, measures how many edges of the complete graph 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 very 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 large meta-nodes are merged.

We look at changes in the value of $T(r)$ between subsequent r values, and output only those partitions which are associated with a large change in $T(r)$. For the current presentation we set a threshold δ, and output a solution at level r if and only if $\Delta T(r) > \delta$. This is described in Figure 2.

```
procedure STAGE-2:
input:   3D array of the probabilities $p_{ij}^r$
output:  hierarchy of a few selected partitions.

for $r = 1 \ldots N$:
   let $G(V,E)$ be a complete weighed graph of $N$ nodes
   and assign weight $p_{ij}^r$ to each edge $(i,j) \in E$.
   for each $(i,j) \in E$:
      if $p_{ij}^r < 0.5$ then
         $E \leftarrow E \backslash (i,j)$  (remove the edge from E)
      end-if
   end-loop
   find connected components $(A_1, A_2, \ldots, A_s)$ in $G(V,E)$.
   compute $\Delta T(r) = T(r) - T(r-1)$  (use Equation (1))
   if $\Delta T(r) > \delta$ then
      (*) relabel small parts (see text)
      report partition $(A_1, A_2, \ldots, A_s)$.
   end-if
end-loop
```

Figure 2: Pseudo-code which finds typical cuts, measures the resemblance between subsequent cuts, and reports the "meaningful" partitions.

The existence of pronounced peaks in $\Delta T(r)$ does not guarantee that we find the desired solutions. A necessary requirement is that the set of N candidate typical cuts, that correspond to the N possible r values, *indeed contains* the desired solutions. Whether this is the case or not, it depends on the output of the first stage of the algorithm, the core of our method. The simple heuristic that is applied in the second stage (Figure 2) is based on the assumption that a good set of candidates is given.

The code line denoted with (*) in Figure 2 is optional, and involves an additional parameter. One may not be interested in a cluster whose size is very small, e.g., 1% of the number of data points. Small clusters are formed either by boundary points, due to the competition between conflicting labels, or by background points (unstructured noise) due to their isolation. A conservative strategy regards the points clustered in

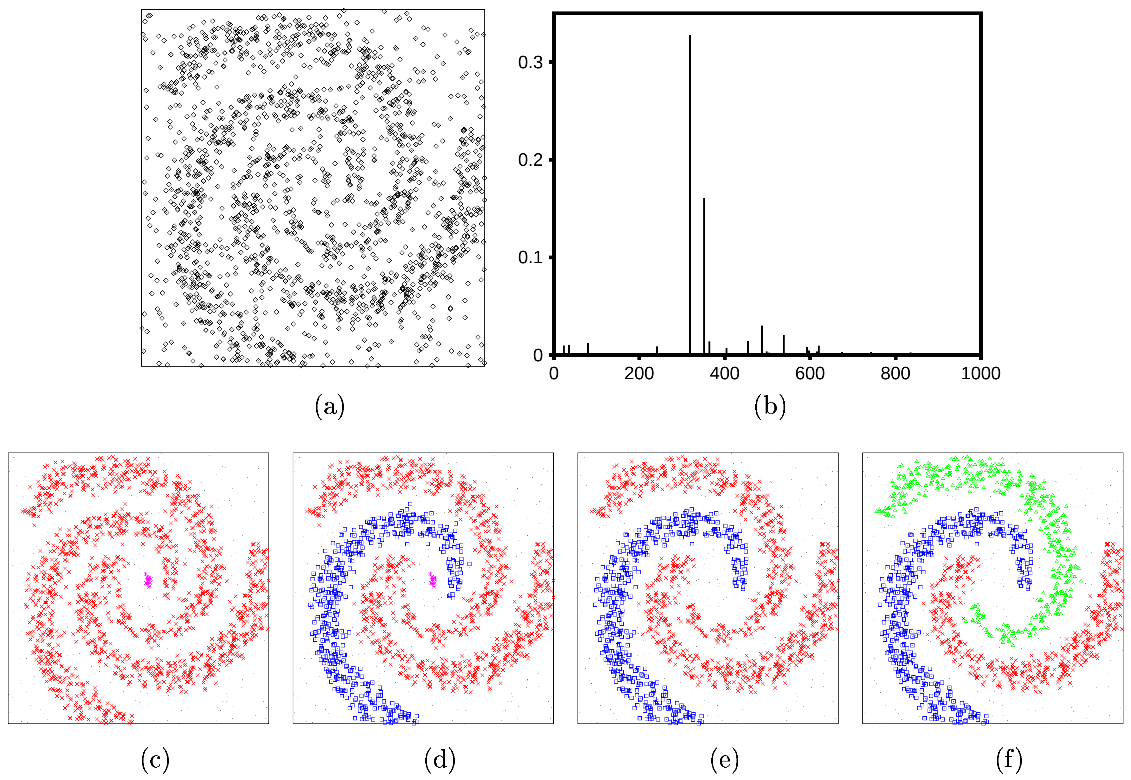

Figure 3: Clustering of points in the Euclidean plane. (a) The 2000 data points; the coordinates of the points are not available to the clustering algorithm, which uses only the matrix of pairwise distances. (b) The graph of $\Delta T(r)$, computed for *every* integer r between 1 and N. Two peaks are clearly observed at r=319 and r=352. (c-f) From left to right: the typical cuts at r=318, 319, 351 and 352. Different components are indicated by different colors and symbols, while isolated points (clusters of size one) are marked by small black dots.

very small parts as points whose labels cannot be safely determined. An aggressive strategy, on the other hand, may sustain the larger parts but relabel the smaller parts by attempting to recover their original label. We use such a relabeling strategy which revives some of the deleted connections, adding back edges in decreasing order of p^r_{ij}, thus letting the small clusters join the larger ones. More details on the optional relabeling procedure are given in Section 4.1.3.

2.3 Illustrative example

To illustrate some features of our method and its relative advantages over other methods, we proceed with an illustrative synthetic example. We use a point set example in two dimensions which can be easily visualized, see Figure 3a. The data consists of N=2000 points in $\mathcal{R}^2$ arranged in three dense spiral regions and sparse background. The vectorial nature of the data in this example, namely points in $\mathcal{R}^2$, is used for visualization but is hidden from the clustering algorithm. The information which is made available to the clustering algorithm includes only the matrix of pairwise Euclidean distances d_{ij} between points.

During preprocessing, the Euclidean distances d_{ij} are transformed to similarity weights, which decrease with increasing distances. Psychophysical studies find that the similarity, or the confusion frequency, between stimuli decay exponentially with some power of the perceptual measure of distance [36, 7]. We use the same functional form as in [1, 28, 37], namely, $w_{ij} = \exp(-d^2_{ij}/a^2)$, where a is the average distance to the n-th nearest neighbor (we used n=10, but our results are not very sensitive to this choice; see Section 4.2). We then construct a complete graph $G(V, E)$ with N nodes, where node i represents the i-th data point and the weight of edge (i, j) is set to w_{ij}.

The constant a reflects some reasonable local scale. If $d_{ij} \gg a$ then w_{ij} is very small and the edge (i, j)

is unlikely to be selected by the procedure CONTRACT. In this case the decision whether points i and j are in the same cluster depends solely on transitive relations via other nodes, same as when w_{ij} is unknown. Thus, in order to increase the computation efficiency, we delete from the graph every edge (i, j) whose weight is negligible. In the current example we discarded edges with weights smaller than 0.01 (note that the weights are in the range $[0, 1]$). This threshold and the number n=10 (which determines a) are the only parameters involved in the preprocessing stage. The number of remaining edges in this example was about 46000.

Having constructed the similarity graph $G(V, E)$, we are ready to apply our clustering algorithm. The first stage is to estimate the pairing probabilities p^r_{ij}. This is done using a sample of M partitions at each r level (see procedure STAGE-1 in Figure 1). In the second stage of our algorithm we call the procedure STAGE-2 (Figure 2), which computes the typical cut for each r between 1 and N. To select between the N candidate partitions, procedure STAGE-2 looks for large changes in the function $T(r)$ defined above. Figure 3b shows the graph of $\Delta T(r) = T(r) - T(r-1)$ as a function of r. It is an impulse graph, illustrating that partitions which correspond to large changes in $T(r)$ are few and easy to identify. Two obvious peaks appear at r=319 and r=352, and they mark the meaningful levels of data organization in this example.

The two peaks in $\Delta T(r)$ correspond to 3 hierarchical levels of partitions. At r=318, just before the large peak on the left, the three spirals form one cluster (Figure 3c). There is also a small cluster containing 10 points at the center, and the rest of the points form isolated clusters of size one. This is the coarsest level of interest. In the next interesting level, at r=319, the giant cluster splits into two - one part contains a single spiral and the other part contains two spirals (Figure 3d). Increasing r further reveals some boundary effects, where 41 additional points become isolated, but the overall picture remains unchanged as long as r <352 (Figure 3e). The peak at r=352 signals the next significant change, giving the partition shown in Figure 3f. The meaningful partitions are reported by the procedure STAGE-2 when its parameter δ is set to 0.1 (Figure 3b shows that our method is not very sensitive to the exact value).

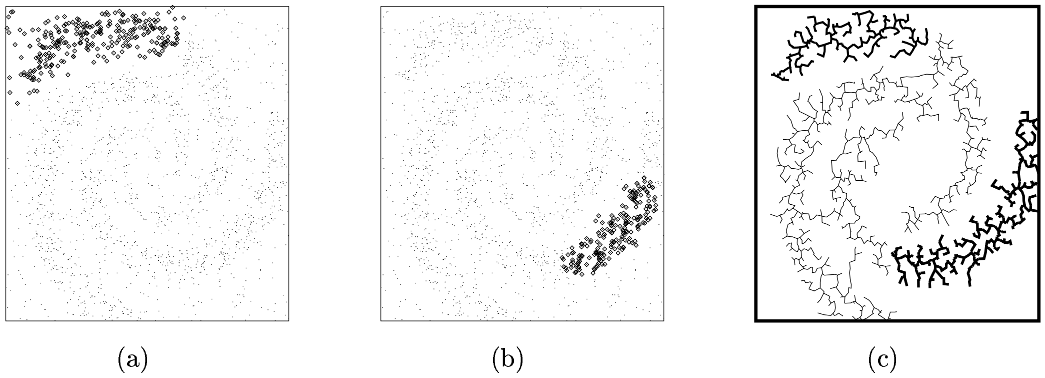

Figure 4: The results of other algorithms applied to the data of Figure 3. The same preprocessing and exponential transformation from distances to weights was used (see text). (a) The best normalized cut [37] partition. (b) The result obtained by the factorization method [28]. (c) Deterministic single 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.

For comparison, we apply to the same data exactly (after identical preprocessing) a few other methods. The results with two spectral methods are shown in Figure 4(a,b); clearly these methods fail here, while our algorithm produces good results. Likewise, the deterministic single-linkage algorithm fails to find the desired solution in any of its hierarchical levels (rendering irrelevant the issue of selecting a stopping criterion), as is illustrated in Figure 4c.

2.4 Complexity: the sample size

The detailed complexity analysis is postponed to Section 4.1, where we describe our efficient implementation. The efficient version drastically decreases the number of estimated variables (p^r_{ij}) from N^3 to $|E|$, which is

$O(N)$ for sparse graphs and $O(N^2)$ for general graphs. Moreover, it will be shown there that one graph contraction (one iteration of the external loop in STAGE-1, Figure 1) can be implemented in $O(|E|\log N)$ time.

In this section we address the question of the desired sample size, denoted M. We show that $M = O(\log N/\epsilon^2)$ for an accuracy level ϵ, hence the overall complexity of STAGE-1 for a fixed accuracy level is $O(|E|\log^2 N)$. The efficient implementation of STAGE-2 takes only $O(|E|\log N)$ time (Section 4.1.3), making $O(|E|\log^2 N)$ the overall complexity bound for our algorithm.

