

Stochastic Clustering and its Applications to Computer Vision

Thesis submitted for the degree “Doctor of Philosophy”

Yoram Gdalyahu

Submitted to the Senate of the Hebrew University in Jerusalem (1999)

This work was carried out under the supervision of
Prof. Daphna Weinshall.

Acknowledgments

Daphna, it was a great pleasure to be your student. You always encouraged me to be creative and to search for new directions and new ideas. Your judgment and critique was always sharp, hence it was especially pleasing to get from you positive feedbacks. I thank you for giving me so many of them, which was a great encouragement along the way. I thank you for being always available for me, always interested in the progress of the work. I do hope that our work will have a continuation, as well as our friendship.

I would like to thank all those who collaborated with me along the years, and especially Michael Werman, Ran El-Yaniv, Golan Yona, Noam Shental, Ido Bregman and Assaf Zomet. I thank the members of both the vision lab and the machine learning lab, past and present, for creating a lively and friendly atmosphere.

Contents

Abstract	iii
1 Introduction	1
1.1 The motivation and the goal of this work	1
1.2 Segmentation, perceptual grouping, and image retrieval	3
1.3 Properties of clustering algorithms	4
1.4 Pairwise (graph based) clustering: literature review	6
1.5 Our contribution, and the thesis organization	15
2 The Typical Cut Algorithm	17
2.1 Notations and Definitions	17
2.2 Outline of algorithm	18
2.3 Illustrative example	22
2.4 Complexity	26
2.5 Robustness	28
2.5.1 External parameters	28
2.5.2 Data perturbation	30
2.5.3 False structure	33
2.6 Cross Validation	34
2.7 Efficient implementation	37
2.7.1 Computing the level crossing of probability 0.5	37
2.7.2 Contraction of sparse graphs	39
2.7.3 The second stage	45
2.8 Probability bounds and relation to other work	46

3	Dissimilarity between Shapes	51
3.1	Curve matching: problem and related work	52
3.2	Flexible syntactic curve matching: algorithm	54
3.2.1	The proposed matching method	54
3.2.2	Matching results	58
3.2.3	Dissimilarity measurements: comparison	61
3.3	Flexible syntactic matching: details	62
3.3.1	Preprocessing and contour representation	63
3.3.2	Global alignment: pruning the starting points	63
3.3.3	Syntactic operations which determine the edit distance	65
3.3.4	Minimizing edit cost via dynamic programming	68
3.3.5	Potential: bounding the edit distance	70
3.3.6	Computing curve similarity after matching	71
3.3.7	Discussion and comparison to other methods	72
4	Results	76
4.1	Segmentation of intensity images	76
4.2	Segmentation of color images	78
4.3	Unstructured background and grouping of edge elements	79
4.4	Organization of an image database	88
5	Discussion	94
A	An Alternative Implementation of The Typical Cut Algorithm	99
B	Syntactic Operations for Curve Matching: Comparison	102
	List of Publications	105
	Bibliography	107

Abstract

This work applies cluster analysis as a unified approach for a wide range of vision applications, thus it combines two research domains: the domain of computer vision and the domain of machine learning. Cluster analysis, a fundamental problem of machine learning, attempts to recover the inherent structure within a given dataset. Many problems of computer vision have precisely this goal, to find which visual entities belong to an inherent structure in the image, or in the sequence of images, or in a database of images. For example, a meaningful structure in the context of image segmentation is a set of pixels which correspond to the same object in the scene. Cluster analysis can be used to partition the pixels of an image into meaningful parts, which correspond to different objects.

This framework clearly consists of two parts. The first is to define and measure for every pair of visual entities the likelihood that they belong to the same structure, or the pairwise similarity between them, and the second is to aggregate similar entities into clusters. In this work we focus on the problems of image segmentation, perceptual grouping of edge elements, and organization of an image database in shape categories. The visual entities to consider in each case are pixels, edge elements and images, respectively.

While natural measures exist for pixel similarity, and extensive work is done in the area of edgels similarity, shape similarity is not well understood. Our contribution in this work includes the development of a novel algorithm to measure shape similarity between images of objects. The primal property of this algorithm is that it is suitable for weakly similar shapes, hence it provides a graded measure of shape similarity. In addition, the algorithm is local, avoids using global shape characteristics which are sensitive to occlusion, and it is invariant under scaling and rigid transformations of the image.

Our second contribution in this work is a novel clustering algorithm, which is used as a unified platform to solve a variety of computer vision problems. In accordance with our general scheme, the

clustering algorithm uses pairwise representation, where the visual objects (pixels, edgels, images) are mapped to a nodes of an undirected weighted graph, with edge weights representing pairwise similarity relations. The clustering problem is formulated as a graph partitioning problem, namely a meaningful hierarchical partitioning of the nodes into clusters is thought.

The central insight to our clustering algorithm is that it converts the pairwise similarity weights into higher order weights, or “collective” similarities. To define the collective similarity of two nodes we induce a probability distribution over the set of all possible graph partitions. In other words, every r -way cut of the graph is assigned a certain probability, with higher probability assigned to low capacity cuts. Under this distribution, and for every integer r , the collective similarity of two nodes i and j is defined as the marginal probability p_{ij}^r that i and j are at the same side of a random r -way cut.

This transformation of the similarity weights incorporates information from the whole distribution over cuts. On the contrary, most other clustering algorithms select a pivotal cut, either by the minimization of a certain cost function or by the application of a certain heuristic. Since the pairing probabilities p_{ij}^r are not determined by a single pivotal cut (e.g., the minimum capacity cut) the dependence on the arbitrary chosen cost function is relaxed. Thus our algorithm is not committed to a single partition, and it acquires a great amount of robustness.

In addition to its robustness and the fact that our algorithm uses pairwise representation and avoids parametric modeling of the clusters, it is also very efficient. We provide a full analysis of its asymptotic running time, showing that for a sparse similarity graph consisting of n nodes, the asymptotic complexity is of order $O(n \log^2 n)$ for a fixed accuracy level.

Getting back to the computer vision applications, we demonstrate how our clustering algorithm can be used for segmentation of brightness and color images, for perceptual organization of point sets and edge elements, and for image database organization. The last application integrates our algorithm for shape similarity and our clustering algorithm. Specifically, we demonstrate our scheme using a database of 121 images of objects, extracting their silhouettes and measuring their mutual shape similarities. The images are then mapped to the nodes of a graph, with the pairwise shape similarities assigning weights to the graph edges. Our clustering algorithm is applied, and a hierarchical organization of the images in shape categories is obtained.

Chapter 1

Introduction

We present the goal of our work in Section 1.1, and relate it to fundamental problems of computer vision in Section 1.2. Background on clustering algorithms is provided in Sections 1.3 and 1.4. In light of the existing work we outline our contribution in Section 1.5, where we also describe the organization of the rest of the thesis.

1.1 The motivation and the goal of this work

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 entities of data (pixels, edgels, images). Nevertheless, they all come to serve a common goal, which is the partitioning of the visual entities into “coherent” parts.

Data partitioning, or *clustering*, is a vague concept to define, as 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 subsets, or by organizing the data in a hierarchy of sets, cluster analysis attempts to recover the inherent structure within a given dataset. Data clustering is a fundamental problem in unsupervised machine learning, with a strong connection to cognitive science [112, 137, 124, 125].

The goal of this work is to use cluster analysis as a unifying principle for a wide range of vision

problems. 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-circuitry and good continuity. High level attributes may be as complex as 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 the organization of the 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 with a common topic.

Partitions of these kinds are associated with different levels of image understanding. It is important to distinguish this 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, a form of understanding.

Note that according to the *appearance based approach* to object recognition, an object is represented by a collection of images, which in some sense span the “image space” of that object. Our method can be used to divide the appearances of an object into clusters of similar views, in order to assist the construction of an appearance based representation.

The prohibitively large amount of raw data is a characteristic of many vision applications. In addition to the problem of compact representation of the raw data, which we have considered above, there is the problem of efficient processing of the sheer amount of data. Algorithms that use low level image features, such as pixels and edge elements, can benefit from dividing the raw image data into parts that can be processed separately. Image segmentation and perceptual grouping of

low level elements are therefore also motivated by the need to design more efficient algorithms. In the analysis of image databases, a related problem appears. Since we are typically dealing with a large number of images, it is often necessary to divide them into subsets that can be processed separately during image retrieval.

1.2 Segmentation, perceptual grouping, and image retrieval

Segmentation, perceptual grouping, and image retrieval are fundamental problems of computer vision. It is evident that the extensive study in these areas cannot be surveyed in a short section. However, to put our work into context, we focus on some of the more recent or classical methods. Many image segmentation algorithms are reviewed in [102, 51]. The classical techniques include gray level thresholding, region growing and recursive splitting, relaxation by the Markov Random Field approach (originated in the work of [40]), relaxation by neural networks, and variational formulations (e.g., [9, 98]). More recently, several authors considered image segmentation as a direct application of pairwise clustering. This includes the usage of objective functions which are minimized by graph theoretic methods [152, 127], deterministic annealing [54], multi grid techniques [121], and non local Markov chains [13]. Orthogonal to these approaches, which try to minimize a certain cost function, are the methods which are built upon local and greedy aggregation [29, 15] or upon spectral decomposition of the similarity matrix [109, 147].

Image segmentation may be regarded as perceptual grouping of pixels into homogeneous parts. However, the term perceptual grouping is usually associated with mid level vision, and more specifically with grouping of edge elements. It is frequently formulated as a saliency detection problem, where the goal is to assign a value to each edge, representing to what extent it is a part of a shape or a part of background noise. Given the distribution of saliency values, it is often possible to choose a threshold which will segment the edges into shape and noise classes. Saliency is a measure of non-accidental mutual properties of visual clues, such as collinearity, curvilinearity, termination, crossing, parallelism, convergence, equal spacing and more. The history of the taxonomy of non-accidental properties goes back to the Gestalt psychologists at the beginning of the 20th century.

Perceptual grouping methodologies consist of two parts: (i) combination of properties which are usually non accidental into an affinity measure, and (ii) detecting the internal structure, or

shape, by computing saliency or by direct segmentation. The second step is the clustering part of the grouping methodology. Among the methods of choice for this second step are relaxation neural networks [123, 1], cost minimization using simulated annealing [53], spectral decomposition of the affinity matrix [117, 49], and stochastic completion fields [149, 150].

The perceptual organization of high level visual entities, such as images, must face with the highly complex problem of estimating the similarities between such entities. There are clearly different dimensions of similarity between high level features, and even one dimensional measures are hard to define. One alternative, whose perceptual significance is doubtful but is relatively easy to compute, is to compare color histograms computed from the entire image (e.g., [132, 34, 50]). For images of objects, the dimension of shape seems to be mostly relevant to their perceptual organization [142]. In fact, part of this work is concerned with precisely this issue, hence we postpone the discussion of related work on shape similarity to Chapter 3.

A measure of similarity between images is essential for image retrieval applications. Indeed, many of the proposed measures are integrated into retrieval systems, such as QBIC [32], SQUID [96], Blobword [7], Surfimage [100], Virage [48], Photobook [107], and more. Given a high level similarity measure for images and a clustering algorithm, it is possible to organize an image database into categories of interest. 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. However, although it is a natural extension of the organization principles of lower vision levels, this kind of image organization is not well studied in the literature.

1.3 Properties of clustering algorithms

Clustering algorithms are employed in diverse variety of problems, and consequently show large divergence themselves. The principal properties that distinguish different clustering algorithms are listed in the short survey below. In the next section we concentrate on the class of pairwise algorithms, which are more relevant to the current work.

Representation.

Two different types of data lead to two different representations. Either the data items are mapped to some real normed vector space (called *feature space*), or they are mapped to the nodes of a weighted graph, with edge weights representing similarity or dissimilarity relations. The second

form, called “pairwise representation”, lacks geometrical notions such as scatter and centroids. However, it has the advantage that no feature selection is required, which is often an elusive task. Moreover, pairwise relations may violate metric properties such as the triangular inequality, a situation which cannot be modeled when data is embedded in a vector space. The same is true with respect to symmetry violation, which can be represented by a directed graph.

Objective function.

The clustering problem can be formulated as a discrete optimization problem, in which case it involves two distinct steps: (i) determine some suitable cost function over the partition space; (ii) compute a partition which minimizes the cost. The cost function may reflect a measure of point scatter or graph connectivity, depending on the representation used. However, defining a meaningful cost is task dependent, and the search for its global extremum is in general computationally intractable. Alternative approach is to rely on well motivated heuristics, like agglomeration, local density estimation or spectral decomposition (see below).

Parametric modeling.

The assumption which underlies the modeling approach is that the given data to cluster is generated by some statistical source (e.g., mixture model). The statistical source is assumed to depend on several (unknown) parameters. The clustering problem is then reduced to a parameter estimation problem, which may be addressed by iterative optimization techniques. Classical examples are the k -means and, more generally, the EM algorithms. Customarily, the statistical source is assumed to be stationary and of a density mixture type, and the first one or two statistical moments are used. This is equivalent to Gaussian mixture modeling of the data.

Exclusiveness.

An exclusive (hard) partition of the data assigns each data item to a unique cluster, while an inclusive (soft) partition assigns each data point, with some probability, to every cluster. Soft assignment is typically associated either with parametric modeling or with a global cost function. In the first case, the probability of a point being assigned to a cluster is the posterior probability of being generated by the corresponding statistical source. In the second case, the soft assignment is a weighted average over all possible assignments, each weighted by its global cost function.

Scale (hierarchical versus partitional clustering).

The problem of scale is the problem of the “correct” number of clusters. This problem is associated with the well known information-modeling tradeoff, and a “correct” answer cannot be defined universally. However, clustering algorithms may produce a hierarchy of solutions in order to reflect multi scale interpretations of the data. Hierarchical partitions can be generated in a number of ways: agglomeration, recursive partitioning, and generalized cost functions which are defined over partition trees instead of over single partitions [94, 114]. The problem of clustering validation is intimately connected with the problem of scale.

1.4 Pairwise (graph based) clustering: literature review

A weighted graph is a set of nodes, a set of edges connecting them, and a function that assigns weight to each edge. For pairwise clustering the weights are positive real numbers, and they reflect (dis)similarity values between pairs of datapoints, which are mapped to the nodes. This section reviews clustering methods which are formulated as graph partitioning. We review various algorithms only briefly, emphasizing possible connections between them. We consider agglomerative and spectral heuristics, discrete optimization formulations, dynamical system approaches, and models from statistical mechanics. We also motivate the important notion of higher order (“collective”) similarity values.

Embedding

Metric dissimilarity relations d_{ij} between datapoints may be transformed to vectorial representation. Consequently, instead of solving the graph partitioning problem, one faces with the embedding problem followed by the application of vectorial clustering (using an algorithm which may use such geometrical notions as centroids and scatter). The embedding problem is to map each node (datapoint) i to a vector v_i in some real normed space, such that $\|v_i - v_j\| = d_{ij}$. It can be shown that the mapping that assigns $v_i \leftarrow [d_{i1}, \dots, d_{in}]$ embeds n points in \mathcal{R}^n with the max norm ℓ_∞ . However, n is usually very large and dimensionality reduction is often required, where possible methods are principal component analysis [33], random projection [68], principal curves [52] and others, thus introducing distortion into the low dimensional representation.

A low dimensional graph embedding with controlled distortion is proposed in [88], and used for

clustering in [87]. An alternative method, called multi dimensional scaling, is applied in [80, 27, 83, 125], where the embedding might preserves the ranking of the pairwise distances, not necessarily their ratios. We note that dimensionality reduction is another name for the problem of feature selection, namely it involves the assumption that a small number of features can be found, which describe the datapoints with sufficient accuracy. Moreover, even if this is the case, it is not at all guaranteed that a specific geometrical moment, like centroid, is a useful feature (consider two concentric circles for example).

Collective similarity measure

In very simple clustering problems the similarity values between clusters are much smaller than those within clusters. Hence there exists a threshold θ , such that if all edges with similarity weight smaller than θ are removed, the graph is disconnected into components which represent the desired clusters (the generalization to k -connected components or cliques is computationally less attractive). Real problems are rarely that simple. The objective of clustering is usually more global, while edge thresholding is purely local. However, if the pairwise relations are replaced by higher order relations, then they start to reflect global properties and the simple connected components algorithm becomes surprisingly effective.

High order relations (which we call “collective” or “transitive” relations) can be defined in various ways. A simple and direct approach is taken in [130, 44], where the dissimilarity value between nodes i and j is transformed to $m+n$ if node i is the m th nearest neighbor of node j , and node j is the n th nearest neighbor of node i . A much more complicated transformation is hidden in the min-cut algorithm for clustering (see below), as it is known from the Gomory-Hu theorem [59] 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. Our work is inspired by the successful definition of collective similarities made in [10], where an analogy between the clustering problem and the behavior of a magnetic system is proposed. For more details see Section 2.8.

Let us define the similarity profile of a node as the vector of all its pairwise relations. Hence $p_i = [d_{i1}, \dots, d_{in}]$ is the profile vector of node i , where d_{ij} are either similarity or dissimilarity relations. For simplicity we assume that d_{ij} are dissimilarity relations, but they may violate metric

properties such as the triangular inequality. Hence the distance between profile vectors $\|p_i - p_j\|$ is not necessarily equal to the pairwise distance d_{ij} , as in the embedding procedure above. On the contrary, the distance between two profile vectors p_i and p_j might be large in spite of d_{ij} being small, if the two data items i and j do not share similar neighborhoods. This means that by taking global considerations into account, through the collective profile distance, a simple thresholding algorithm can potentially assign nodes i and j to different clusters, although their direct (pairwise) connection indicates the opposite. The collective dissimilarity measure (between profile vectors) can be defined in a number of ways. We have experimented with the ℓ_p norms of the vector difference and with statistical correlation in a related work concerned with supervised learning [64, 145], and we have experimented with ℓ_p norms and the Jensen-Shannon divergence in a recent clustering work [25].

For completeness we refer to a similar collective manipulation which is used by clustering algorithms that act on vector spaces. The basic idea behind the *distributional clustering* approach [108, 55] is to represent each data item by a probability distribution over features with which it co-occurs. The Kullback Leibler divergence between the co-occurrence distribution then serves to measure the dissimilarity between objects and cluster centroids..

Agglomeration versus objective functions, and the minimal cut

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 most similar clusters are merged together, until the similarity of the closest clusters is lower than some threshold. Different similarity measures between clusters distinguish between different agglomerative algorithms. In particular, the *single linkage* algorithm defines the similarity between clusters as the maximal similarity between two of their members, and the *complete linkage* algorithm uses the minimal such value. A very general class of agglomerative algorithms is formalized in [85].

Agglomerative heuristics lack the ability to separate between the problem definition and algorithmic solution, while in the discrete optimization approach (Section 1.3) the design of proper a cost function is ill posed. Greedy agglomeration can also be connected with global properties [29], or even with the global cut capacity cost function [99]. Namely, the particular agglomeration scheme defined in [99] minimizes the cut capacity, defined as the sum of the positive pairwise similarities between members of different clusters (see also Section 2.1). The minimal cut clustering algorithm

defines the optimal partition as the one with minimal capacity [152, 69]. For bi-partitions, which split the data into two parts, this optimization problem is extensively studied and is solved in polynomial time. However, clustering applications frequently involve very large graphs, and the exact solution becomes impractical.

The minimal cut clustering algorithm minimizes the total similarity weight between clusters, while maximizing the total similarity weight within clusters. With dissimilarity relations, the analog cost function sums the dissimilarity weights *within* clusters. In [56] there is an attempt to axiomatize the definition of pairwise cost functions, and it is shown that if the total dissimilarity weight within each cluster is normalized by the cluster size (the number of nodes it contains), then additional invariance and robustness properties are satisfied. In particular, the partition which minimizes this cost is invariant under linear shift of the data (a desirable property especially when the data is defined on interval scale), and it is not dominated by a small number of links. The last observation points to a major disadvantage of the capacity (cut) cost function, namely, that it is biased towards partitions which disconnect small number of edges, and may even separate a single node from the graph. This led to the proposal of another cost function which is called “normalized cut” [127], see below. It resembles the bisection cost function from graph theory, which is the number (or total weight) of edges between two equal size parts of the graph [12, 75, 30].

Many other cost functions can be defined. The cost definition is influenced by the possibility to design an efficient algorithm which optimizes it. Optimization via graph cuts of cost functions with a regularization term is proposed in [13, 63]. Cost minimization using multi grid method is found in [121].

Spectral methods

Spectral methods identify good partitions via the eigenvectors of the similarity matrix, or other matrices derived from it. Although one important instance of these methods, which uses the Laplacian matrix, can be motivated by a continuous approximation to a discrete optimization problem [17], in general spectral methods are not associated with any global cost function and can be thought of as useful heuristics.

Consider the adjacency matrix A of a directed graph, where $A[i, j] = 1$ if there is an edge directed from node i to node j , and $A[i, j] = 0$ otherwise. The singular vectors of A are, by definition, the eigenvectors of the symmetric matrices AA^T and $A^T A$. Note that the entry $[i, j]$ of

AA^T is the number of nodes which both i and j are incident on, while the entry $[i, j]$ of $A^T A$ is the number of nodes from which there is an edge to both i and j . Thus the matrices AA^T and $A^T A$ can be considered as collective similarity matrices, in the sense defined above.

In accordance, [78] made the following observation about the values of the entries in the singular vectors of A : If entry i of the principal eigenvector of $A^T A$ is large, then node i is considered “authoritative”; if entry i of the principal eigenvector of AA^T is large, then node i is a “hub”. Authorities and hubs exhibit a mutual reinforcement relationship: a good authority is a node that is accessed by many good hubs, while a good hub is a node that points to many good authorities. In the application discussed in [78], hub and authority levels are used to measure the importance of Web pages. It generalizes impact indices which are used in the field of bibliometrics, or citation analysis. The relation with clustering is that the joint set of authorities and hubs might be a dense subgraph, which is sparsely connected to the rest of the graph.

In case that A is symmetric, its eigenvectors and the singular vectors are the same, and the method reduces to the computation of the principal eigenvector of the adjacency matrix. A similar approach is taken in [109] with symmetric similarity matrices, constructed from real values in $[0, 1]$. The principal eigenvector v is treated as an indicator function: a threshold θ is chosen, each node i is assigned to one part if $v[i] > \theta$ and to the other part otherwise. In the rest of our work we will refer to this algorithm as the “factorization method”. Related work include [117] from the computer vision domain, and [110, 39] from the bibliometrics domain.

The use of non principal eigenvectors can sometimes add more power to the algorithm. Assuming from now on that A is symmetric, let V denote the $n \times k$ matrix whose columns are the k largest eigenvectors¹ of A . As shown in [147], the algorithm described in [19] effectively uses the matrix $Q = VV^T$ for grouping. If the rows of V are normalized, namely $\hat{V} = H^{-1}V$ where H is a diagonal $n \times n$ matrix with $H[i, i] = \|V[i, \cdot]\|$, then [119] 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. We note that in this case, \hat{Q} should be regarded as an improved (collective) similarity matrix, in the sense discussed above, and it remains to be seen whether the algorithm can be applied recursively to \hat{Q} (see our discussion on dynamical systems below)².