In practice, one may monitor the convergence of the estimated quantities during execution, and terminate when a sufficient level of accuracy is obtained. However, to determine the asymptotic complexity of the algorithm an estimate of M is required. Our goal is to determine a lower bound on M which guarantees, with a sufficient level of certainty, a sufficient amount of accuracy. We denote by p^r_{ij} the correct pairing probabilities, which are obtained at the limit of infinite M. We define an accuracy parameter ϵ and an uncertainty parameter δ, and require our empirical estimations to deviate from p^r_{ij} by no more than ϵ, with probability 1-δ.

Since the lower bound which we will find for M will not depend on r, we pretend that r is fixed to some arbitrary value r_0, which is omitted from the notations. Hence the notation p_{ij} stands for p^r_{ij} and $r = r_0$. For simplicity, we also prefer to enumerate the edges of the graph instead of its nodes. Hence the notation p_e stands for the pairing probability of the nodes that are adjacent to edge e. Equivalently, p_e is the probability that edge number e is an inner edge of a meta node when $r = r_0$. Procedure STAGE-1 of Section 2.2 counts the number of times (out of M) in which an edge e is an inner edge of a meta node. If this is the situation S_e times, then our empirical estimation for p_e is $\hat{p}_e = S_e/M$.

Let X_i be the sequence of M Bernoulli trials such that $X_i = 1$ for a round when edge e is an inner edge, and $X_i = 0$ otherwise. Thus $Pr[X_i = 1] = p_e$ and $Pr[X_i = 0] = 1 - p_e$. Let $\hat{p}_e = (\sum X_i)/M$ be our empirical estimation. The Hoeffding-Chernoff bound [23] implies that for $0 \le \epsilon \le 1$:

$$Pr[\ |\hat{p}_e - p_e| > \epsilon\ ] \le 2e^{-2M\epsilon^2}$$

Thus we have a bound on the probability of making a too large error in estimating p_e for one edge. But there are $|E|$ edges in the graph, and we need to ensure that we do not make a large error for too many of them. We thus use the union bound with our certainty parameter δ:

$$\begin{array}{rcl} Pr[\ \exists e\ \ |\hat{p}_e - p_e| > \epsilon\ ] & \le & \\ \sum_e Pr[\ |\hat{p}_e - p_e| > \epsilon\ ] & < & \\ |E| \cdot 2e^{-2M\epsilon^2} & < & \delta \end{array}$$

which results in

$$M > 0.35 \frac{\log_2 |E| - \log_2 \delta + 1}{\epsilon^2}$$

The asymptotic dependence of the sample size on the number of nodes is therefore $M = O(\log N)$, even for complete graphs where $|E| = O(N^2)$. The efficient computation of the half probability level (Section 4.1.1) introduces an additional error, since it involves an approximate median computation. However, this affects the parameter ϵ but not the logarithmic dependence on the number of nodes.

2.5 Pre-processing and other parameters

The main part of the algorithm, STAGE-1, depends only on the accuracy parameter, which affects only the number of iterations M. Hence no parameter tuning is required at this stage [1]. The second part, STAGE-2,

[1]A parameter which is fixed in our algorithm, but in principle could be treated as a free one, is the threshold (0.5) in the definition of the typical cut. However, the algorithm is not sensitive to the exact value of the threshold. In fact, the p^r_{ij} values tend to be either close to 0 or to 1 [1], which is the reason for the abrupt transitions that we detect in STAGE-2. Under small changes of the threshold value the selected r levels may change slightly, but the partitions found are hardly affected.

involves the parameter δ that determines whether a change in the function $T(r)$ is significant. Our examples throughout this paper always show the complete graph of $\Delta T(r)$, demonstrating that significant changes are pronounced and easy to recognize. The minimal cluster size of interest, if it is known, is another parameter which may be used by `STAGE-2`. Only the *interpretation* of the results depends on this parameter, thus reflecting prior information. For example, in Figure 3, had we used a minimal size parameter larger than 10, the small cluster at the center would not have been shown, and its members would have been left unlabeled.

It is mostly the preprocessing stage, which constructs the weighted graph G, that depends on external parameters. These parameters are related to the definition of similarity (which is task dependent), and to the transformation from perceptual similarity to edge weight. Given a dissimilarity measure d_{ij} between stimuli i and j, we define $w_{ij} = \exp(-d_{ij}^2/a^2)$. Here a is a decay parameter which reflects some suitable local scale, and it needs to be tuned. Sometimes the dissimilarity between stimuli is measured along different dimensions, like in an image segmentation task where dissimilarity between pixels is a function of their spatial proximity and relative brightness. In this case a different local scale parameter is defined for every dimension μ of similarity, namely:

$$w_{ij} = \prod_{\mu} \exp(-d(\mu)_{ij}^2/a(\mu)^2) \tag{2}$$

Exponential decay is supported psychophysically [36, 7] and can be understood from the properties of the cut cost function that we use. The cut value, which is the total similarity weight between clusters, is affected by the size of the clusters. Large clusters should not be judged similar just because there are many graph edges connecting them. Exponential decay, as well as local neighborhoods, avoid this undesirable bias. In [39] the weights are treated as transition probabilities, and an exponential decay is used to relate between the length of a trajectory (which is additive, like the cut capacity) to its probability (which is multiplicative for independent events).

Additional experiments described in [9] show that the results reported above are robust with respect to the scale parameters used to implement Equation (2). We had to change the neighborhood size used to determine a by orders of magnitude to have any noticeable effect. Only very drastic pruning of "small" edges, leading to rather small graphs where most of the information is lost, affected the final result noticeably.

2.6 Our work put in context

We now wish to put the contraction algorithm in the context of agglomerative clustering. As explained above, agglomerative clustering is a family of algorithms which partition the data using a greedy local merging criterion. Thus an agglomerative procedure starts with the trivial partition of N points into N clusters of size one, and proceeds with $N-1$ merging steps, where at each step two clusters are selected and merged together. The selection is based on a local definition of clusters similarity, and at each step the two clusters which are most similar are chosen and merged. In particular, the single linkage algorithm defines clusters similarity as the maximal pairwise similarity between their members.

The resemblance between the stochastic contraction algorithm and the single linkage algorithm is evident, one being the randomized version of the second. Specifically, stochastic contraction selects an edge with probability proportional to its weight, while single linkage selects the edge with maximal weight. The connection is interesting, since single linkage is a purely local method, which does not attempt to optimize a global criterion. Although its output can be characterized globally as constructing a minimal spanning tree, it is not a clustering algorithm of the type that minimizes similarities between clusters and/or maximizes similarities within clusters.

The stochastic contraction is designed to generate with high probability *all* of the low capacity cuts (within a constant from the lowest capacity cut). Together with these low capacity cuts, however, cuts of higher capacity will also be generated. There is an important difference, though: in the original graph there is an exponential number (as a function of the number of nodes) of cuts of higher than threshold capacity, while in our sampling the total sample space is polynomial. Since this enormous reduction in the size of the sample space is obtained without losing *any* low capacity cut, the result is a very significant sampling bias towards the lower capacity cuts (see Section 4.3). Assuming that cut capacity is a global measure for clustering quality, this change in cut distribution is what makes our algorithm work in practice.

Using the cut capacity as a global quality measure for clustering is a well known approach (e.g., [43]). One of its problems is that the minimal cut tends to break small parts from the graph. Consequently [37] proposed another cost function, called normalized cut, which introduced a penalty for non balanced partitions. Our approach is different. We do not seek the global optimum of a certain cost function. Instead, we are interested in "averaging" the partitions with weights proportional to their quality (measured by capacity). In this sense our approach is similar to clustering methods inspired by statistical mechanics [13, 18, 30] and in particular it was inspired by the granular magnet model of [1], also called the super paramagnetic clustering (SPC) algorithm.

Our algorithm can be thought to replace the pairwise weights, prior to clustering, by higher order "collective" weights. Using higher order relations is used explicitly or implicitly by other methods as well. For example, it is known from the Gomory-Hu theorem [19] that the minimal cut separates the nodes of the graph in such a way, that the max-flow value for every two nodes within a component is larger than every max-flow value between components. Hence the min-cut algorithm is equivalent to the definition of max-flows as collective similarities, followed by thresholding. Relaxation methods which use weight propagation can be viewed in the same way.

To summarize, we note that one of the important aspects of our work is the link it provides between clustering approaches which seemed to be very different, i.e., SPC, the min-cut algorithm and the single linkage algorithm.

3 Results

In this Section we apply our clustering algorithm to three fundamental computer vision problems: image segmentation, perceptual grouping of edge elements, and image classification. These problems deal with the aggregation of visual elements, such as pixels, edgels or images, into coherent groups. In fact, the synthetic point set example used in Section 2.3 is no exception, but in this section we focus on real world computer vision problems, analyzing real images and contours extracted from images. In Sections 3.1 and 3.2 we consider segmentation of intensity and color images respectively. For comparison, we provide the segmentation results obtained by the split-and-merge segmentation algorithm[2] from the same images. In Section 3.3 we discuss the case of "unstructured background", which includes the problem of grouping edge elements. In Section 3.4 we integrate our contour matching algorithm and our clustering algorithm into an application of image database organization. Specifically, a collection of 121 images is hierarchically divided into shape categories. Finally, we summarize the comparison of our method with other methods in Section 3.5.

3.1 Segmentation of intensity images

For image segmentation, nodes in the graph represent individual pixels. We consider images where each pixel has a single intensity attribute. In accordance, the similarity weight w_{ij} between pixels (nodes) i and j increases with increasing spatial proximity and brightness likelihood. From Equation (2) we have:

$$w_{ij} = e^{-\frac{d(1)^2_{ij}}{a^2} - \frac{d(2)^2_{ij}}{b^2}} \tag{3}$$

where $d(1)$ is the Euclidean distance between the pixels in the image plane, $d(2)$ is the intensity difference between the two pixels, and a, b are corresponding scale parameters. As in [28, 37], we determine the parameters a and b manually. To reduce the number of edges and get a sparse graph, we eliminate edges

[2]The implementation of the split-and-merge algorithm was taken from KUIM, an image processing system provided by Prof. John M. Gauch from the university of Kansas.

whose weight is below some threshold and consider short range neighborhoods (see captions of Figures 5-9 for details).

The observation that the similarity graph is sparse is crucial for practical image segmentation applications. In [37] a sparse graph was obtained by randomly picking a small number of edges for each pixel in a limited range neighborhood. We adopt a similar approach: we consider the eight nearest neighbors of each pixel, or we consider the four nearest neighbors and randomly pick another four (we do not observe significant differences between these two methods). In addition, we eliminate edges whose weight is below some threshold.

Figure 5 shows a gray level image taken during a baseball game. We use the same image used by Shi and Malik [37] to be able to compare the two methods.[3] Next to the gray level image, the graph of $\Delta T(r)$ is shown. 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 four levels of segmentation results which correspond to the four highest peaks are shown below the image and the impulse graph.