¹We use the terms largest and smallest eigenvectors to refer to the eigenvectors which correspond to maximal and minimal eigenvalues, respectively.

²A real $n \times n$ matrix A is said to be completely positive if there exists a real nonnegative $n \times k$ matrix B such

It is instructive to make a comparison with vectorial methods. Instead of the square similarity matrix A , vectorial representation uses an $n \times m$ feature matrix, with entries representing the endorsement of each item by each feature. The spectral decomposition of the feature matrix is widely used for dimensionality reduction, e.g. [21]. If the data is clustered, one hopes to project it onto well separated dimensions, which correspond with the data clusters. A certain model, from the domain of document retrieval, is studied analytically in [103].

The last spectral method which we consider is the *normalized cut* algorithm [127]. We first give our own formulation of the algorithm. Given a symmetric similarity matrix A , define D to be a diagonal matrix with $D[i, i] = \sum_j A[i, j]$. The operation $D^{-1}A$ normalizes the sum of each row in the similarity matrix to one, hence a vector whose entries are all 1's is an eigenvector of $D^{-1}A$. Moreover, it is the principal eigenvector, with the largest eigenvalue (which is 1). The normalized cut algorithm uses the next principal eigenvector, denoted v_2 , in order to partition the nodes into two parts; if $v_2[i] > \theta$ then node i is in one part, otherwise it is in the other. This formulation clarifies the relation with previous methods. See [147] for the relation with the principal eigenvectors of $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.

The name of the method, “normalized cut”, reflects the intuition which motivated it. The original formulation [127] uses the eigenvectors of the Laplacian matrix, defined as $D - A$. The relations between the spectral properties of the Laplacian matrix and the connectivity of the graph were first investigated in [31, 24]. In particular, finding a partition that minimizes a certain cost function is a discrete optimization problem, which depends on the connectivity of the graph. But when the cost function is defined to be the normalized cut cost (see below), then the discrete solution may be approximated by a real valued solution of the generalized eigenvalue problem $(D - A)v = \lambda Dv$. Namely, the vector v_2 which correspond to the second smallest λ is the real approximation to the discrete problem. Note that this vector is the eigenvector of $D^{-1}A$ with second largest eigenvalue, as presented above. In fact, there is no guarantee that the real solution will bear any relationship with the discrete one, and as far as we know the quality of the approximation has never been examined experimentally.

For completeness, we quote the definition of the normalized cut cost function. Let the set V of

that $A = BB^T$. In this case, if $A[i, j]$ can be interpreted as the probability that nodes i and j are in the same cluster, then $B[i, l]$ is the probability that node i is in cluster l . So far, however, the characterization of completely positive matrices is not well understood, as well as the possibility to approximate a given proximity matrix by a completely positive one [45, 116, 101].

nodes be split into two disjoint sets R, S . The cost $Ncut(R, S)$ of the partition is:

$$Ncut(R, S) = \frac{f(R, S)}{f(R, V)} + \frac{f(R, S)}{f(S, V)}$$

where $f(X, Y) = \sum_{i \in X, j \in Y} A[i, j]$. Since R and S are disjoint, each numerator is the cut capacity (sum over edges which cross the cut). In the denominator the terms $f(R, V)$ and $f(S, V)$ are called the associations of R and S , respectively. They act as penalties for unbalanced cuts.

Spectral methods and dynamical systems

Given a system which is described at time t by a state vector $v(t)$, defined up to scale, a discrete time dynamics is an operator \mathcal{O} which transforms $v(t)$ to $v(t+1)$. Linear dynamics is $v(t+1) = Mv(t)$ with some matrix M . A fixed point is a vector v^* which is invariant under the dynamics, and for a linear system it is clearly an eigenvector of M . The power method for calculating the principal eigenvector of M starts from a state $v(0)$ which is not orthogonal to the eigenvector, and recursively applies the dynamics until convergence. To find the k principal eigenvectors, the power method starts with k initial vectors (subjected to a similar condition), and keeps them orthogonal (e.g. by Gram Schmidt procedure) at every iteration. The orthogonalization introduces non linearity into the computation of the non principal eigenvectors. For exact formulation see [42].

We have shown above how the principal eigenvectors of a certain matrix M are associated with graph partitioning. The matrix M might be the symmetric similarity matrix A [109], the normalized (non symmetric) matrix $D^{-1}A$ [127], or the matrices AA^T and $A^T A$ for directed graphs [78]. Thus spectral methods can be understood on the basis of general dynamical systems, namely, they compute an indicator function which is the fixed point of a certain linear system.

A generalization to non linear dynamics is proposed in [41]. In analogy with the computation of non principal eigenvectors in linear systems, the algorithm of [41] iteratively applies non linear dynamics to several state vectors, and after each iteration uses a Gram Schmidt procedure to keep them orthogonal. This generalizes the concept of non principal eigenvectors to non linear systems.

It is interesting to view in this light the algorithm proposed in [106] for the maximal clique problem. One could argue for using this criterion for clustering, when similarity relations are binary and one requires all the members of the cluster to be similar to each other. The algorithm proposed in [106] iterates a state vector, using non linear “replicator dynamics”. The fixed point vector is a bi-valued vector, which indicates which nodes are in the maximal clique. See [104] for

the relation to non linear relaxation labeling [115].

Getting back to the general formulation of a dynamic system, if a fixed point v^* exists, then it is the limit of $\mathcal{O}^t v$ when $t \rightarrow \infty$. This gives motivation to consider the operator \mathcal{O}^t at this limit. As early as in 1968 an algorithm called “iterative intercolumnar correlation analysis” was proposed [93], and further developed in [82]. It can be directly motivated by our concept of collective similarities above. Namely, if a column of the proximity matrix is interpreted as a similarity profile with respect to all data points, we may construct a new proximity matrix whose $[i, j]$ entry is the inter column similarity between the two profiles, and repeat this process iteratively. The column similarity in [93, 82] is measured by statistical correlation.

More formally, construct the matrix \hat{A} by subtracting from each column of A its mean, and consider the operator $\mathcal{O}(A) = \hat{D}^{-\frac{1}{2}} \hat{A}^T \hat{A} \hat{D}^{-\frac{1}{2}}$, where $\hat{D} = \text{diag}(\hat{A}^T \hat{A})$. A sufficient condition is proved in [82], which guarantees that the matrix $\mathcal{O}^t(A)$ converges to a bi-valued matrix A^* with either +1 or -1 entries. The fixed point matrix A^* is permutation equivalent to a block diagonal matrix, with two diagonal blocks of sizes k and $n - k$ containing the +1 entries, and two off diagonal blocks containing the -1 entries. This indicates a partition of the data into two clusters. The sufficient condition which is shown in [82] to guarantee the convergence, is that $\max |A[i, j] - A^*[i, j]| < (-1 + \sqrt{1 + 4e^2})/2e$, where $e = 8(k/n)(1 - k/n)$.

Independently we have recently experimented with a similar scheme, using different intercolumnar similarity measures [25]. We found a similar behavior, indicating that the convergence to a fixed point might be a general property of collective similarities, and not a property of the specific correlation type measure.

Spectral methods and stochastic processes

Let us assume that the matrix M in the linear dynamics $v(t+1) = Mv(t)$ is stochastic, namely, its columns sum to one. For example, the transpose of $D^{-1}A$ used by the normalized cut algorithm is stochastic. In this case we may interpret the dynamics as a stochastic traversal on the graph, $M[i, j]$ being the probability of going from node j to node i . The principal eigenvector of M , properly normalized, then describes the stationary probability of visiting each node³. A spectral method which uses the principal eigenvector of a stochastic matrix can therefore be interpreted as separating the nodes having large visiting probability from the nodes having low visiting probability.

³The rate of convergence to the stationary probability is associated with the second eigenvalue of M , see e.g. [97].

We now wish to make a connection between the visiting probability and a global cost function, which is defined over graph partitions. For this we consider a much larger graph, where each node represents a feasible *partition* of the data. We refer to a node in this graph as a *configuration*, while the term “node” is kept for the original graph. The number of configurations is exponentially large, for example 2^n configurations if we consider all partitions of n nodes into two parts. The goal is to introduce dynamics which respects the cost function and converges to a steady state, where a configuration visiting probability depends only on its cost.

Assuming this goal has been achieved, one is able to approximate quantities of interest. One can translate the configurations visiting probabilities into nodes visiting probabilities, or one can estimate the stationary probability of two nodes being in the same cluster. The approximation is carried out by simulating the dynamics, and counting the the number of times that the event of interest occurs. In [10] the pairing probability of every two nodes being in the same cluster is used as a collective similarity measure, which replaces the original pairwise similarities.

This clustering method includes the following steps: (i) define a cost function, (ii) select a cost dependent probability distribution over partitions, (iii) design a stochastic dynamics which converges to the desired distribution, and (iv) simulate the dynamics and count events of interest. The cost dependent probability distribution at step (ii) is chosen to be Gibbs distribution, constrained to yield some fixed average cost through its temperature parameter. Using this distribution is inspired by statistical mechanics, and supported by the principle of least commitment [66], see below.

It is beyond our scope to describe dynamical schemes which respect the required stationary probability distribution. A simple method, however, which became quite popular outside the physics community, is the Metropolis algorithm. The Metropolis algorithm picks a configuration at random and accepts or rejects it based on its relative cost (energy). A randomly picked configuration is accepted if it lowers the cost, and it is accepted with some (temperature dependent) probability otherwise. In case of rejection, another configuration is randomly picked. Gradual decrease of the temperature is called simulated annealing [77], whose usage in vision was popularized by [40]. See [141] for minimizing a balanced cut cost function with simulated annealing.

Efficiency is a major problem with stochastic simulations, as the size of the configuration graph is exponential in n , and we require convergence to stationary probability in polynomial time. A process having this property is called rapid mixing [67]. A fast simulation process which is suitable

for multiway cut cost function, and which is conjectured to be polynomial, is the Swendsen-Wang algorithm [144]. It is used by the SPC clustering method [10, 151] (see the beginning of this introduction and Section 2.8).

A direct optimization approach would seek the most probable configuration, which has minimal cost. However, the disadvantage of this approach is that it commits to a single solution. On the other hand, replacing the pairwise relations by the pairing probability of nodes, as suggested in [10], exploits the information from the entire distribution over the configuration space and is therefore more robust, see also Section 2.5. Inducing the Gibbs distribution over the configuration space is supported by maximal entropy inference, as this distribution is the flattest one under the constraint of fixed average cost.

We note that deterministic annealing methods [113, 114, 57] postulate a parameter dependent Gibbs distribution over the configuration space, and optimize the parameters using EM method. See [141, 153, 54] for the relation with mean field approximation and neural networks. The deterministic approach avoids the stochastic simulation, but necessitate parametric modeling of the clusters (see “parametric modeling”, Section 1.3). As a consequence, these methods are usually suitable for vectorial representation, with [57] as an exception.

1.5 Our contribution, and the thesis organization

In this work we present a novel clustering algorithm, and apply it to problems of image segmentation, perceptual grouping of edge elements and image database organization. For the later task we develop a novel dissimilarity measure for object images, which compares the shapes of the objects in the images. This measure allows for the organization of image database into shape categories.

Our novel clustering algorithm is a pairwise hierarchical algorithm, which is 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 [10]. Like in the algorithms which apply stochastic simulation of a certain dynamics, we also generate a sample of configurations, and we count the number of events when two nodes are in the same cluster. This leads to the definition of a powerful collective similarity measure, which is the pairing probability of each two nodes.

However, unlike the methods which simulate dynamics, our algorithm does not use a Markovian process to generate the sample of configurations. Instead, we apply another sampling tool, originally developed as the core of a probabilistic minimal cut algorithm [72]. 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 an improve in efficiency, we make an elegant connection between cut based algorithms and agglomerative algorithms. Regarding efficiency, our algorithm can be analytically analyzed, and for sparse graphs and fixed accuracy level it runs in $O(n \log^2 n)$ time, where n is the number of datapoints.

From the point of view of vision applications, our contribution is the presentation of a unified approach for low, middle and high level vision problems. For the application of our method to high level organization of image databases, we suggest a novel algorithm which extracts shapes from images and estimates their dissimilarity. Our algorithm relates the image dissimilarity with the residual distances between matched feature points which are automatically extracted on the objects' boundaries. The bottleneck of this approach is the task of matching the feature points in a perceptually plausible manner. We achieve this goal by improving known syntactic matching techniques. In addition, we present an algorithm which is invariant with respect to substantial deformations and arbitrarily large scaling and rigid transformations of the image.

Thesis organization

The rest of this work is organized as follows. Chapters 2 and 3 describe our methods. The novel clustering algorithm is described in Chapter 2, and the algorithm to measure dissimilarity between images, whose core is a novel curve matching algorithm, is described in Chapter 3. Our results are summarized in Chapter 4, where we apply our clustering algorithm to intensity and color image segmentation, perceptual grouping of edge elements, and integrate our two algorithms into one application of image database categorization. A concluding discussion follows in Chapter 5.

Chapter 2

The Typical Cut Algorithm

This chapter presents our novel clustering method, which is used in chapter 4 for a few vision applications. The general approach and the principles of our method are described first. 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 worked out in details in Section 2.3. Fundamental aspects in the complexity analysis of the algorithm are discussed in Section 2.4, while full analysis is postponed to Section 2.7. In Section 2.5 we use a few synthetic examples to demonstrate the robustness of our method. Section 2.6 contains a novel approach which we have developed for validating partitional structures, whose general idea is applicable for other algorithms of pairwise clustering. In Section 2.7 we discuss the efficient implementation of our algorithm, and complete its complexity analysis. The relations between our algorithm and others are investigated in Section 2.8. There we also quote the known properties of the probability distribution that is imposed by the contraction algorithm (introduced in Section 2.2).

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, \dots, 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 ($\alpha = 1 \dots r$) 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 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 implementation issues which arise from considerations of space and time complexity, and emphasizing the general principles. The algorithm is divided into two stages, described in pseudo-code in Figures 2.1, 2.3 and explained below.

For a given value of r ($r = 1 \dots N$) our algorithm generates a sample of M possible r -way cuts, and use this sample to estimate the probability p_{ij}^r that (i, j) is not a crossing edge of a random r -way cut. The pseudo-code in Figure 2.1 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_{ij}^r that they are on the same side.

```

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

 $s_{ij}^r \leftarrow 0$  for  $i, j, r = 1 \dots N$  (initialize counters)
for  $m = 1 \dots M$ :
   $G'_N \leftarrow G(V, E)$ 
  for  $r = (N - 1) \dots 1$ :
     $G'_r \leftarrow \text{CONTRACT}(G'_{r+1})$  (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
           $s_{ij}^r \leftarrow s_{ij}^r + 1$ 
        end-if
      end-loop
    end-loop
  end-loop
end-loop
 $p_{ij}^r \leftarrow s_{ij}^r / M$  for  $i, j, r = 1 \dots N$  (compute empirical probabilities)
return array  $p$ .

```

Figure 2.1: Pseudo-code which transforms similarity weights into pairing probabilities.

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 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. A simple (non efficient) way to implement the stochastic edge selection is described in Figure 2.2.

The contraction procedure is the cornerstone of our method, since it defines the sample of M cuts, according to which the empirical probabilities p_{ij}^r are computed. Thus the contraction procedure is our sampling tool, assigning higher probability to cuts with lower capacity as is shown in [72]. In fact, [72] 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

```

procedure CONTRACT
input:   graph  $G'_k$  with  $k$  meta-nodes.
output: graph  $G'_{k-1}$  with  $k-1$  meta-nodes.

let  $e$  be an index to the set of edges of  $G'_k$  ( $e = 1 \dots \#e$ ).
 $S \leftarrow \sum_{e=1}^{\#e} w_e$ .
select  $x$  uniformly at random from  $(0, S]$ .
find minimal  $e_0$  such that  $\sum_{e=1}^{e_0} w_e \geq x$ .
merge the two meta-nodes  $A_\alpha$  and  $A_\beta$  of  $G'_k$  that are connected by the edge  $e_0$ .
remove self loops resulting from previous connections between  $A_\alpha$  and  $A_\beta$ .
return the resulting graph.

```

Figure 2.2: Simple implementation of the procedure CONTRACT

estimate p_{ij}^r – 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 Stirling number of the second kind, denoted $\tau(r, N)$. Let $\alpha(r) = 1 \dots \tau(r, N)$ be an index to 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_{ij}^α to be 1 if the edge (i, j) crosses the cut α and 0 otherwise. It is readily seen that

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

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 $1 - p_{ij}^r$ 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 $p_{ij}^r < 0.5$. To find the typical cut for every integer r between 1 and N we first remove all the edges whose transformed weight p_{ij}^r 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.

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

$$T(r) = \frac{2}{N(N-1)} \sum_{i>j} N_i N_j \quad (2.1)$$

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 clusters 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 $\Delta T(r) > \delta$.

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, where the wisdom of our method lies. The simple heuristic that is applied in the second stage (Figure 2.3) is based on the assumption that a good set of candidates is given.

The code line denoted with (*) in Figure 2.3 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 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_{ij}^r , thus letting the small clusters join the larger ones. More details on the optional relabeling procedure are given in

```

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

for  $r = 1 \dots N$ :
    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
             $E \leftarrow E \setminus (i, j)$  (remove the edge from E)
        end-if
    end-loop
    find connected components  $(A_1, A_2, \dots, A_s)$  in  $G(V, E)$ .
    compute  $\Delta T(r) = T(r) - T(r - 1)$  (use Equation (2.1))
    if  $\Delta T(r) > \delta$  then
        (*) relabel small parts (see text)
        report partition  $(A_1, A_2, \dots, A_s)$ .
    end-if
end-loop

```

Figure 2.3: Pseudo-code which finds typical cuts, measures the resemblance between subsequent cuts, and reports the “meaningful” partitions.

Section 2.7.3.

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 2.4a. 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 [125, 126, 28]. We use the same functional form as in [10, 109, 127], namely $w_{ij} =$

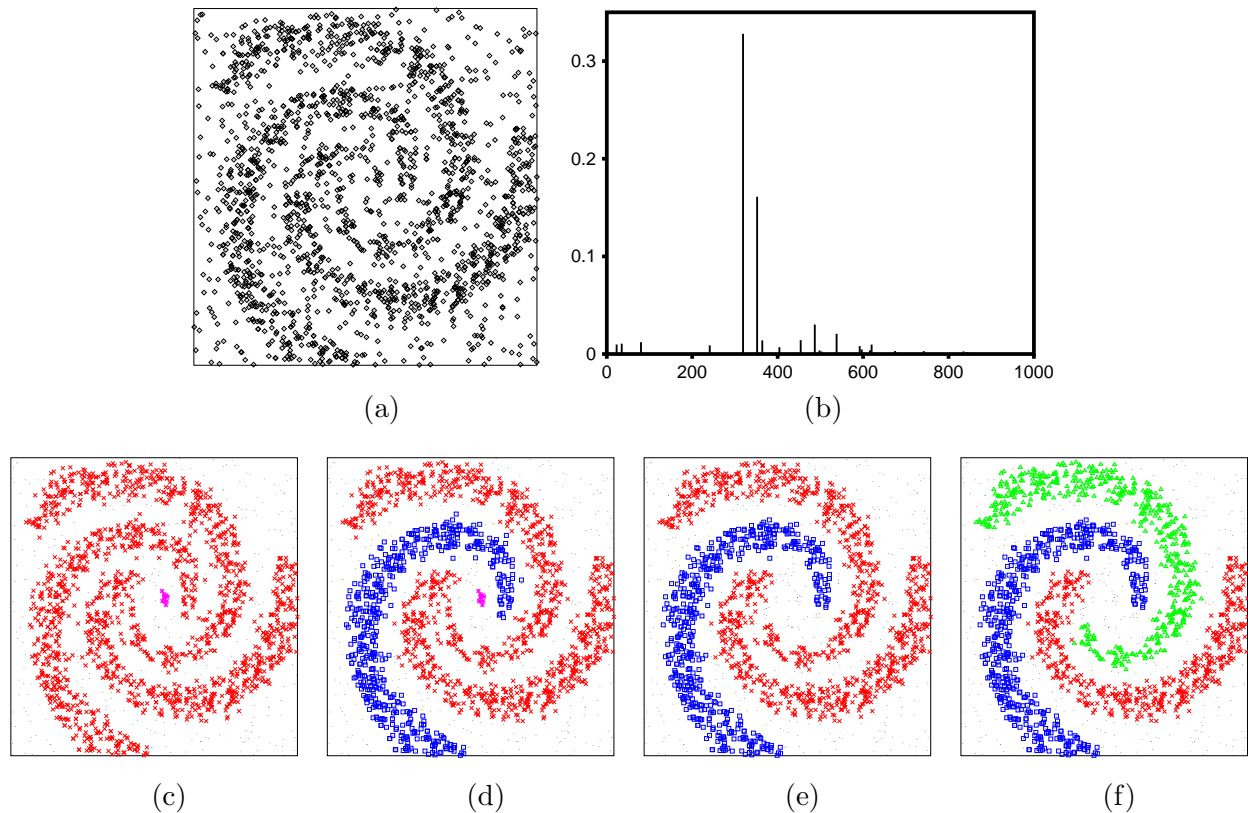


Figure 2.4: Clustering of points in the Euclidean plane. (a) The 2000 data points; the coordinates of the points are not available to the clustering algorithm, which uses only the matrix of pairwise distances. (b) The graph of $\Delta T(r)$, computed for every integer r between 1 and N . Two peaks are 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.

$\exp(-d_{ij}^2/a^2)$, 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 2.5). 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_{ij}^r . This is done using a sample of M partitions at each r level (see procedure **STAGE-1** in Figure 2.1). We will claim in Section 2.4 that for an accuracy level ϵ the asymptotic dependence of the sample size (M) on the number of points (N) is $M = O(\log N/\epsilon^2)$. Here we have used $M=1000$, although a much smaller could suffice as well (see Figure 2.7 which uses the same data with only 100 iterations). The output of this stage is the three dimensional array p_{ij}^r .