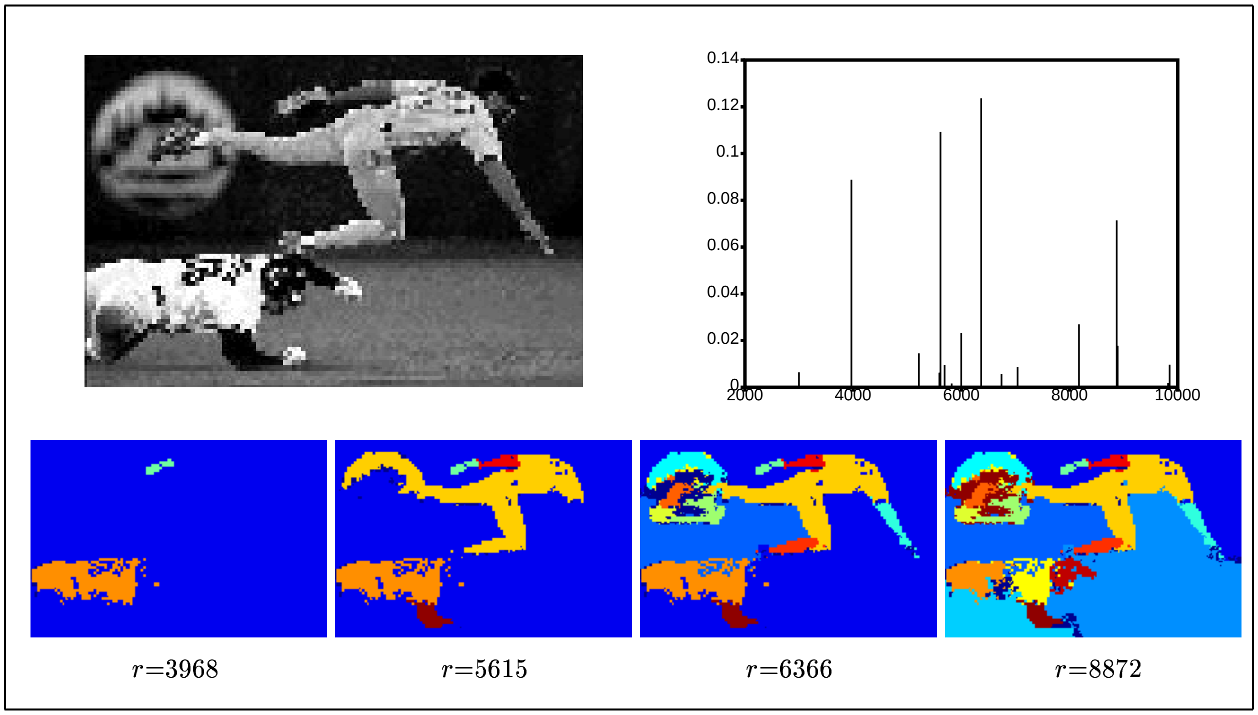

Figure 5: Brightness image segmentation: the original image of size 147×221, the impulse graph of $\Delta T(r)$, and the four segmentation results which correspond to the four highest peaks. **Parameter setting:** Intensity range is [0,1], a=8, b=0.1, edges with weight w_{ij} below 0.01 are eliminated, and only edges connecting each pixel with its four spatial nearest neighbors plus four random neighbors are included. Minimal cluster size of interest is 100. The graph contained 191,756 edges. Time per iteration: 2.11sec on Pentium II 450 Mhz. M=1000.

In all our image segmentation examples in this section, we regard clusters which contain less than 100 pixels as small clusters, and initialy we do not label them. The pixels which belong to these small clusters may be joined later with the larger clusters, as discussed in Section 4.1.3. We note that it is possible to get even finer level of resolution by further decreasing the minimal cluster size. For example, the palms of the hands of the lower player contain less than 100 pixels, and they can be detected too if a lower value is used. There is a natural tradeoff between the required resolution and the robustness to noise that generates spurious clusters.

For comparison, Figure 6 shows the results when using the split-and-merge segmentation algorithm. In the implementation we used, a user-specified variance threshold indirectly controls the number and size

[3]The same image is used in [8] but in different resolution.

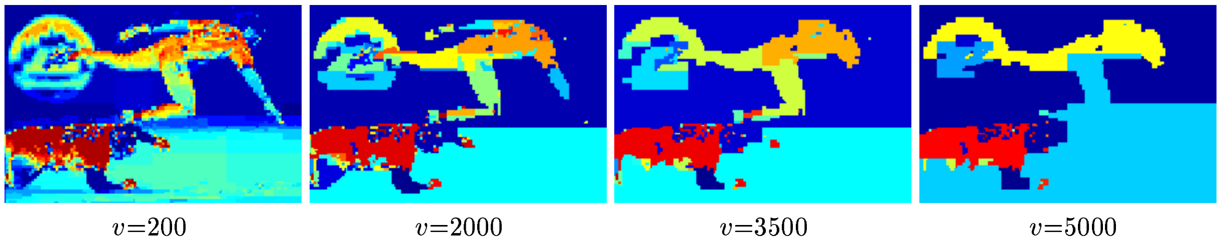

Figure 6: split-and-merge segmentation results for the image in Figure 5 with a few parameter values (variance thresholds v).

of regions produced by the algorithm. There is no methodology for selecting a good threshold, and the segmentation result changes gradually with the chosen parameter. (On the other hand, one of the most important features of our algorithm is that the segmentation shows phase transitions when the control parameter is varied.) As can be clearly seen, the segmentation results of the split-and-merge algorithm are not as good as ours, even when the control parameter is manually tuned for best visual results.

3.2 Segmentation of color images

Here too nodes in the graph represent individual pixels. The difference between the segmentation of color images and brightness images lies solely in the similarity measure between pixels. While brightness is a one dimensional attribute, the color resemblance between pixels is measured in a three dimensional color space. The weight w_{ij} is computed as before from (3), but now $d(2)$ is measured in a *3D* color space. If using one of the "perceptually uniform" color spaces, CIE-LAB or CIE-LUV, $d(2)$ is the Euclidean distance in the corresponding space.

Figure 7 shows an outdoor scene with a rainbow and a cow. The arrangement of the inserted images is the same as in Figure 5, with the impulse graph of $\Delta T(r)$ shown next to the original image, and the three segmentation results which correspond to the three highest peaks in the impulse graph shown below. Note the saliency of the leftmost segmentation level at r=678, where a very large peak is obtained due to the splitting of two very large areas (the sky and the grass regions). The analysis is in CIE-LAB color space.

Figure 8 shows an image of four airplanes flying above a desert. We present the analysis in CIE-LUV color space. The four highest peaks in the graph of $\Delta T(r)$ correspond to the separation of the four planes from the background; each peak separates one plane. The other peaks indicate partitions of the background into regions of different tones of yellow (not shown). For comparison we conducted a control experiment with the split-and-merge algorithm; in this experiment none of the planes was detected and the segmentation result was quite arbitrary.

3.3 Unstructured background and grouping of edge elements

Our previous examples of image segmentation explicitly assume that the images can be partitioned into regions of homogeneous brightness or color. If, on the other hand, the image contains a region of random noise, then the pixels belonging to this region are not similar to each other and cannot be clustered by methods that are based on homogeneity maximization. Our algorithm is useful for this case as well, as long as the relabeling option is not used. Our algorithm handles such cases by letting the random noise items form many isolated clusters.

Image segmentation with unstructured background

We start with a synthetic example, where the task is brightness image segmentation. Here we follow the recent work of Perona and Freeman [28], who considered the segmentation of "structured" foreground from "unstructured" background. Figure 9a shows a synthetic image that was generated according to the parameters given in [28]. Next to it, in the impulse graph of $\Delta T(r)$, we observe a large peak at r=654. The

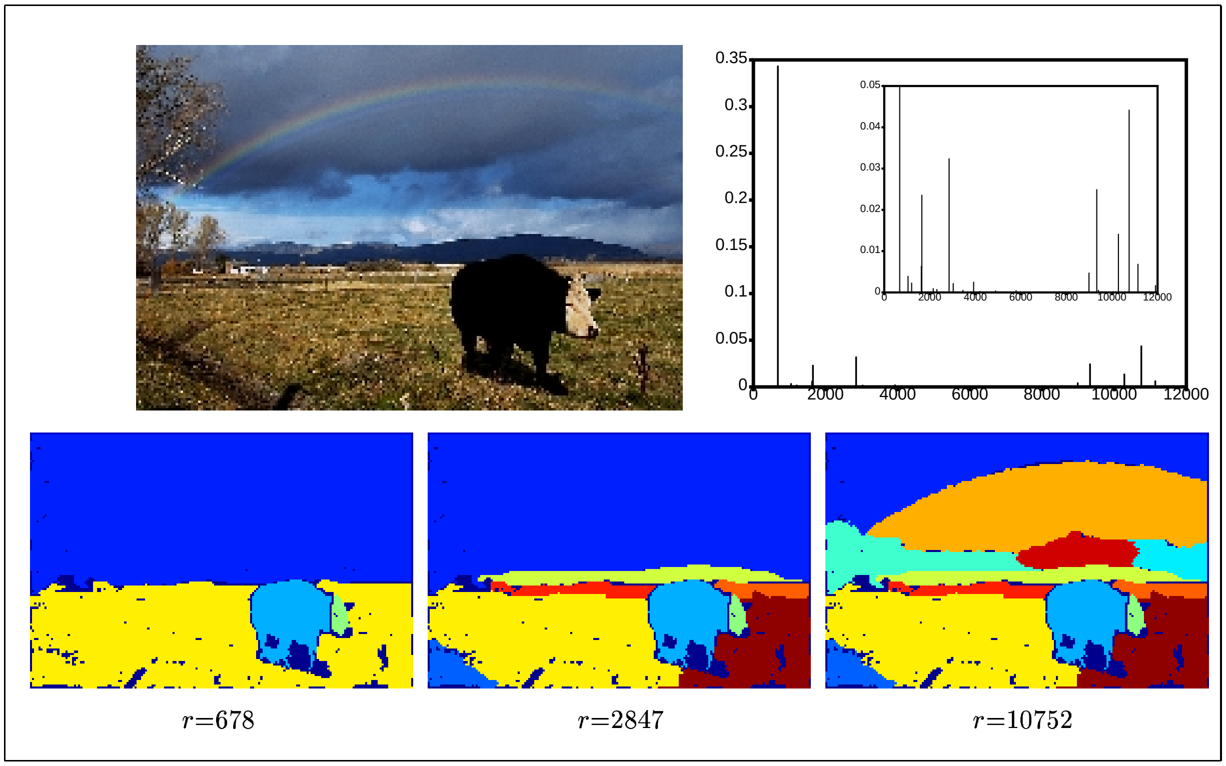

Figure 7: Color image segmentation in CIE-LAB color space: the original image of size 122×183, the impulse graph of $\Delta T(r)$ (insert shows an enlarged graph), and the three segmentation results which correspond to the three highest peaks. **Parameter setting:** Colors are represented in CIE-LAB color space, a=64, b=9, edges with weight w_{ij} below 0.001 are eliminated, and only edges connecting each pixel with its eight spatial nearest neighbors are included. Minimal cluster size of interest is 100, and a sample of M=500 cuts is used to estimate the pairing probabilities. The graph contained 71,765 edges. Time per iteration: 0.89sec on Pentium II 450 Mhz.

partitions which are obtained on both sides of this peak are shown in parts (c) and (d). In both (c) and (d), the background pixels form tiny clusters, each one including a few neighboring pixels which happen to have similar brightness values. These tiny clusters reflect true structure in the image, which we can screen out by using small size threshold, leaving these pixels unlabeled.

To decide how many meaningful partitions there are, we use a cross validation scheme (see [9] for details). The cross-validation indices which we got for the five highest peaks were: 0.46, 0.02, 0.08, 0.02, 0.08 respectively, clearly indicating that only the first partition should be accepted.

A comparison with the normalized cut algorithm [37] and with the factorization algorithm [28] is presented in Figure 10. The results are in agreement with those reported in [28], showing that the normalized cut algorithm fails completely in the presence of unstructured noise, while the factorization algorithm performs less well than our method.

Perceptual grouping of line segments

As a second example of separating structure from cluttered background, we consider the problem known as perceptual grouping of line segments. In such problems the raw data is typically a set of edgels, or line segments, and accordingly nodes in the graph represent line elements. The task is to group together a set of line segments that together define a perceptually appealing "edge", typically expected to have one or more of the following properties: smoothness, closure, or convexity. In these problems typically the clusters of interest contain only a few elements in comparison with the number of elements in the background (the clutter), see Figure 11.