In the second stage of our algorithm we call the procedure **STAGE-2** (Figure 2.3), 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 2.4b 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 2.4c). 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 2.4d). Increasing r further reveals some boundary effects, where 41 additional points become isolated, but

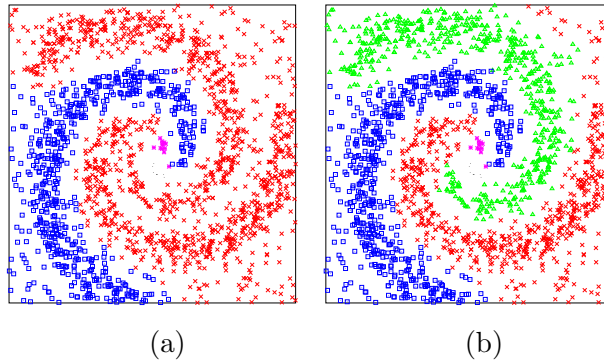


Figure 2.5: Using the relabeling option of procedure **STAGE-2**, which is discussed in more details in Section 2.7.3. (a) The result at $r=319$, after the first transition, (b) The result at $r=352$, after the second transition. The idea behind the relabeling algorithm is to weaken the dependence on the specific 0.5-threshold used in the definition of the typical cut. It is therefore suitable for boundary points, which are left unlabeled due to the competition between two or more different clusters. It is not suitable for “unstructured background” as exists here, where it is better to leave the isolated points unlabeled.

the overall picture remains unchanged as long as $r < 352$ (Figure 2.4e). The peak at $r=352$ signals the next significant change, giving the partition shown in Figure 2.4f.

The meaningful partitions are reported by the procedure **STAGE-2** when its parameter δ is set to 0.1 (Figure 2.4b shows that our method is not very sensitive to the exact value). Figure 2.5 shows the results of applying the relabeling option (line $(*)$ in the pseudo-code describing **STAGE-2** procedure). This option is not suitable here, since when “unstructured background” exists, it is usually preferable to leave isolated clusters unlabeled. We show these results for pedagogical reasons only.

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 2.6(a,b); clearly these methods fail here, while our algorithm produces good results.

Our algorithm generates a few good partitions among the N hierarchically generated partitions, and automatically chooses the best ones. To illustrate that generating good partitions is not trivial, we also tested the deterministic single linkage algorithm, which is a deterministic version of our algorithm. This algorithm gives a single hierarchy of N partitions, while in our method we average over M stochastically generated hierarchies. If we try to select good solutions among the N partitions that are generated by the deterministic single linkage, we find that this hierarchy does not contain the desired solutions. No matter what stopping criterion is used, whether it is a large change of $T(r)$ or the selected weight or anything else, the single linkage algorithm cannot return

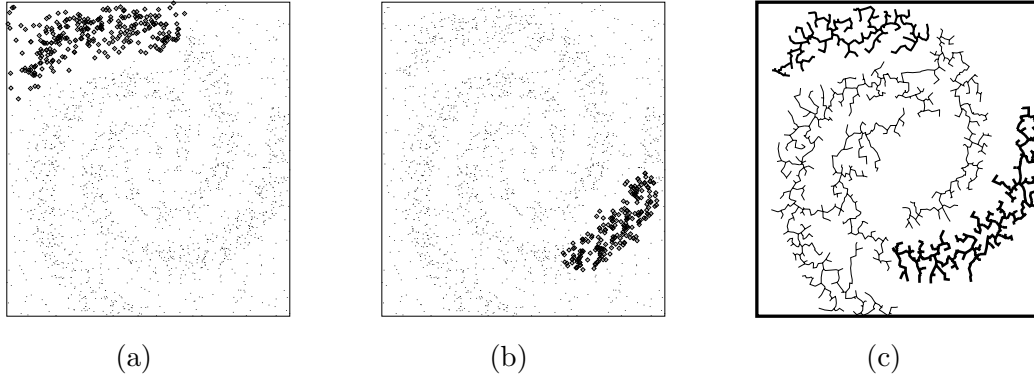


Figure 2.6: The results of other algorithms applied to the data of Figure 2.4. The same preprocessing and exponential transformation from distances to weights was used (see text). (a) The best normalized cut [127] partition. (b) The result obtained by the factorization method [109]. (c) Deterministic 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.

the desired solutions (see Figure 2.6c).

The spiral arrangement of points and the added noise in this example 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, these partitions are not found by recursive divisions into two parts, the method employs by most other hierarchical methods, hence we have a natural possibility to leave problematic points unclassified.

2.4 Complexity

The detailed complexity analysis is postponed to Section 2.7, where we describe our efficient implementation. The efficient version drastically decreases the number of estimated variables (p_{ij}^r) from N^3 to $|E|$, which is $O(N)$ for sparse graphs and N^2 for complete graphs. Moreover, it will be shown there that one graph contraction (one iteration of the external loop in **STAGE-1**, Figure 2.1) can be implemented in $O(N \log N)$ time for sparse graphs.

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 $O(N \log^2 N)$ for sparse graphs. The efficient implementation of **STAGE-2** takes only $O(N \log N)$ time in this case (Section 2.7.3), making $O(N \log^2 N)$ the overall sparse graph 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_{ij}^r 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_{ij}^r by no more than ϵ , with probability $1-\delta$.

Since the lower bound which we will find for M will not depend on r , we pretend that r is fixed to some arbitrary r_0 , which is omitted from the notations. Hence the notation p_{ij} stands for $p_{ij}^{r_0}$ 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 number 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 [74] implies that for $0 \leq \epsilon \leq 1$:

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

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 :

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

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 2.7.1) introduces an additional error, since it involves an approximative median computation. However, this affects the parameter ϵ but not the logarithmic dependence on the number of nodes.

2.5 Robustness

Three aspects of robustness are discussed below. Section 2.5.1 considers the external parameters of the algorithm, and discusses the sensitivity of the algorithm to their values. Sections 2.5.2 demonstrates robustness with respect to data perturbation. Section 2.5.3 shows that our method, which finds an average solution, is robust with respect to the clustering hypothesis.

2.5.1 External parameters

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. We consider them in more details below.

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. The second part, **STAGE-2**, involves the parameter δ that determines whether a change in the function $T(r)$ is significant. Our examples throughout this work 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 2.4, 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.

We now investigate the preprocessing stage, which defines the weights w_{ij} of the graph edges. Given a dissimilarity measure d_{ij} between stimuli i and j , we define $w_{ij} = \exp(-d_{ij}^2/a^2)$. Here a is

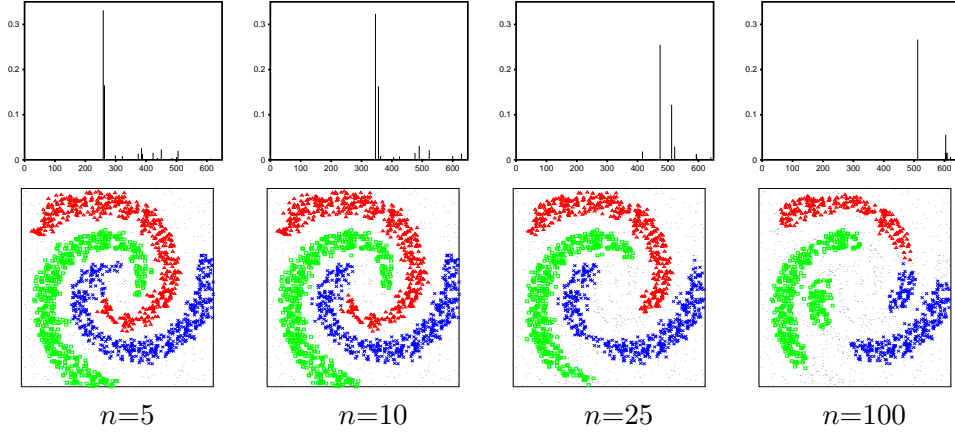


Figure 2.7: Robustness with respect to the local scale parameter. The local scale (a) is defined as the average distance to the n -th nearest neighbor, and it determines the decay rate of the weights w_{ij} . We repeat the experiment with $n=5, 10, 25$ and 100 . Partitions that corresponds to the second peak are shown for each case. The sample size (the parameter M of STAGE-1) was set to 100 .

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:

$$w_{ij} = \prod_{\mu} \exp(-d(\mu)_{ij}^2 / a(\mu)^2) \quad (2.2)$$

Figure 2.7 extends the illustrative example of Section 2.3. We repeat the same experiment with different local scale values, choosing a to be the average distance to the n -th nearest neighbor, setting $n=5, 10, 25, 100$. The larger n is, the slower is the decay rate of the weights w_{ij} . Only the 50,000 edges with largest weights have been kept, and the rest were discarded¹. Note that the ranking of the edges according to their weights does not depend on n , hence we keep the same 50,000 edges in each case. Thus the graph topology is kept unchanged while the weight decay rate is varied. We conclude that the parameter n in this example can be varied by almost an order of magnitude before the local structure is lost.

The elimination of small weight edges is used to increase efficiency. We investigate next the sensitivity to the elimination process, using a fixed local scale ($n=10$) and a variable threshold

¹The analysis of the complete graph with $n=10$ produces the same results, as we have reported in [36]. The analysis of a complete graph with uniform weights follows in Section 2.5.3.

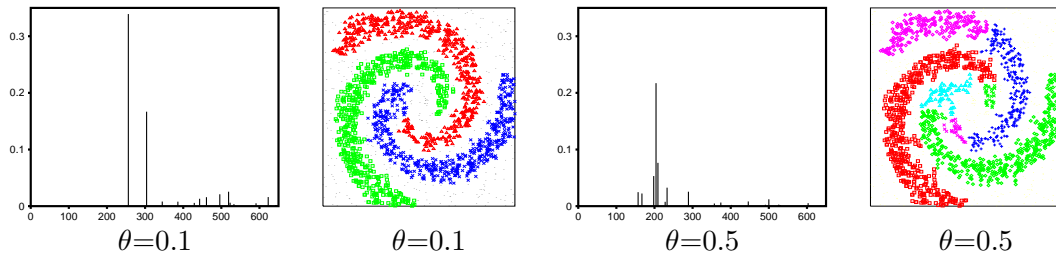


Figure 2.8: Levels of thresholding. An edge whose weight w_{ij} is smaller than a parameter θ is discarded from the graph, in order to increase computational efficiency. We usually use $\theta \simeq 0.01$, which is one percent of the weight range. Here we show the results obtained after more drastic edge elimination, using $\theta = 0.1$ and 0.5 . Under these conditions the number of remaining edges is 25331 and 8589 (out of 1999000, the original number of edges in the complete graph).

θ , where edges with $w_{ij} < \theta$ are discarded. In the experiment shown above (Figure 2.7, second column), the most significant 50000 edges are kept, and this corresponds with threshold $\theta = 0.00625$. Our earlier experiments show that this result is not sensitive to the exact value of θ , as long as it remains small: see [36] for our results on the same data using the complete graph ($\theta = 0$), and the results reported in Section 2.3 above (using $\theta = 0.01$).

For completeness we also experiment with relatively high values of $\theta = 0.1$ and 0.5 , which are not reasonable for real applications (only 25331 and 8589 edges are left, respectively). While the results remain good even for $\theta = 0.1$ (Figure 2.8), for $\theta = 0.5$ too much information is lost, and the algorithm fails to find global structure.

We note that weight thresholding is not the only possible way to reduce the number of edges and obtain a sparse graph. In some applications, like image segmentation, the definition of neighborhood is natural and can be used; only nodes within such a neighborhood can have a viable edge to the reference node. Spatial neighborhoods are combined with random selection of viable neighbors in [127]. Mutual neighborhoods [10, 151] are another alternative, where an edge (i, j) is kept only if node j is one of the n -th nearest neighbors of node i and vice versa, for some chosen parameter n .

2.5.2 Data perturbation

Consider two statistical sources of points in the Euclidean plane. Each source generates points at a distance r from the origin, where r is a random variable with mean $r^{(1)}$ and $r^{(2)}$ for the first and the second sources, respectively. The combined set of generated points thus appears as two concentric circles in the plane, and the task of a clustering algorithm is to partition a given set of points into the two distinct circles.

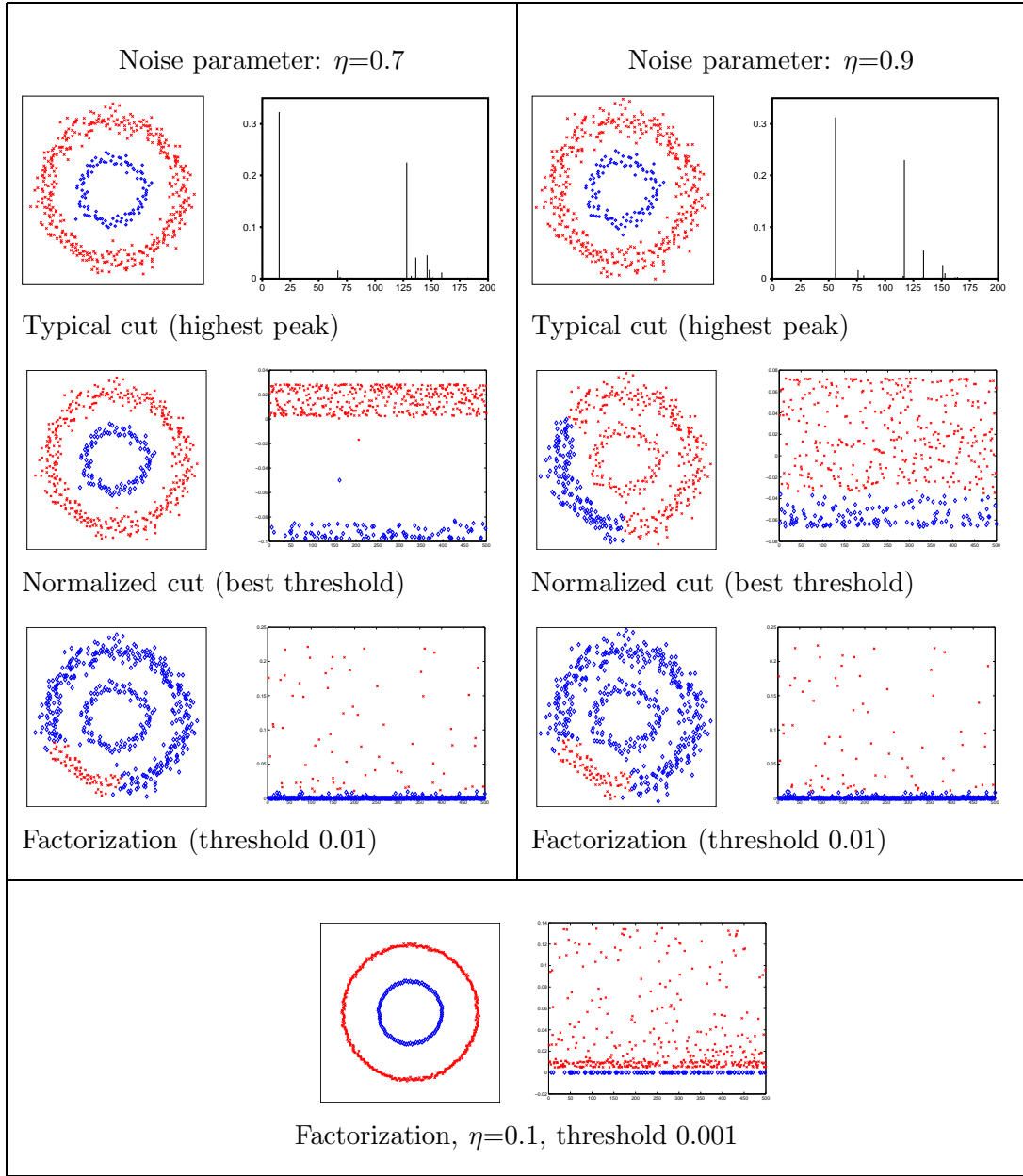


Figure 2.9: Robustness under data perturbation. The control parameter η defines the spread of the points, or the “width” of each circle. Results for $\eta=0.7$ and 0.9 are shown in the upper part. The factorization algorithm fails for both, the normalized cut algorithm fails for the the larger noise 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. See text for further discussion. The bottom part of the figure shows that the factorization algorithm succeeds to find the structure where the noise level is very low ($\eta=0.1$), although even then the possibility to automatically find the correct threshold is questionable; note that this method already fails for $\eta=0.2$ (not shown).

Without prior knowledge on the nature of the sources it is not known that polar representation is advantageous, and algorithms which are based on moments estimation, like the k-means algorithm, are likely to fail. Spectral graph methods may find the desired partition, since they exploit pairwise information. In this section we compare the robustness of our algorithm with two spectral methods, and show that our algorithm is more robust under data perturbation.

In our experiment we define the following two sources. Let $x \sim \mathcal{N}(0, 1)$ be a random normal variable with zero mean and unit variance, and let $\eta \in \mathcal{R}$ be our control parameter. Each source generates points which are evenly spaced in their θ coordinate, while the radius coordinate is a random variable generated by $1.5 + 0.2\eta x$ for the first source, and $0.7 + 0.1\eta x$ for the second. We generate a set of 500 points, 400 from the first source and 100 from the second. The pairwise Euclidean distances d_{ij} are computed, and transformed into similarity weights by our usual transformation, namely, $w_{ij} = \exp(-d_{ij}^2/a^2)$, where a is the average distance to the 10-th nearest neighbor. The 15,000 edges having maximal weights are kept, while the rest are discarded (this corresponds with a weight threshold of about 10^{-5}). The obtained weighted graph is fed into our clustering algorithm, as well as to two spectral algorithms (the normalized cut algorithm [127] and the factorization algorithm [109]).

The results are shown in Figure 2.9, for two noise levels ($\eta=0.7$ and 0.9). In the first row we show our results; the impulse graphs of $\Delta T(r)$ and the partitions which correspond with the highest peaks. In the second row we show the results of the normalized cut algorithm, and in the third row the results of the factorization algorithm. These spectral methods employ the computation of the second and first eigenvector of the similarity matrix (or some normalized version of it) respectively, and they split the data by thresholding the entries of this vector. The method is useful if a salient threshold exist, as in Figure 2.9 - second row on the left². We find that there is no salient threshold for the vector computed by the factorization method, even when the noise level is very low (bottom of Figure 2.9). The vector computed by the normalized cut algorithm contains values which can be easily grouped when the noise level is $\eta=0.7$, but for slightly larger noise the values form a continuum.

²There are at most $N - 1$ different choices for the threshold value, as the N entries of the eigenvector are real numbers, and the same partition into two sets is obtained when the threshold is varied in the interval between subsequent values. For the normalized cut algorithm, where a cost function is defined, we exhaustively check each one of the different partitions, and select the one having lowest cost. With the factorization algorithm this cannot be done, since no cost function is defined, and we select the threshold by inspection.

2.5.3 False structure

A clustering algorithm can be viewed as a search algorithm in some hypothesis space, where a hypothesis is a feasible partition. The algorithm may define a scalar functional over this space, seeking its optimum, or it may search heuristically like the agglomerative methods. In both of these cases the search ends with a single solution - some point in the hypothesis space. The problem with a single point solution, even when it obtains the global optimum of the problem, is that it does not take into account the notion of support. By the support of a specific partition we vaguely refer to the total quality weight of other partitions which are “similar” to the chosen one.

Some notion of support is captured by soft clustering algorithms, usually inspired by models from statistical mechanics. Instead of a single point solution, these algorithms look for a distribution over the whole hypothesis space, and under this distribution average properties are computed (like the probability that a certain point belongs to a certain cluster, or the probability that two points are in the same cluster). The sharpness of the distribution is controlled by an external parameter, called temperature.

Our algorithm follows the same lines. We induce a probability distribution over all r -way cuts, and compute pairing probabilities p_{ij}^r under the induced distribution. To show that we gain robustness, we use an example of unstructured data. This is the extreme case where single point (zero temperature) algorithms tend to impose structure on the data rather than detect it (or its absence). They lack the information that the partition with best quality is not supported by similar high quality partitions. On the contrary, other partitions which degrade only slightly in quality, may divide the data in a very different way.

Our example consists of a complete graph over 100 vertices. Every edge in the graph is assigned the weight $1 + x$, where x is uniformly distributed in $[-0.1, 0.1]$. Hence all points are about equally similar to each other, and no structure is to be found. Indeed, Figure 2.10 shows that our algorithm finds the two possible interpretations: either all the points are at one cluster, or they are all isolated.

For comparison, we apply deterministic agglomeration to the same data, Figure 2.11. For every number of clusters between 1 and N a deterministic agglomeration suggests a single solution. To conclude that there is no real structure in the data, the resulting dendrogram must be validated using a statistical hypothesis testing procedure [65], which is known to be a difficult task. Figure 2.11 shows that the power of our algorithm does not lie in the heuristic $\Delta T(r)$ criterion, but

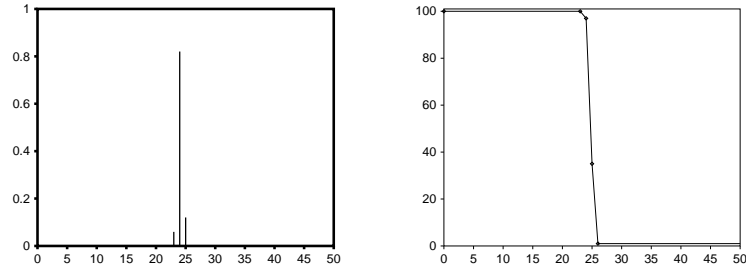


Figure 2.10: No structure example. A complete graph over 100 nodes is considered, with weights that are uniformly distributed between 0.9 and 1.1. Left: the graph of $\Delta T(r)$ shows a transition from a single cluster at low r values to N clusters at high r values. The “transition width” is $\Delta r = 3$. Right: the size of the largest cluster found (the number of nodes in the largest component of the typical cut) as a function of r . The graph shows a sharp decrease from the value 100 (all points in one cluster) to the value 1 (all points isolated).

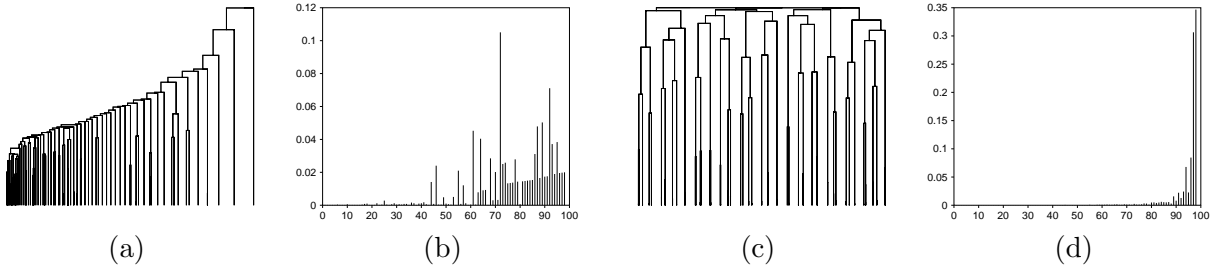


Figure 2.11: Deterministic agglomeration applied to the same data used in Figure 2.10. By inspection, the dendrograms do not suggest a meaningful partition of the data, but a general validation criterion is hard to find. (a) Single linkage result, (b) Our $\Delta T(r)$ heuristic applied on the single linkage hierarchy of partitions, (c) Complete linkage result, and (d) Our $\Delta T(r)$ heuristic applied on the complete linkage hierarchy. The spacing between endpoints in the dendrograms are proportional to their level of merging, hence some lines cannot be noticed and appear as a single thicker line due to the limited graph resolution.

in the averaging over the hypothesis space. The heuristic criterion is not useful when applied with deterministic agglomeration, even in this extreme case example.