A full treatment of this problem, which takes care of clutter as it interferes with procedure STAGE-2,

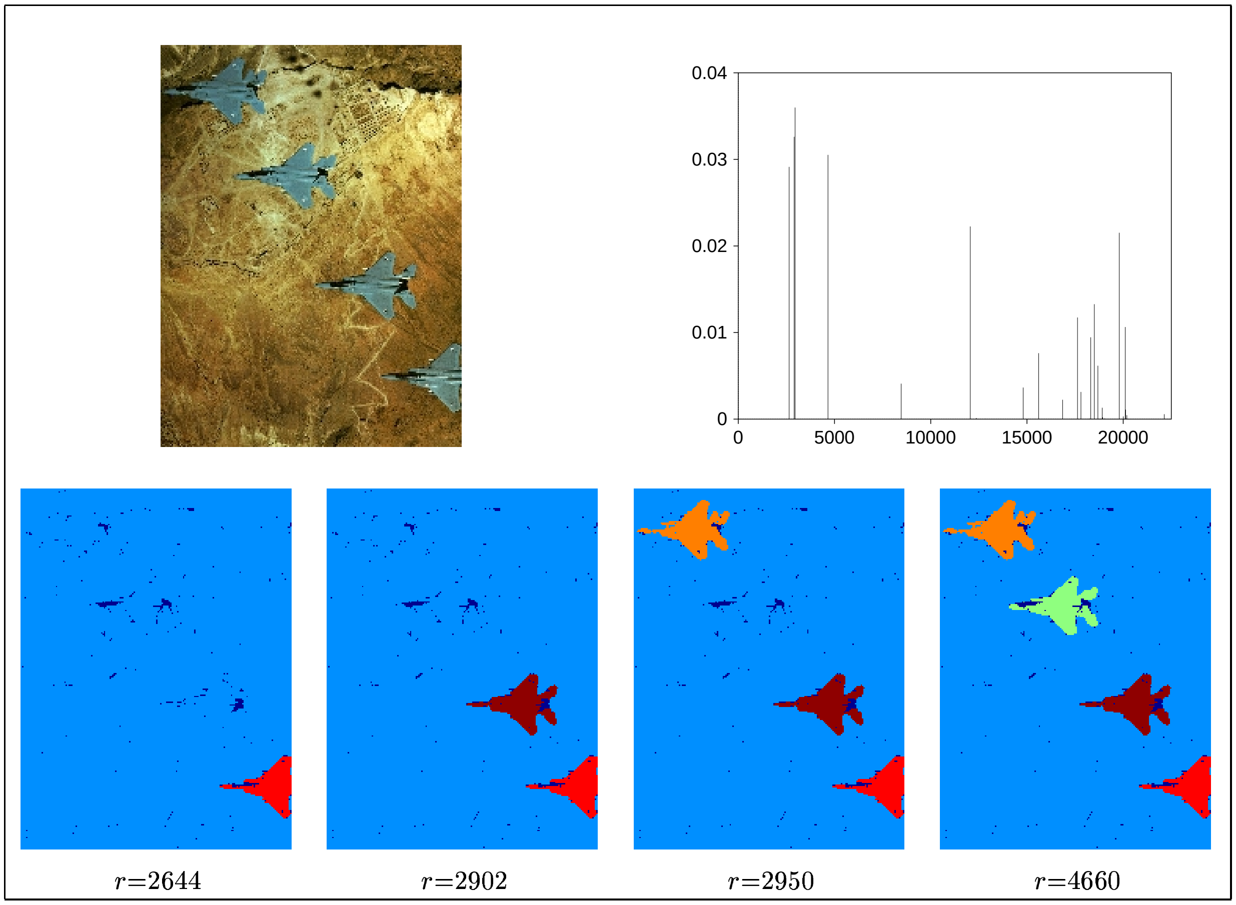

Figure 8: Color image segmentation in CIE-LUV color space: the original image of size 256×192, the impulse graph of $\Delta T(r)$, and the four segmentation results which correspond to the four highest peaks. **Parameter setting:** Colors are represented in CIE-LUV color space, a=64, b=9, edges with weight w_{ij} below 0.001 are eliminated, and only edges connecting each pixel with its four spatial nearest neighbors plus four random neighbors are included. Minimal cluster size of interest is 100. The graph contained 255,837 edges. Time per iteration: 4.10sec on Pentium II 450 Mhz; M=500.

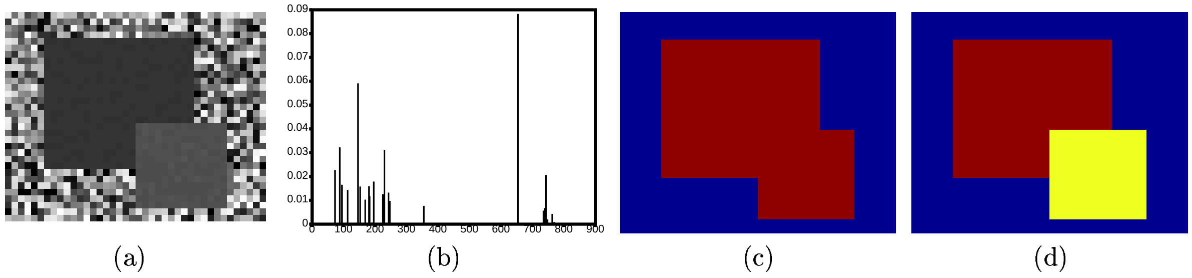

Figure 9: Separation of homogeneous objects from noisy background. (a) Synthetic 30×42 image, generated with the same parameters as in [28]: the brightness values are uniformly distributed between 0 and 1 for the background, between 0.2 and 0.21 for the larger rectangle, and between 0.3 and 0.31 for the smaller one. (b) The graph of $\Delta T(r)$: the largest peak appears at r=654, the second largest appears at r=146. Minimal size of interest for a cluster is 10. (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. Parameter setting: a=3, b=0.1 (like in [28]), only edges with the 8 nearest neighbors of each pixel are included.

is beyond the scope of the current paper (see [12] for a detailed discussion). But in order to illustrate the usefulness of our approach to mid-level vision problems, we show next one example where our clustering algorithm is used to perform perceptual grouping.

Figure 11 shows a contour of a lemon superimposed on random textured background. The total number of line segments in this figure is 440, while the number of segments which construct the lemon contour is only

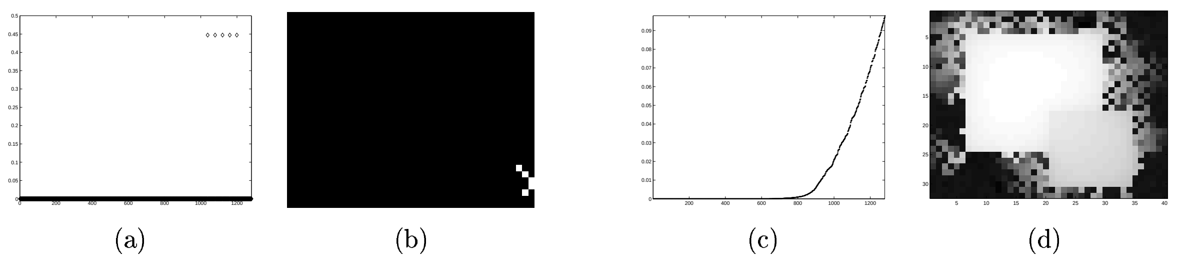

Figure 10: The performance of other algorithms applied to the data of Figure 9. The same preprocessing and exponential transformation from distances to weights was used (see text). **Left:** The normalized cut algorithm [37]. (a) One of the best Ncut partitions (there are many partitions which correspond to a zero eigenvector, 33 of them separate a single pixel). (b) The partition which is suggested by the eigenvector shown on the left. The five isolated entries are connected to each other, but not to the rest of the image, hence the cut value is zero. **Right:** The factorization method [28]. (c) The first eigenvector of the similarity matrix, sorted by the value of its entries. There is no clear threshold to choose. (d) The same eigenvector presented using log of intensity scale, where the entries are ordered like the image pixels.

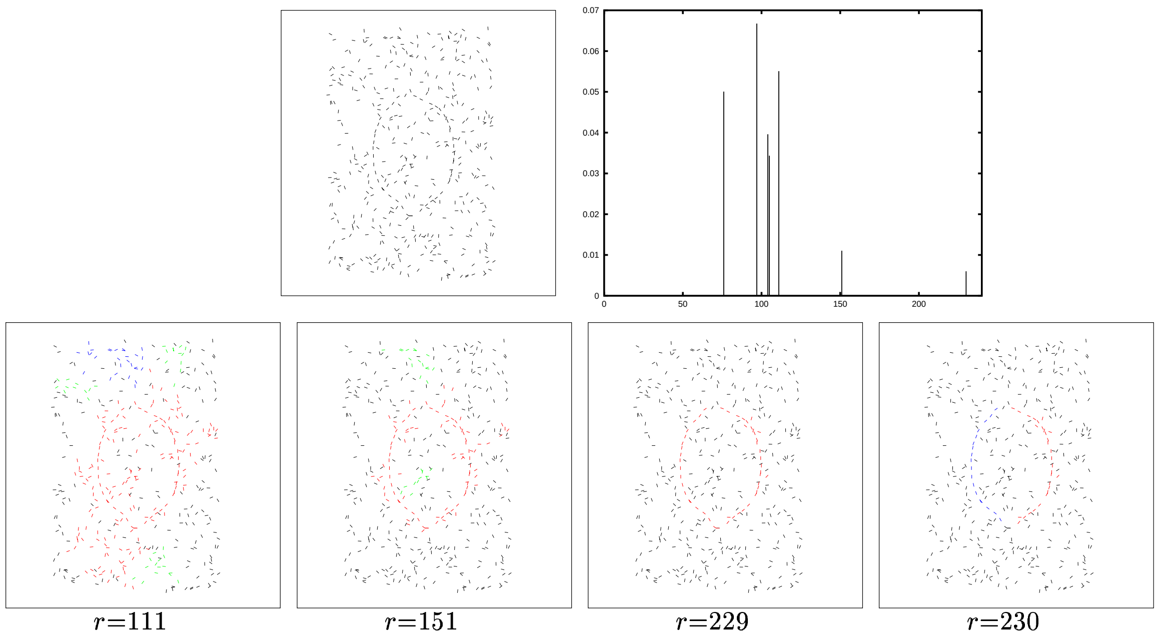

Figure 11: Perceptual grouping of line segments. The data is shown in the upper left image. If the minimal size of interest for a cluster is set to 10, then 7 transitions are detected, which are marked by the 7 peaks in the impulse graph of $\Delta T(r)$ (upper row, on the right); each peak signals the segmentation of a set of at least 10 line segments. The segmentation corresponding to the last 3 peaks are shown in the bottom row. The meaningful peak, at r=230, is not high since the cluster of interest is small. The partitions obtained on both sides of this peak are shown for (r=229, 230).

44 (signal to noise ratio of about 10%). The image and the similarity measure w_{ij} are adopted from [38]. We do not repeat here the similarity (affinity) formula, which is quite complex, and note that it depends on relative orientation and spatial proximity between line segments. After the similarity between every two line segments is computed, we discard low weight edges; our elimination criterion uses the mutual-k principle [1], namely, a weight w_{ij} is kept if edge i is one of the k-nearest neighbors of edge j and vice versa (k=10 is used). The remaining weights are fed into our clustering algorithm.