2.6 Cross Validation

Every meaningful level of clustering is identified by a large change in the function $T(r)$ defined in Equation (2.1), or equivalently by a pronounced peak in the graph of $\Delta T(r)$. Here we are concerned with the inverse direction, whether a particular peak of $\Delta T(r)$ signals an interesting interpretation of the data. In theory we would expect it to be the case, since false signals are expected to be canceled out by averaging, as is demonstrated by the clique example above. However, if the data is too sparse, a misleading interpretation can capture a sampling artifact rather than a true structure.

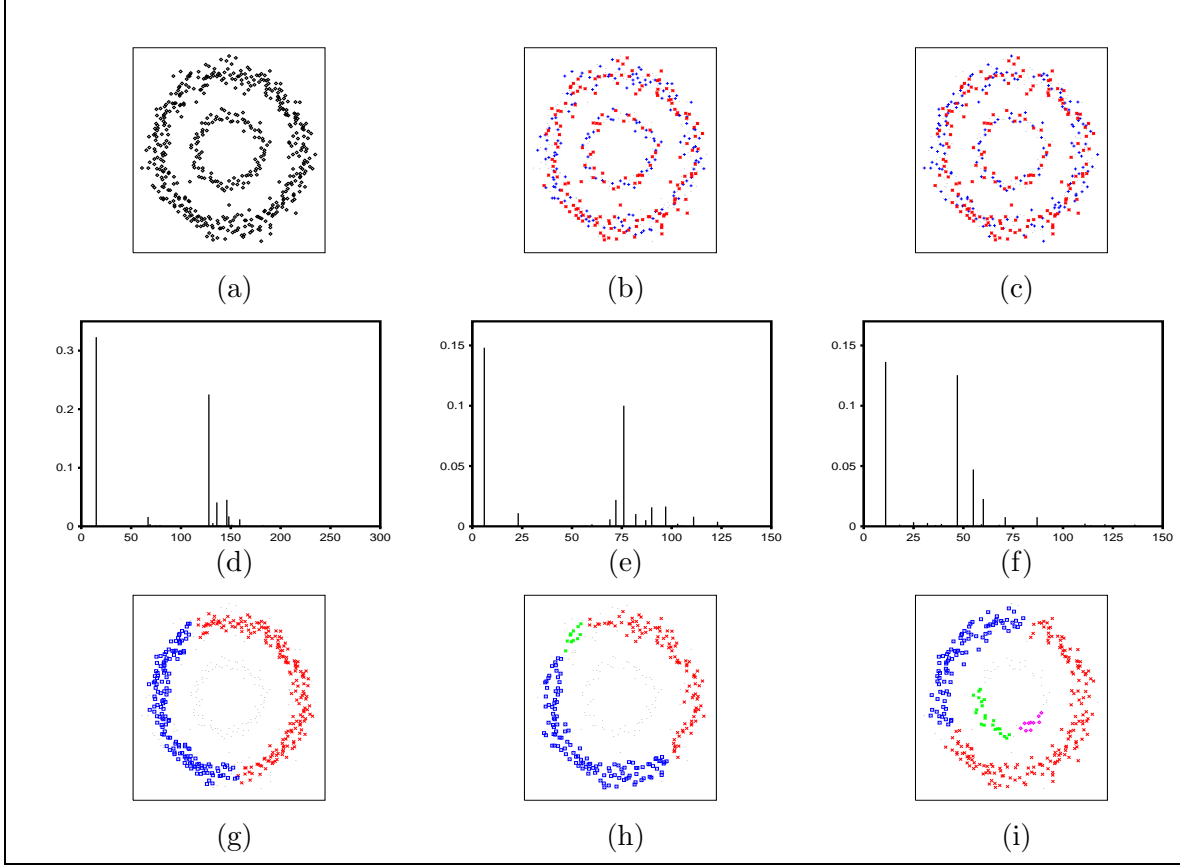


Figure 2.12: Cross validation example. (a) The complete data, 500 points. (b) The subsample S_1 : the red crosses are points which are in $S_1 \cap S_2$. Black dots mark points which are in the complete data but not in S_1 . (c) The same as b, for the second subsample, S_2 . (d-f) The graphs of $\Delta T(r)$ obtained for each case. The cross validation test aims to distinguish between peaks that reflect true structure to peaks which are sample dependent. (g-i) The partitions which correspond with the second highest peak. See Figure 2.9 for the first partition of the complete data.

An example is given in Figure 2.12(a,d,g), where we show the partition that corresponds to the second highest peak in the concentric circles example used before in Section 2.5.2. Evidently, the outer cluster is broken into two parts due to accidental gaps.

Although the partition shown in Figure 2.12(g) is meaningful with respect to the given sample of points, it is not robust with respect to sampling. Thus for another sample of points drawn from the same statistical sources, it is not likely that a “similar” partition would emerge. Similarity between partitions can be measured with respect to common pairs of points, which are contained in both samples. For example, two points which are clustered together according to the first partition are expected to be clustered together according to the second.

To formulate our cross validation scheme, which is inspired by [88, 25], let S be a sample of N points and divide S into two random overlapping parts, S_1 and S_2 , such that $|S_1| \simeq |S_2| \simeq 2N/3$ and $|S_1 \cap S_2| \simeq N/3$. We apply the clustering algorithm independently to the subsets S_1 and S_2 , and we compare the pairing of points which belong to the intersection $S_1 \cap S_2$. Our goal is to validate the peaks of $\Delta T(r)$ obtained independently for the two subsamples. To this end, we start with the two partitions that correspond to the highest peaks in both subsamples, and we test their agreement with respect to common pairs of points. We continue to the next highest peaks, until the agreement test fails³.

Recall that the function $T(r)$ counts the number of edges which cross the typical cut⁴. Our cross validation scheme check whether *the same* edges are counted in both cases. Let $x_{ij}^{(r1)}$ be a binary variable, indicating whether the edge (i, j) contributes to $\Delta T(r1)$ when it is computed for the subsample S_1 . Similarly, let $y_{ij}^{(r2)}$ indicate the same property with respect to S_2 . In addition, let $z_{ij}^{(r1),(r2)}$ be 1 if the edge is counted in both case (in the relevant r value each time), and 0 otherwise. We define X , Y and Z to be the sums of x_{ij} , y_{ij} and z_{ij} respectively, over all edges (i, j) (of the complete graph) for which $i, j \in S_1 \cap S_2$. Our cross validation statistic is:

$$\rho = \min \left(\frac{Z}{X}, \frac{Z}{Y} \right).$$

The difference between our scheme and classical validation methods is that we do not test the null hypothesis of “no structure”. Instead, we test the hypothesis that two given partitions are identical. The hypothesis is rejected if unusually small values of the statistic ρ are obtained.

³Note that a more careful peak correspondence scheme is required if there exist peaks with similar heights, which may be interchanged.

⁴More accurately, $T(r)$ counts only the crossing edges which incident on components of sufficient size.

Unfortunately, as in the classical case, the distribution of the statistic under the null hypothesis is unknown. Quantitative analysis calls for Monte Carlo simulations to estimate its distribution under some noise model, but this is beyond our scope. Instead, we reject a pair of partitions if $\rho < 0.9\rho_1$, where ρ_1 is the value obtained for the first (highest) pair of peaks.

Figure 2.12 shows the concentric circles example of Section 2.5.3 (with noise parameter $\eta=0.7$). We evaluated the cross validation indices for the 5 largest peaks, and repeated the evaluation 15 times with different random subsampling of the data. One of the 15 trials is shown in the figure. The average values found for $\rho_1 \dots \rho_5$ are the following (RMS in parenthesis): 0.99(0.02), 0.43(0.12), 0.10(0.12), 0.08(0.14), 0.01(0.03). According to our rule of thumb, we accept the first partition and reject the others. See Section 4.3 for the application of our cross validation scheme to image segmentation.

2.7 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_{ij}^r 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_{ij}^r 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_{ij}^r in this case is determined solely by transitive relations, we can count 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_{ij}^r is never used; we only need to know whether it is smaller or larger than $\frac{1}{2}$. This observation is used in Section 2.7.1 to further reduce the number of computed variables down to $|E|$. Second, the contraction of a sparse graph can be implemented in $O(|E| \log |E|)$ time. This is shown in Section 2.7.2. Finally, Section 2.7.3 shows how the typical cuts at successive values of r can be computed incrementally.

2.7.1 Computing the level crossing of probability 0.5

We notice that p_{ij}^r is monotonic in r by construction, with $p_{ij}^N=0$ and $p_{ij}^1=1$, since 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_{ij}^r is smaller or larger than 0.5, it is sufficient to know for every i - j pair which r satisfies $p_{ij}^r = 0.5$. We denote it r_{ij} . We then know that $p_{ij}^r < 0.5$ if and only if $r > r_{ij}$.

The immediate consequence is that the number of variables to compute is reduced from $N|E|$ to $|E|$. Another consequence, which will be discussed in Section 2.7.3, 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 we 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 estimation, 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 [47]. 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 2.13. 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**.


```

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  $a_1 \dots a_K$  is kept ordered.

```

Figure 2.13: Updating an existing histogram with a new value.

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.

2.7.2 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 ($r=N\dots1$) 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 2.14, 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.

```

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  $m = 1 \dots M$ :
  (04)   INITIALIZE( $T$ )
  (05)    $ne \leftarrow |E|$ 
  (06)    $r \leftarrow N$ 
  (07)   while  $ne > 0$  do:   (the contraction loop)
  (08)      $leaf \leftarrow \text{SELECT-EDGE}(T)$ 
  (09)      $u, v \leftarrow \text{META-NODES-NAMES}(leaf)$ 
  (10)      $L \leftarrow \text{CONNECTING-EDGES}(u, v)$ 
  (11)      $ne \leftarrow ne - |L|$ 
  (12)      $r \leftarrow r - 1$ 
  (13)     for each  $leaf \in L$ :
  (14)        $T \leftarrow \text{DISCARD-EDGE}(leaf, T)$ 
  (15)        $H(leaf) \leftarrow \text{INSERT}(r, H(leaf))$ 
  (16)     end-loop
  (17)   end-loop
  (18) end-loop
  (19) for  $leaf = 1 \dots |E|$ :
  (20)   use lookup table to convert  $leaf$  into an  $i$ - $j$  pair
  (21)    $r_{ij} \leftarrow \text{MEDIAN}(H(leaf))$ 
  (22) end-loop
  (23) return  $R = \{(i, j, r_{ij})\}_{(i, j) \in E}$ 

```

Figure 2.14: Efficient implementation of the transformation from similarity weights to pairing probabilities.

Line 1:

The estimation of the 0.5-probability level uses a histogram data structure, as discussed in Section 2.7.1. 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.

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 $u=\text{FIND}(i)$ and $v=\text{FIND}(j)$.

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 become 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 2.13.

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 2.7.1, 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. An alternative implementation (which we have realized and used) is described in the appendix. It seems to be better than the implementation described here, but its full complexity analysis is currently beyond our reach.

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, $\text{FIND}(i)$ returns the value of $\text{names}(i)$, and is clearly an $O(1)$ operation. On the other hand, assuming w.l.o.g that $\text{sizes}(u) \geq \text{sizes}(v)$, the operation $\text{UNION}(u, v)$ takes $O(\text{sizes}(v))$ time. Instead of describing it separately, we incorporate the union operation into the procedure CONNECTING-EDGES and analyze them together. Figure 2.15 shows the implementation.

```

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

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

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

(15)  $\text{sizes}(u) \leftarrow \text{sizes}(u) + \text{sizes}(v)$ 
(16)  $\text{sizes}(v) \leftarrow 0$ 
(17)  $\text{members}(u) \leftarrow \text{CONCATENATE}(\text{members}(u), \text{members}(v))$ 
(18)  $\text{members}(v) \leftarrow \text{NULL}$ 
(19) return  $L$ 

```

Figure 2.15: 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 $\text{sizes}(v) \leq \text{sizes}(u)$, 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 2.14) is $O(N \log N)$.