Most of the large peaks in the impulse graph of $\Delta T(r)$ result from the separation of small sets of line segments from a large set. However, by the time that the interesting separation occurs, the cluster which is split is small (see the transition from r=229 to 230), and the splitting is not accompanied by a large change in $T(r)$. Nevertheless, if we ignore variations in $T(r)$ that result from segmentation of clusters smaller that 10 elements, as we did in Figure 11, then there are only 7 candidate partitions to consider; and as noted above - the selection between them can rely on additional external perceptual criteria. The crucial result to

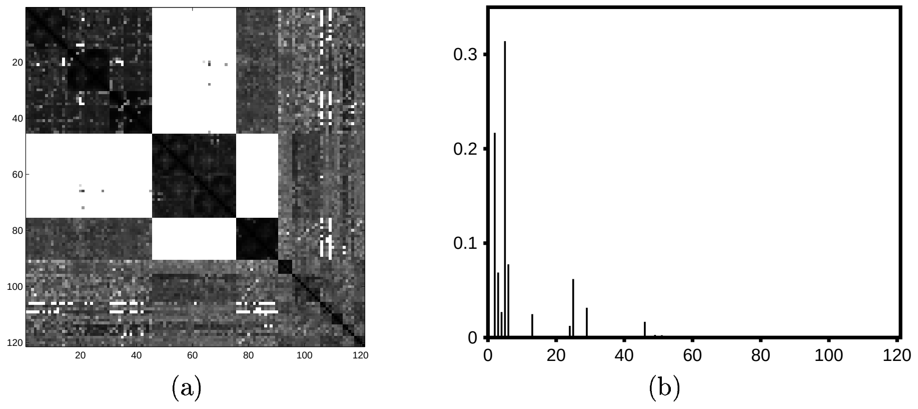

Figure 12: (a) The dissimilarity matrix d_{ij} computed for 121 images as described in [10], by silhouettes extraction, syntactic matching, and computation of residual distances between matched points. Larger dissimilarity values are represented by brighter intensities. The infinity value is represented by pure white. Images which belong to the same class are ordered next to each other, hence the matrix shows a block diagonal structure. The order of the classes is: cows, hippos, wolves, cars, sport cars, children, hand palms, fish, planes, rabbits, tools and artificial shapes. (b) Clustering (Section 2): The impulse graph of $\Delta T(r)$ versus r, computed with 1000 iterations. The peaks correspond to the partitions shown in Figure 13. We have considered peaks above 0.01 to be meaningful.

note is that the desired segmentation is indeed found and identified as a member of a small set of partitions that should be evaluated.

3.4 Organization of an image database

We now describe another application related to *3D* object recognition. The database contains 121 images of 12 different objects, including the image database created by Sharvit & Kimia (see [34]), which consists of 31 silhouettes of 6 classes of objects (including fish, airplanes, and tools). In addition, we collected 90 images by placing 6 toy models on a turntable, so that the objects could be viewed from different viewpoints. For each of the 6 toy models we collected 15 images, by rotating them in azimuth ($\vartheta = -20°, -10°, 0°, 10°, 20°$) and elevation ($\varphi = -10°, 0°, 10°$). We used models of a cow, wolf, hippopotamus, two different cars and a child. There is weak geometrical similarity between all the 45 silhouettes of the three mammals, and there is weak geometrical similarity between the 30 different silhouettes of the two cars. A "good" shape categorization procedure should reveal this hidden hierarchical structure.

All the images were automatically preprocessed, to extract the silhouettes of the objects and represent them syntactically. The dissimilarities between the silhouettes were estimated using the algorithm described in [10]. In order to compare all the image pairs in our database of 121 images, we performed 7260 matching assignments; this took about 10 hours on an INDY R4400 175Mhz workstation (about 5 seconds per image pair, on average). The output of this computation is a dissimilarity matrix d_{ij} of size 121×121, shown in Figure 12(a).

Our next step is to represent the images as nodes in a graph, and assign a similarity weight w_{ij} to each edge. In accordance with Equation (2) we define

$$w_{ij} = e^{-\frac{d_{ij}^2}{a^2}}$$

where a is a local scale parameter, here chosen to be the average distance to the second nearest neighbor. After transforming d_{ij} to w_{ij} we threshold the edges (by $\theta = 10^{-5}$) to obtain a sparse representation.

Figure 12(b) shows the graph of $\Delta T(r)$, while the corresponding hierarchical classification (dendrogram) is shown in Figure 13. At the highest level (r=1) all the images belong to a single cluster. As r is increased, finer structure emerges. Note that related clusters (like the two car clusters) split at higher r values; comparing with the ideal human perceptual classification, our finest resolution level is almost perfect, with only two classification errors (in the boxes marked by $*$) and the undesirable split of the fish cluster.

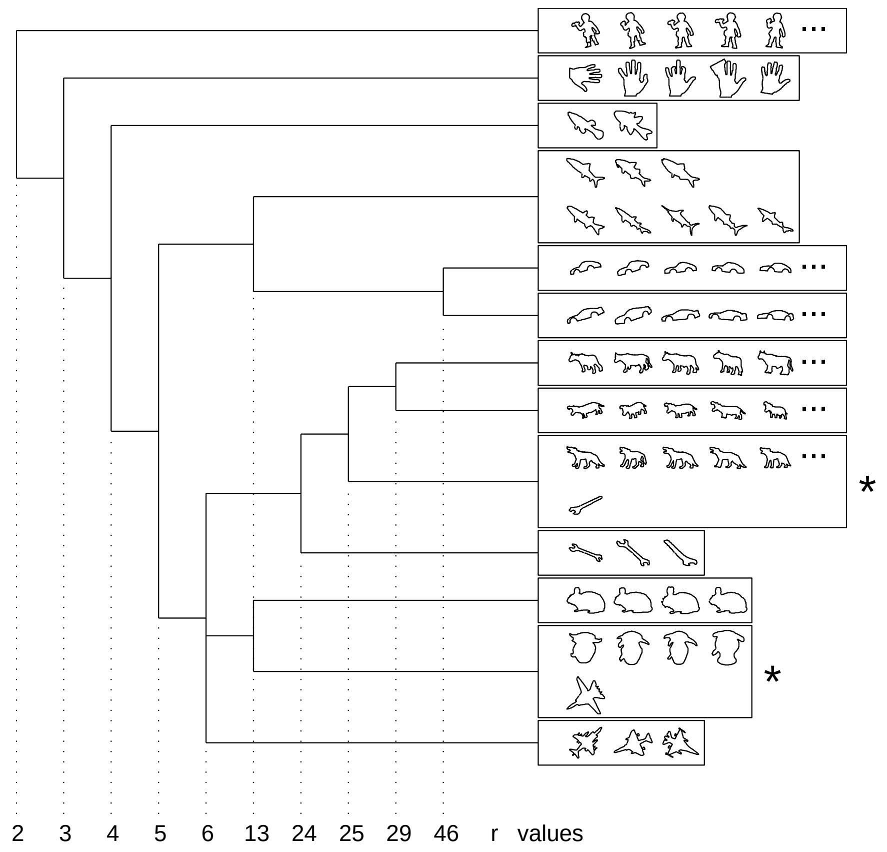

Figure 13: The classification tree obtained for an image database consisting of 121 images. It is based on the distance matrix shown in Figure 12, which is the input to the current step. As can be seen in Figure 12, the clustering task is not easy, since many entries which are off the diagonal blocks are quite dark, and vice versa. Recall also that the input matrix does not come with the correct rows and columns order. In the dedrogram shown, the finest classification level is shown by putting each cluster of silhouettes in a box. For the large clusters representing our own toy models (see text) the figure shows only 5 exemplars out of 15, but the other 10 are classified correctly as well. Note that the lower levels of the tree correspond to meaningful hierarchies, where similar classes (like the two cars or the three sets of mammals) are grouped together. The vertical axis is not drawn to scale.

Comparisons

Our image classification procedure consists of two parts: the estimation of pairwise dissimilarities and the detection of clusters. In all our comparison experiments we used as input the same matrix w_{ij}, to eliminate any dependence on the data preprocessing. We applied a few agglomerative algorithms and two spectral methods. The conclusion from the performance of the agglomerative techniques is that the images form chained structures, where images in the same cluster might be related to each other not directly but through "mediating" images. Since the nearest neighbors of all the images are in the same class , the single linkage algorithm performs quite well, although it does not give hints as to the natural thresholds which determine the hierarchies. On the other hand, the complete linkage algorithm, which seeks compact structures, fails completely.

The quality of the results obtained by the normalized cut algorithm is comparable with ours. While our algorithm computes the whole hierarchy at once, the normalized cut algorithm needs to be applied recursively to each part. The stopping condition of this divisive process is not well formalized; the division should be halted when the entries of the eigenvector do not show a clear partition into two groups. We have inspected the entries and halted the divisive process when we could not "see" a clear partition. In the final stage most of the natural classes were correctly identified, with the exception of the fish class (which was broken into 3 sets), and the tools class - members of which had been erroneously merged with the hippo class and the wolf class.

Finally, we recursively applied the factorization method to our data. The final partition was worse than the previous two algorithms, but still sensible: the classes of 6 toy models were correctly identified, although the 30 images of the two car classes could not be separated; in the final identification of the remaining classes a few members were typically missed, while the fish class was completely broken to pieces. In addition, this method did not suggest any hierarchical interpretation of the data.

3.5 Comparison with other work: summary

The spiral arrangement of points in the example studied in Section 2.3, and the added noise, were chosen in order to emphasize a few advantages of our method. First, our algorithm does not exploit prior knowledge to model the data distribution, hence arbitrary cluster shapes can be handled. This is in contrast with parameter estimation methods, that usually assume Gaussian distributions. Second, our algorithm is robust to noise. This is in contrast with nearest neighbor methods such as single linkage, which are sensitive to "bridges" connecting large clusters. Third, though our algorithm is not given the desired number of clusters, it finds the desired natural partitions. Moreover, to find these partitions we do not use recursive divisions into two parts, the method employed by most other hierarchical methods; hence we have a natural possibility to leave problematic points unclassified.

Some of our results were compared above with other methods, including the split-and-merge segmentation algorithm, and the recently proposed normalized cut algorithm [37] and the factorization algorithm [28]. We conclude that there are cases where our method works well while the other segmentation methods may fail completely (Figures 4,18). In other cases, like in the image segmentation of Figure 5, our method appears to work better. In particular, in this example we succeed to segment fine details such as different body parts which the normalized cut algorithm failed to find. In the clustering of image database the performance of our algorithm and the normalized-cut was comparable, while the factorization method performed somewhat worse. Finally, our clustering algorithm is general enough to handle perceptual grouping, showing better performance over other methods such as the factorization method [28] (while the normalized cut method completely fails as it typically runs into singularities when dealing with such problems).

It must be noted that the complexity of our algorithm appears to be lower: while our running time for sparse graphs is $O(N \log^2 N)$, the complexity of the normalized cut for sparse graphs is $O(LN)$, where L is the maximal number of matrix-vector multiplications allowed in the Lanczos procedure. The number L depends on many factors, and Shi and Malik observed that it is typically less then $O(\sqrt{N})$. In addition, our algorithm provides hierarchical clustering in a single path and at no additional cost, and it provides better measures of which solution to choose and when to stop the divisive process. On the other hand, we note that the brute-force (or high-complexity) implementation of the normalized cut algorithm is by far simpler, a few lines in matlab in its simplest form. A parallel implementation of our algorithm is trivial, since it is constructed from $O(\log N)$ *independent* iterations. Thus the total work can be easily divided between $O(\log N)$ processors.

4 The typical cut algorithm: details

We now provide an extended description and analysis of our algorithm. In Section 4.1 we discuss the efficient implementation of our algorithm, and complete its complexity analysis. Section 4.2 uses a few synthetic examples to demonstrate the robustness of our method. In Section 4.3 we quote the known properties of the probability distribution that is imposed by the contraction algorithm (introduced in Section 2.2). The full implementation is available from [44].

4.1 Efficient implementation

The efficient implementation of the algorithm is based on two observations and one assumption. The assumption is that the typical cut is not sensitive to the pairing probability p^r_{ij} of an edge whose original weight w_{ij} is small. This assumption is not straightforward; although a zero weight edge is never selected during contraction, its associated probability p^r_{ij} will be nonzero for some r's, due to transitive relations via other nodes which put nodes i and j in the same meta-node. Our assumption is that since p^r_{ij} in this case is determined solely by transitive relations, we can rely on them in the formation of connected components by STAGE-2. Thus, if very small weight edges are discarded, the number of pairing probabilities to compute reduces from N^3 to $N|E|$.

Two important observations allow us to construct an efficient algorithm. First, the value of p^r_{ij} is never used; we only need to know whether it is smaller or larger than $\frac{1}{2}$. This observation is used in Section 4.1.1 to further reduce the number of computed variables down to $O(|E|)$. Second, the contraction of a sparse graph can be implemented in $O(|E| \log N)$ time. This is shown in Section 4.1.2. Finally, Section 4.1.3 shows how the typical cuts at successive values of r can be computed incrementally.

4.1.1 Computing the level-crossing of probability 0.5

We notice that p^r_{ij} is monotonic in r by construction, with $p^N_{ij}=0$ and $p^1_{ij}=1$, since in a single contraction path if i and j belong to the same meta-node of G'_r for some r then they remain in the same meta-node for all subsequent (smaller) values of r, see procedure STAGE-1 in Section 2.2. Thus in order to know for each r whether p^r_{ij} is smaller or larger than 0.5, it is sufficient to know for every i-j pair which r satisfies $p^r_{ij} = 0.5$. We denote it r_{ij}. We then know that $p^r_{ij} < 0.5$ if and only if $r > r_{ij}$. The immediate consequence is that the number of computed variables is reduced from $N|E|$ to $|E|$. Another consequence, which will be discussed in Section 4.1.3, is that the computation of the typical cuts by STAGE-2 becomes incremental.

To see how r_{ij} is estimated we refer again to procedure STAGE-1, and observe the nodes i and j at the m-th iteration ($m = 1 \dots M$). As already discussed, there is a single r value, here denoted r_m, in which the edge (i, j) is contracted. Thus in the m-th iteration nodes i and j are at different meta-nodes for every $r > r_m$, and at the same meta-node for every $r \leq r_m$. It is easy to see that the median r' of the sequence $\{r_1, r_2 \dots r_M\}$ is the sample estimate for the level r_{ij}. This is simply because at level r' there are $M/2$ iterations in which nodes i and j belong to the same meta-node, and $M/2$ iterations in which they do not.

The rest of this section describes how the median r' is approximated (for each i-j pair) without storing the whole sequence of r_m's. Assume that the sequence has been somehow arranged in K bins, such that every bin contains the same number of elements. We could then use the mean of the middle bin as an estimator for the median. Note that K=1 would give the whole sequence average as the estimator. A larger K (but still $K \ll M$) would give a better estimate, due to the non symmetric distribution of the r_m's over the positive real axis.

The question is how to (approximately) arrange the sequence in such a K-bins histogram. We use the following heuristic, inspired by an online K-means algorithm for non stationary data [14]. For every bin k ($k = 1 \dots K$) we store its running average a_k, and the number n_k of elements accumulated in it. The two lists $\{a_1 \dots a_K\}$ and $\{n_1 \dots n_K\}$ together will be called below the histogram associated with a certain edge. Suppose that the first m-1 values $r_1 \dots r_{m-1}$ were used to create a histogram, and a new value r_m is obtained at the m-th iteration. The histogram is updated using the procedure INSERT described in Figure 14. The idea is to merge the two adjacent bins which together include a minimal number of elements, and create a new bin which contains just the new element. This procedure eliminates almost completely any dependence on the first collected measurements. When the whole sequence has been accumulated, we find the minimal k_0 that satisfies $\sum_{k=1}^{k_0} n_k > M/2$, and we take a_{k_0} to be the median estimator. For future reference, the operation that returns a_{k_0} is called MEDIAN.

Experimentally, our strategy is found to be successful in approximating the medians of test distributions, which are not symmetric. It is also found to be insensitive to the prefix of the supplied sequence.

```
procedure INSERT:
input:   a value $r_m$ and a histogram of $m-1$ elements.
output:  a histogram over $m$ elements.

Find $k$ such that $n_k + n_{k+1}$ is minimal.
Merge bins $k$ and $k+1$:
    $a_k \leftarrow (n_k a_k + n_{k+1} a_{k+1})/(n_k + n_{k+1})$
    $n_k \leftarrow n_k + n_{k+1}$
Delete bin number $k+1$.
Create a new bin, call it bin number $l$, with $n_l = 1$, $a_l = r_m$.
Locate the new bin (choose $l$) such that $a_1 \ldots a_K$ is kept ordered.
```

Figure 14: Updating an existing histogram with a new value.

4.1.2 Contraction of sparse graphs

The term "graph contraction" refers to a single path from an N-way cut to a 2-way cut. A single graph contraction generates for every r (r=N...1) a single sample of an r-way cut by repeating the following three steps until 2 meta nodes remain:

- select edge (i, j) with probability proportional to w_{ij}.
- replace (meta) nodes i and j by a single meta 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.

In this section we propose a novel algorithm for the contraction of sparse graphs. Our algorithm uses the building blocks of the classical union-find algorithm, which partitions a set of items into equivalence classes. Accordingly, a name is assigned to each meta node, which for convenience is the node number of one of its members. Two operations are defined:

- The `UNION` operation gets two meta nodes names, and assigns the name of the larger meta node to the members of the smaller one.
- The `FIND` operation gets a node index, and returns the name of the meta node of which this node is a member.

Our algorithm is described by the pseudo-code in Figure 15, explained line by line below. For completeness we have included in the code the outermost loop, which repeats the contraction M times, and the procedures `INSERT` and `MEDIAN` discussed in the preceding section. Hence we present here a complete new version of the procedure `STAGE-1` introduced in Section 2.2.

Line 1: The estimation of the 0.5-probability level uses a histogram data structure, as discussed in Section 4.1.1. For every edge $e \in E$ the assigned histogram is $H(e)$.

Line 2: T is a binary tree with $|E|$ leaves. Each leaf represents an edge, and can be directly accessed by a leaf index. Both top-down and bottom-up traversals are supported (this tree supports the efficient selection of edges).

Line 3: Repeats the contraction M times, and collects the data for median estimation.

Line 4: Initialization of the binary tree T. The procedure `INITIALIZE` assigns each leaf of T with the weight of the corresponding edge, and each inner node of T with the sum of weights of its two children.

Line 5,6: The variable *ne* keeps the number of edges which cross the current cut, hence initially it is set to $|E|$ (every node is a meta node of size 1). The level r is set to N (the number of parts in the current cut).

Line 7: The contraction loop implements the edge selection, the meta nodes unification, and the maintenance of the tree data structure. Since the number of meta nodes is reduced by one at each iteration, the loop is executed no more than N-1 times.

```
procedure EFFICIENT-STAGE-1:
input:   sparse graph $G(V,E)$ with $N$ nodes.
output:  0.5-probability level $r_{ij}$ for each edge $(i,j) \in E$.

(01)  Let $H$ be an array of $|E|$ histograms, one per edge.
(02)  Let $T$ be a binary tree whose leaves represent the edges.
(03)  for $m = 1 \ldots M$:
(04)     INITIALIZE($T$)
(05)     $ne \leftarrow |E|$
(06)     $r \leftarrow N$
(07)     while $ne > 0$ do:   (the contraction loop)
(08)        $leaf \leftarrow$ SELECT-EDGE($T$)
(09)        $u,v \leftarrow$ META-NODES-NAMES($leaf$)
(10)        $L \leftarrow$ CONNECTING-EDGES($u,v$)
(11)        $ne \leftarrow ne - |L|$
(12)        $r \leftarrow r$-1
(13)        for each $leaf \in L$:
(14)           $T \leftarrow$ DISCARD-EDGE($leaf$,$T$)
(15)           $H(leaf) \leftarrow$ INSERT($r$,$H(leaf)$)
(16)        end-loop
(17)     end-loop
(18)  end-loop
(19)  for $leaf = 1 \ldots |E|$:
(20)     use lookup table to convert $leaf$ into an $i$-$j$ pair
(21)     $r_{ij} \leftarrow$ MEDIAN($H(leaf)$)
(22)  end-loop
(23)  return $R = \{(i,j,r_{ij})\}_{(i,j)\in E}$
```

Figure 15: Efficient implementation of the transformation from similarity weights to pairing probabilities.

Line 8: Stochastic edge selection. The procedure SELECT-EDGE returns one of the leaves of T with probability proportional to its weight. The selection is implemented by constructing a stochastic path from the root of T toward the selected leaf. At each inner node a coin is flipped in order to decide whether the path continues to the left or to the right child, where the probability of selecting a child is proportional to its weight. The procedure returns the index ($leaf$) of the chosen edge.

Line 9: Identification. The procedure META-NODES-NAMES gets the index of the chosen edge, and finds which two meta nodes u and v are connected by this edge. It uses a fixed lookup table to convert the edge index ($leaf$) into a pair of indices i and j of nodes in the original graph. Using the FIND operation, the meta nodes names are given by u=FIND(i) and v=FIND(j).

Line 10: The task of the procedure CONNECTING-EDGES is twofold. It applies the UNION operation to merge the selected meta nodes u and v, and in addition it returns a list of all the edges connecting them. Each member in the returned list L is a leaf index, of an edge connecting u and v. The algorithm for the constructing of the list L is described separately below, under the title "Complements and complexity analysis".

Line 11,12: The number of crossing edges (ne) and the number of meta nodes (r) are decreased, as a consequence of the contraction.

Line 13: Loop over all contracted edges. Each one of them has just became an inner edge of a meta node, hence we need to prevent its re-selection, and to store the level r in which the contraction occurred.

Line 14: Maintenance of the tree data structure. To prevent re-selection of a contracted edge the procedure DISCARD-EDGE set its leaf weight to zero (remember that the leaves can be directly accessed by their index). The procedure DISCARD-EDGE then follows the path from the leaf to the root of T, and subtracts the weight

of the leaf from each node that is passed through.
Line 15: Update the histogram of edge number $leaf$ with its contraction level r in the m-th iteration. The procedure `INSERT` is described in Figure 14.
Line 19–22: Use the information accumulated in the histograms during the M graph contractions to estimate the level of 0.5 probability for every edge. The conversion from $leaf$ index to (i, j) notation at line 20 is included for consistency with the notation of Section 4.1.1, where the procedure `MEDIAN` is described.

Complements and complexity analysis.
In the rest of this section we describe in more details the procedure `CONNECTING-EDGES`, and we analyze the asymptotic complexity of our algorithm. For both objectives we must provide some implementation details.

Let us define the following mappings. The array *names* is a mapping from node indices to meta nodes names, hence $names(i) = u$ if node i belongs to meta node u. The array *members* is a mapping from meta nodes names to lists of members, hence $members(u)$ is a list of nodes belonging to u. The array *sizes* is a mapping from meta nodes names to their sizes, hence $sizes(u) = n$ if meta node u contains n nodes.

With these mappings, `FIND`(i) returns the value of $names(i)$, and is clearly an $O(1)$ operation. On the other hand, assuming w.l.o.g that $sizes(u) \geq sizes(v)$, the operation `UNION`(u, v) takes $O(sizes(v))$ time. Instead of describing it separately, we incorporate the union operation into the procedure `CONNECTING-EDGES` and analyze them together. Figure 16 shows the implementation.

```
procedure CONNECTING-EDGES:
input:   two meta nodes names $u, v$
output:  list $L$ of connecting edges (and modified mappings)
use mappings:  $names$,$members$ and $sizes$ (see text).

(01)  Let $L$ be an empty list of edge indices.
(02)  if $sizes(u) < sizes(v)$ then
(03)     call CONNECTING-EDGES($v, u$)   (exchange arguments)
(04)     return
(05)  end-if

(06)  for each $i \in members(v)$:
(07)     for each $\{j, leaf\}$ in the adjacency list of $i$:
(08)        $t \leftarrow$ FIND($j$)
(09)        if $t$ equals $u$ then
(10)           add the edge index $leaf$ to $L$
(11)        end-if
(12)     end-loop
(13)     $names(i) \leftarrow u$   (part of UNION)
(14)  end-loop

(15)  $sizes(u) \leftarrow sizes(u) + sizes(v)$
(16)  $sizes(v) \leftarrow 0$
(17)  $members(u) \leftarrow$ CONCATENATE($members(u)$,$members(v)$)
(18)  $members(v) \leftarrow$ NULL

(19)  return $L$
```

Figure 16: Merging two meta nodes and finding all the edges which connect them.

In lines 2–5 we exchange the arguments u and v if u is smaller in size. Thus we can assume in the rest of the code that $sizes(v) \leq sizes(u)$, and hence the loop at lines 6–14 is executed for each member of the smaller meta node (v). Lines 7–12 constructs the list L of connecting edges (see below), and lines 13, 15 and 17 implement the union operation. We note that for efficient implementation of the list concatenation at line 17, it is desirable to keep pointers to the tails of the membership lists. Lines 16 and 18 are added for esthetics, as $sizes(v)$ and $members(v)$ are never accessed again.

The section of the code which might appear nontrivial is the construction of the list L at lines 7–12. We assume that the graph G is stored in the form of adjacency lists. A member of the adjacency list of node i has the form $\{j, leaf\}$, where j is a node adjacent to i, and $leaf$ is the index of the edge connecting them. The loop at line 7 queries all the edges which incident on nodes in v, checking if their other vertex is in u (line 9). If this is the case, then the edge index is added to the list.

Let us analyze the complexity of `CONNECTING-EDGES`. We are interested in sparse graphs, where the maximal length of an adjacency list is a constant independent of N. In this case the inner loop at lines 7–12 iterates a constant number of times, and can be ignored in the analysis of asymptotic complexity. However, ignoring this part we obtain a usual implementation of `UNION`, which is known to be $O(N \log N)$ for the entire contraction loop. We repeat the argument below.

Line 13 moves the members of v to the new meta node u. The time spent by a single union operation is therefore proportional to the number of nodes which are moved to a new meta node, namely $O(sizes(v))$. Since we always move the nodes of the smaller meta node, it turns out that whenever a node is moved to a new meta node, its new meta node's size is at least twice as large as the former one. As a consequence, no node can be moved more than $\log N$ times during the N-1 union operations which contract the graph. In other words, the total amount of time spent by all union operation during the contraction loop (line 7, Figure 15) is $O(N \log N)$.

To complete the complexity analysis we refer again to the pseudo-code in Figure 15. The call to `INITIALIZE` at line 4 costs $O(|E|)$ time. Every call to `SELECT-EDGE` at line 8 costs $O(\log |E|)$ time, and every call to `META-NODES-NAMES` at line 9 is of complexity $O(1)$. These bounds accumulate to $O(N \log |E|)$ and $O(N)$ respectively during one execution of the contraction loop (line 7). The overall complexity of `CONNECTING-EDGES` at line 10 has been shown above to be $O(N \log N)$ during one graph contraction. The procedures `DISCARD-EDGE` and `INSERT` at lines 14 and 15 are executed exactly once for every graph edge during one execution of the contraction loop. Since the complexity of `DISCARD-EDGE` is $O(\log |E|)$ and that of `INSERT` is $O(1)$, the total amount of time spent by them during one graph contraction is $O(|E| \log |E|)$ and $O(|E|)$ respectively. Finally, the computation of all medians by the loop at line 19 is $O(|E|)$. Summarizing these bounds and using the fact that $|E| \leq N^2 \Rightarrow O(log|E|) = O(logN)$, we conclude that a single graph contraction can be implemented in $O(|E| \log N)$ time. This complexity multiplied by M (line 3) is the total complexity of computing all the 0.5-probability levels. Using the result of Section 2.4, the overall complexity of `EFFICIENT-STAGE-1` is $O(|E| \log^2 N)$ time and $O(|E|)$ space. For sparse graphs, where $|E| = O(N)$, this is the same as $O(N \log^2 N)$ time and $O(N)$ space.

4.1.3 The second stage

In Section 4.1.1 we claimed that the monotonic dependence of p_{ij}^r on r leads to an incremental and efficient computation of the typical cuts. Here we show how this is done. The basic idea is to sort the graph edges in decreasing order of r_{ij} (the level of 0.5 probability). To find the typical cut at level r we mark all edges for which $r_{ij} \geq r$, then we find connected components of marked edges. Proceeding from level r to level $r-1$, marked edges can only be added due to the monotonic dependence of p_{ij}^r on r. Hence the partition at level $r-1$ can be found incrementally from the partition at level r.

Figure 17 summarizes our efficient `STAGE-2` algorithm. The list R is sorted at line 1 using quicksort, which takes $O(|E| \log N)$ time on average. The data structures *names*, *members* and *sizes* in line 3 are described in the previous section. They are initialized in accordance with level $r = N$, where every p_{ij}^r=0, and the typical cut is necessarily a partition to N clusters of size 1. The name which is given to every cluster is its node number.

The loop in lines 4–16 is a classical `UNION-FIND` algorithm for a dynamic graph, combined with incremental computation of $\Delta T(r)$, as defined in Equation (1). The complexity of the loop is $O(N \log N)$, as discussed in Section 4.1.2.

```
procedure EFFICIENT-STAGE-2:
input:   set $R$ of triplets $\{i,j,r_{ij}\}$ $r_{ij}$ is the 0.5-probability level of edge $(i,j)$
output:  hierarchy of a few selected partitions.

(01)  sort $R$ in decreasing order of $r_{ij}$
(02)  set $\{i,j,r_{ij}\}$ to be the first triplet of $R$
(03)  initialize $names$,$members$,$sizes$ to $N$ clusters of size 1
(04)  for $r = N \ldots 1$:
(05)     $\Delta T(r) \leftarrow 0$
(06)     while $r_{ij} = r$ do:
(07)        $u \leftarrow$ FIND($i$)
(08)        $v \leftarrow$ FIND($j$)
(09)        $\Delta T(r) \leftarrow \Delta T(r) + 2 \cdot sizes(u) \cdot sizes(v)/N(N-1)$
(10)        UNION($u,v$)
(11)        set $\{i,j,r_{ij}\}$ to be the next triplet of $R$
(12)     end-loop
(13)     if $\Delta T(r) > \delta$ then
(14)        store current partition ($names$,$members$,$sizes$)
(15)     end-if
(16)  end-loop
(17)  optional:  relabel small parts of stored partitions (see text)
(18)  return stored partitions
```

Figure 17: Efficient implementation of the second stage.

Line 17 is an optional heuristic to handle small, perhaps not interesting clusters. As discussed in Section 2.2, boundary and background points typically form small independent clusters at levels which are considered meaningful. In a top-down view, inspecting the obtained partitions from the smallest r value to the largest, we observe that beside the splitting that a cluster can undergo into two large clusters (a meaningful transition), it may as well gradually loose its boundary, or abruptly crash into small pieces. This happens when the certainty of labeling is too small, namely the values p_{ij}^r of the connecting edges have dropped below $\frac{1}{2}$. Hence there is a tradeoff: one may keep the uncertain points connected, at the cost of reduced certainty.

Our relabeling algorithm assumes that a minimal size of interest for a cluster is known, denoted S. After the typical cut at level r is found, we sustain the clusters whose sizes are at least S, and relabel the others. For this we scan the sorted list R *to its end*, and apply for every triplet $\{i,j,r_{ij}\}$ the operation `UNION(FIND(`i`),FIND(`j`))` if and only if the three following conditions hold:

- Either i or j (or both) belong to a small cluster (size $< S$);
- Nodes i and j were members of the same cluster at the former partition (with smaller r);
- Consistency is obeyed, namely when all the triplets having the same r_{ij} value are added at once, no node is doubly relabeled by being connected to more than one sustained cluster.

4.2 Robustness

Two aspects of robustness are discussed below, including robustness with respect to data perturbation in Sections 4.2.1, and robustness with respect to the clustering hypothesis in Section 4.2.2.

4.2.1 Data perturbation

Figure 18 shows a task involving the separation of two point sets generated by different statistical sources. The issue addressed here is the response of the various algorithms to the amount of noise in the data. The normalized cut algorithm performs well at low levels of noise (left column), but as the noise is increased it abruptly breaks done and returns false segmentation (right column). The factorization method behaves in a similar way, although it breaks done sooner (for lower levels of noise, left column). In comparison, as the level of noise is changed continuously, the performance of our algorithm degrades gracefully and continuously, without ever showing an "abrupt" breakdown as the spectral methods do; we believe that this is another beneficial result of the stochastic nature of our method.

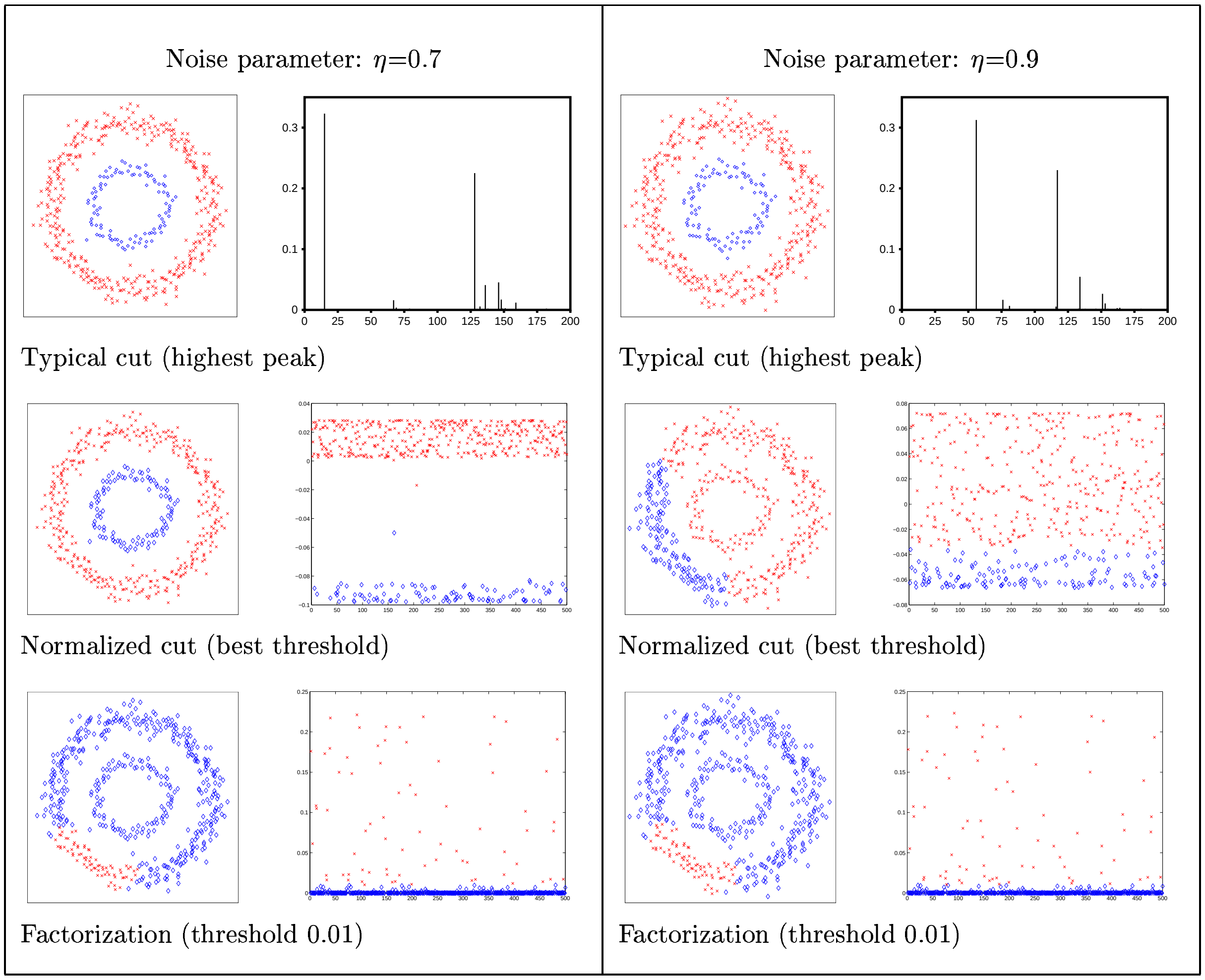

Figure 18: Robustness under data perturbation. The control parameter η defines the spread of the points, or the "width" of each circle. Results for two values, η=0.7 and 0.9, are shown: The factorization algorithm fails for both, the normalized cut algorithm fails for the the larger noise level, while our algorithm always finds the desired structure. For the spectral methods we show the entries of the relevant eigenvector next to the partition found. The magnitude of the entries are plotted versus their serial index. These methods seek a threshold which separates between small and large entries. The color and the symbol used for the eigenvector entries corresponds with those used in the two dimensional plot.

4.2.2 False structure

One problem with most clustering methods is that they look for structure and thus find something, even when the data is un-structured; this is often a single point solution obtaining the global optimum of some criterion, with the property that "similar" solutions (obtained from the chosen solution by a small perturbation) are far from optimal. Soft clustering methods avoid this pitfall, while the stochastic nature of our method, which essentially looks for a distribution over the solution space, also makes it robust against the identification of false structures. As an example, we studied the clustering of a complete graph (clique) where all the weights have roughly (but not exactly) the same weight. Our algorithm found a single significant transition, identifying two possible interpretations: either all the points are in one cluster, or they are all isolated. On the other hand, deterministic agglomeration methods suggested a solution for every number of desired clusters.

4.3 Probability bounds

The contraction scheme was originally invented as the core of a probabilistic minimal cut algorithm, which finds the capacity of the minimal cut with high probability by repeating the contraction procedure a sufficient number of times [22]. We quote here some results concerning the contraction algorithm, and the reader is referred to [22] for details. The probability of the algorithm to return a particular minimum 2-way cut is at least $\binom{2}{N} = \Omega(N^{-2})$, hence repeating it $O(N^2 \log N)$ times gives a high probability of finding the minimum value in some trial. This large number of trials is required to guarantee that the *minimum* is obtained. In our case we are not interested in the minimal capacity value, but rather in the pairing probabilities, which are average quantities. We have shown in Section 2.4 that in this case a sample of size $O((\log N)/\epsilon^2)$ is sufficient for an accuracy level ϵ with high probability.

Every cut value has a certain probability to be returned by the contraction algorithm, but the functional form of the induced probability distribution is unfortunately not known. There is a lower bound, however, on the probability that an α-minimal cut is returned (a cut whose capacity is at most α times the minimum). The probability to contract a crossing edge of a particular α-minimal cut is asymptotically less than $N^{-2\alpha}$ by the time that $r = \lfloor 2\alpha \rfloor$ meta nodes remain. Note that contracting a crossing edge of a cut prevents it from being returned by the algorithm, hence an upper bound on the probability to contract a crossing edge is a lower bound on the survival probability of the cut. The generalization to r-way cuts yields the probability $\Omega(N^{-2(r-1)})$ that a particular minimum r-way cut is returned (if contraction is stopped when r meta nodes remain).

5 Summary and Discussion

Our goal in this work was to apply cluster analysis as a unified approach for a wide range of vision applications. Many vision applications can be viewed as data partitioning problems, and can be addressed by this approach. In particular, we have focused on the tasks of image segmentation, edge elements grouping, and image organization. Thus in Section 3 we investigated experimentally the plausibility of our goal, and showed that using our clustering algorithm we can address different problems in computer vision.

The first step is to define the pairwise similarity between the visual entities: pixels, edgels, or shapes. The second step is to apply cluster analysis to detect internal structures among the visual entities (pixels, edgels or images). Hence we map the relevant entities to the nodes of a graph, while their pairwise relations are used to assign weights to the graph edges.

When the visual entities are represented as graph nodes and edge weights are assigned, our novel clustering method can be applied. The central insight to our algorithm is that it converts the pairwise similarity weights into higher order weights, or "collective" similarities. The collective similarity of two nodes i, j depends on an integer number r. It is the probability p_{ij}^r that i and j are on the same side of a random r-way cut which is generated by the contraction algorithm. The analysis of the contraction algorithm shows that these pairing probabilities are dominated by the low capacity cuts.

Nevertheless, the pairing probabilities are not determined by a single pivotal cut (e.g., the minimum cut), and at this point precisely our algorithm defers from most other clustering algorithms. The usual approach to clustering can be viewed as a search algorithm in some hypothesis space, where a hypothesis is a feasible partition. Whatever search method is applied, whether it is heuristic or guided by an objective function, a regular search algorithm ends in some point in the hypothesis space and returns it as a single possible solution (Section 4.2.2). On the other hand, our algorithm (and clustering algorithms which are inspired by statistical mechanics) induces a probability distribution over the hypothesis space, and returns an average solution. Thus we are not committed to a single partition, and our algorithm acquires a great amount of robustness. Indeed, we demonstrated in Section 4.2.1 that our algorithm is more stable than spectral algorithms under data perturbation.

The algorithm generates an average partition at each r level, which is called the typical cut. The definition of the typical cut relies on the interpretation of 1-p^r_{ij} as the probability that edge (i,j) is a crossing edge in an "average cut" (Section 2.2). The N typical cuts corresponding to $r = 1 \ldots N$ are the candidate solutions to the clustering problem, and a heuristic criterion is applied to select some of them as a hierarchy of partitions. Evidently, successful partitions can be selected only if the set of N candidate typical cuts *indeed contains* the desired solutions. Whether this is the case or not, it depends on the output of the first stage of the algorithm, which transforms the pairwise similarities into the pairing probabilities.

Acknowledgments

We thank Ben Kimia and Daniel Sharvit for 31 silhouette that are used in Section 3.4. We thank Avner Magen for pointing us to Karger's work, and Ido Bregman for helping in the complexity analysis.

References

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

[2] Boykov Y., Veksler O., and Zabih R., "Fast approximate energy minimization via graph cuts", *in Proc. of International Conference on Computer Vision*, 377-384, 1999.

[3] Cho K. and Meer P., "Image segmentation from consensus information", *Computer Vision and Image Understanding* 68, 72-89, 1997.

[4] Chung F.R.K, *"spectral graph theory"*, AMS Press, Providence Rhode Island, 1977

[5] Costeira J. and Kanade T., "A multibody factorization method for motion analysis", *in Proc. of International Conference on Computer Vision*, 1071-1076, 1995.

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

[7] Ennis D., "Modeling similarity and identification when there are momentary fluctuations in psychological amplitudes", *in Multidimensional Models of Perception and Cognition"*, Ashby F. Ed., Lawrence Erlbaum assoc. publishers, 279-298, 1992.

[8] Felzenszwalb P. and Huttenlocher D., "Image segmentation using local variation", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 98-104, 1998.

[9] Gdalyahu Y., Ph.D Thesis, "Stochastic Clustering and its Applications to Computer Vision", The Hebrew University in Jerusalem, April 2000.

[10] Gdalyahu Y. and Weinshall D., "Flexible Syntactic Matching of Curves and its Application to Automatic Hierarchical Classification of Silhouettes", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 21(12):1312–1328, December 1999.

[11] Gdalyahu Y., Weinshall D. and Werman M., "A randomized algorithm for pairwise clustering", *in Advances in Neural Information Processing Systems* 11, 424-430, 1998.

[12] Gdalyahu Y., Shental N., and Weinshall D., "Perceptual Grouping and Segmentation by Stochastic Clustering", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 367-374, 2000.

[13] Geman S. and Geman D., "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 6, 721–741, 1984.

[14] Guedalia I., London M. and Werman M., "An on-line agglomerative clustering method for nonstationary data", *Neural Computation* 11, 521-540, 1999.

[15] Guy G. and Medioni G., "Inferring global perceptual contours from local features", *International Journal of Computer vision* 20, 113-133, 1996.

[16] Herault L. and Horaud R., "Figure-ground discrimination: a combinatorial optimization approach", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 899-914, 1993.

[17] Hofman T., Puzicha J. and Buhmann J., "Unsupervized texture segmentation in a deterministic annealing framework", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 20, 803-818, 1998.

[18] Hofmann T. and Buhmann J., "Pairwise data clustering by deterministic annealing", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19, 1-14, 1997.

[19] Hu T.C., *"Combinatorial Algorithms"*, Addison-Wesley Publishing Company, Reading MA, 1982.

[20] Ishikawa H. and Geiger D., "Segmentation by grouping junctions", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 125-131, 1998.

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

[22] Karger D. and Stein C., "A new approach to the minimum cut problem", *Journal of the ACM* 43, 601-640, 1996.

[23] Kearns M. and Vazirani U., *"An introduction to computational learning theory"*, The MIT press, 1994.

[24] Kleinberg J., "Authoritative sources in a hyperlink environment", *in Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 668-677, 1998.

[25] Lance G. and Williams W., "A general theory of classification sorting strategies: II. clustering systems", *Computer Joutnal* 10, 271-277, 1969.

[26] Mumford D. and Shah J., "Boundary detection by minimizing functionals", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 22-26, 1985.

[27] Pal N. and Pal S., "A review on image segmentation techniques", *Pattern Recognition* 26, 1277-1294, 1993.

[28] Perona P. and Freeman, W., "A factorization approach to grouping", *in Proc. of European Conference on Computer Vision*, 655-670, 1998.

[29] Rosch E. and Mervis C., "Family resemblance: studies in the internal structure of categories", *Cognitive Psychology* 7, 573-605, 1975.

[30] Rose K., Gurewitz E. and Fox G., "Constrained clustering as an optimization method", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 785-794, 1993.

[31] Sarkar S. and Boyer K., "Quantitative measures of change based on feature organization: Eigenvalues and eigenvectors", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 478-483, 1996

[32] Scott G. and Longuet-Higgins H., "Feature grouping by relocalization of eigenvectors of the proximity matrix", *in Proc. British Machine Vision Conference*, 103-108, 1990.

[33] Sharon E., Brandt A. and Basri R., "Fast Multiscale Image Segmentation", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 70-77, 2000.

[34] Sharvit D., Chan J., Tek H. and Kimia B., "Symmetry based indexing of image databases", *J. visual communication and image representation*, 1998.

[35] Shashua A. and Ullman S., "Structural saliency: The detection of globally salient structures using a locally connected network", *in Proc. of International Conference on Computer Vision*, 321-327, 1988.

[36] Shepard R., "Toward a universal law of generalization for psychological science", *Science* 237, 1317-1323, 1987.

[37] Shi J. and Malik J., "Normalized cuts and image segmentation", *in Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 731-737, 1997.

[38] Thornber K. and Williams L., "Analytic solution of stochastic completion firlds", *Biological Cybenetics* 75, 141-151, 1996.

[39] Tishby N. and Slonim N., "Data Clustering by Markovian Relaxation and the Information Bottleneck Method",in *Advances in Neural Information Processing Systems* 2000.

[40] Tversky A., "Features of similarity", *Psychological Review* 84, 327-352, 1977.

[41] Weiss Y., "Segmentation using eigenvectors: a unifying view", *in Proc. of International Conference on Computer Vision*, 975-982, 1999.

[42] Williams L. and Jacobs D., "Stochastic completion fields: a natural model of illusory contour shape and salience", *neural computation* 9, 849-870, 1997.

[43] Wu Z. and Leahy R., "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 1101-1113, 1993.

[44] http:\\www.cs.huji.ac.il\~yoram\#software, http:\\www.cs.huji.ac.il\~daphna