To complete the complexity analysis we refer again to the pseudo-code in Figure 2.14. 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 $|E| = O(N)$ for sparse graphs, we conclude that a single graph contraction can be implemented in $O(N \log N)$

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 $O(N \log^2 N)$ time and $O(N)$ space.

2.7.3 The second stage

In Section 2.7.1 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 .

```

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  $r = N \dots 1$ :
(05)     $\Delta T(r) \leftarrow 0$ 
(06)    while  $r_{ij} = r$  do:
(07)       $u \leftarrow \text{FIND}(i)$ 
(08)       $v \leftarrow \text{FIND}(j)$ 
(09)       $\Delta T(r) \leftarrow \Delta T(r) + 2 \cdot sizes(u) \cdot sizes(v) / N(N - 1)$ 
(10)       $\text{UNION}(u, v)$ 
(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 2.16: Efficient implementation of the second stage.

Figure 2.16 summarizes our efficient **STAGE-2** algorithm. The list R is sorted at line 1 using

quicksort, which takes $O(|E| \log |E|)$ time on average. The data structures *names*, *members* and *sizes* at 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 at 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 (2.1). The complexity of the loop is $O(N \log N)$, as discussed in Section 2.7.2. We note that the alternative implementation of UNION and FIND discussed in the appendix can be used here as well.

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 $\text{UNION}(\text{FIND}(i), \text{FIND}(j))$ 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.

2.8 Probability bounds and relation to other work

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 [72]. We quote here some results concerning the contraction algorithm, and the reader is referred to [72] 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.

A related yet “opposite” situation occurs in network reliability problems, where every edge has probability p to fail, and the probability $P(p)$ of a network disconnection is sought. Thus it is needed to average over all possible cuts, and the most obvious way to estimate $P(p)$ is through Monte-Carlo simulations. Interestingly enough, it was found [73] that $O((\log N)/\epsilon^2 P(p))$ trials suffice to estimate $P(p)$ to within $1 \pm \epsilon$ 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).

We now wish to put the contraction algorithm in the context of agglomerative clustering. 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 [26, 65, 85].

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. On the other hand, the stochastic contraction is designed to generate with high probability low capacity cuts, and cut capacity is a global measure for clustering quality. To complete the picture we note that a deterministic contraction scheme for minimum cut computation is proposed in [99].

Using the cut capacity as a global quality measure for clustering is a well known approach (e.g., [69, 152]). One of its problems is that the minimal cut tends to break small parts from the graph. Consequently [127] 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 [40, 57, 113] and in particular to the granular magnet model of [10, 151], also called the super paramagnetic clustering (SPC) algorithm. In the rest of this section we discuss the relationship between our method and SPC. Terms of statistical mechanics are briefly explained, for readers who are not familiar with the subject.

A basic concept in models of statistical mechanics is that of a configuration space. A configuration \mathcal{S} is a microscopic description of the state of the system. For example, it might specify the location and velocity of every atom. Two scalar functions are defined over the configuration space: an energy function $\mathcal{E}(\mathcal{S})$, and a probability distribution $\mathcal{P}(\mathcal{S})$. To fix the system temperature to a value \mathcal{T} means to set the energy expectation $\langle \mathcal{E}(\mathcal{S}) \rangle$ to some fixed value. Many probability distributions can satisfy this constraint, but the one which appears in nature is Gibbs distribution: $\mathcal{P}(\mathcal{S}) \sim \exp(-\mathcal{E}(\mathcal{S})/\mathcal{T})$. Among all possible distributions which fix $\langle \mathcal{E}(\mathcal{S}) \rangle$ to the same value, Gibbs distribution has the maximal entropy. Entropy is a measure for the flatness of a distribution, with the uniform distribution having maximal entropy. Hence Gibbs distribution satisfies the temperature constraint while being least committed to a specific configuration [66].

The granular magnet model maps the clustering problem to a problem in statistical mechanics as follows. Every labeling (or coloring) of nodes is a configuration \mathcal{S} . There are q labels available, and each node can be labeled independently by each one of them, hence there are q^N possible configurations. It is convenient for our analogy to identify each configuration with a cut, where

connected nodes having the same label are at one side of the cut⁵. In this terminology the energy $\mathcal{E}(\mathcal{S})$ of a configuration, as defined by the granular magnet model, is precisely the capacity of the corresponding cut. According to the Gibbs distribution, low capacity cuts are therefore more probable.

The SPC algorithm proceeds by estimating, at a fixed temperature \mathcal{T} , the probability $p_{ij}^{\mathcal{T}}$ that nodes i and j have the same label. Since the model was originally described using “spin states” instead of node labels, the probability $p_{ij}^{\mathcal{T}}$ was referred as the “spin-spin correlation function”. The analogy with our p_{ij}^r values is evident, although not complete (see below). To find the clusters at a temperature \mathcal{T} , the graph is disconnected by eliminating the edges having $p_{ij}^{\mathcal{T}} < 0.5$. We have adopted the same heuristic in our method.

In the magnetic model the control parameter is the temperature \mathcal{T} , which sharpen or flatten the probability distribution and controls the expected capacity. For each temperature value, the probabilities $p_{ij}^{\mathcal{T}}$ are computed over all r -way cuts. In our method, the control parameter is r , and the probabilities p_{ij}^r are computed over cuts which have exactly r sides. Both choices are somewhat arbitrary, although using Gibbs distribution is appealing from the point of view of maximal entropy inference. This, however, comes with an additional computational cost, to be examined below. Note also that while r is a discrete parameter \mathcal{T} is a continuous one, and finding the temperatures of interest is a search over a continuous domain.

Both methods involve stochastic sampling of cuts in order to estimate the pairing probabilities, either p_{ij}^r or $p_{ij}^{\mathcal{T}}$. The sampling tool in our method is the contraction algorithm, and in SPC it is the method known as Swendsen-Wang (SW) algorithm [144]. There are few important differences between the two. First, the SW algorithm generates cuts at a fixed temperature. Hence the sampling process, which is the computation bottleneck, needs to be repeated for every temperature value. On the other hand, the contraction algorithm is used to estimates p_{ij}^r for every $1 \leq r \leq N$ at once. Second, the required sample size is much smaller in our method. In fact, it is not known what sample size guarantees a certain accuracy level in the SW framework, although experimentally it is known to be polynomial in N . In our framework $O(\log N)$ samples suffice to guarantee specific degree of accuracy. The reason for the larger sample size required by SW might be that the SW sampler is Markovian, and subsequent samples which are generated are not independent. The power of SW is that these dependencies are weak, and this is probably the reason why after generating

⁵Note that this mapping is many to one, since a cut side can take one of q labels.

a polynomial number of samples the process succeeds in converging to the stationary distribution over the configuration space (a property called “rapid mixing”). In comparison, cuts which are generated by the contraction algorithm at a fixed r level are completely independent, a fact which was taken into account in bounding the sample size (Section 2.4), and which seems to speed up the computation. In addition, the independence between samples in our algorithm allows for a trivial parallel implementation.

One of the important aspects of our work is the link it provides between clustering approaches which seemed to be very different. We have presented above the SPC algorithm in terms of graph cuts, but originally it was proposed on the basis of an analogy to a system of microscopic magnets (spins). In spite of being very attractive in terms of performances, the relevance of the physical analog remained questionable. The relations between SPC, the min-cut algorithm and single linkage are revealed by our work.

Chapter 3

Dissimilarity between Shapes

One of our main goals in this work is to organize a set of images in shape categories. Such an organization is useful for image retrieval, due to several reasons: queries are better addressed on the basis of global organization than on the basis of pairwise (nearest neighbor) information; the processing may be limited to a fraction of a large database relying on prior organization; and semi automatic retrieval may use cluster representatives to help the user navigating in the image collection.

We focus on images of objects, and on organization by shape. We ignore other dimensions of similarity (e.g., color, motion, context). In this chapter we propose a dissimilarity measure for shapes, that are the boundaries of objects in the scene. The method relies on a novel curve matching algorithm, which establishes correspondence between key feature points extracted on the boundaries. Our algorithm is flexible, designed to match curves under substantial deformations and arbitrary large scaling and rigid transformations. A syntactic representation is constructed for both curves, and an edit transformation which maps one curve to the other is found using dynamic programming.

We start in Section 3.1 with background on the curve matching problem in vision. Our matching algorithm is outlined in Section 3.2, where we also present extensive experiments, examining partial occlusion, viewpoint variation, articulation, and class matching (where silhouettes of similar objects are matched). We explain how the matching of feature points supports a definition of shape dissimilarity, and we compare our results to other. The details of the algorithm follow in Section 3.3. A comparison between our syntactic operations and similar operations from the literature can be found in Appendix B.

3.1 Curve matching: problem and related work

Contour matching is an important problem in computer vision with a variety of applications, including model based recognition, depth from stereo and tracking. In these applications the two matched curves are usually very similar. For example, a typical application of curve matching to model based recognition would be to decide whether a model curve and an image curve are the same, up to some scaling or $2D$ rigid transformation and some permitted level of noise.

In this work we are primarily interested in the case where the similarity between the two curves is weak. The organization of silhouettes into shape categories (like tools, cars, etc.) necessitates *flexible* matching, which can support graded similarity estimation.

While our approach focuses on the silhouette boundary, a dual approach is based on its medial axis. Specifically, a medial axis together with singularities labeling form a shock graph representation, and matching shock graphs is an isomorphism problem. The methods for solving it includes semi-definite programming [129], replicator dynamics [105], graduated assignment [122] and syntactic graph matching [134]. In some of these cases the matching is only structural, while in others two levels of matching (structural and metrical) are supported. The methods based on shock graphs succeed to define a graded similarity measure, and may be combined with suitable database indexing [128]. In this work we show, however, that our results are of the same quality in spite of using boundary representation, which is inherently less sensitive to occlusion, and which does not involve the NP-complete graph isomorphism problem.

To put our method in the context of existing work on boundary matching, we first distinguish between dense matching and feature matching. Dense matching is usually formulated as a parameterization problem, with some cost function to be minimized. The cost might be defined as the “elastic energy” needed to transform one curve to the other [6, 14, 18], but other alternatives exist [3, 38, 22, 92]. The main drawbacks of these methods are their high computational complexity (which is reduced significantly if only key points are matched), and the fact that they are usually not invariant under both $2D$ rotation and scaling. In addition, the computation of elastic energy (which is defined in terms of curvature) is scale dependent, and requires accurate evaluation of second order derivatives.

Feature matching methods may be divided into three groups: proximity matching, spread primitive matching, and syntactic matching. The idea behind proximity matching methods is to

search for the best matching while permitting the rotation, translation and scaling (to be called alignment transformation) of each curve, such that the distances between matched key points are minimized [5, 62, 70, 140]. Consequently these methods are rather slow; moreover, if scaling is permitted an erroneous shrinking of one feature set may result, followed by the matching of the entire set with a small number of features from the other set. One may avoid these problems by excluding many-to-one matches and by using the points order, but then the method becomes syntactic (see below). Moreover, we illustrate in section 3.3.7 why proximity matching is not adequate for weakly similar curves. As an alternative to the alignment transformation, features may be mapped to an intrinsic invariant coordinate frame [90, 118, 120]; the drawback of this approach is that it is global, as the entire curve is needed to correctly compute the mapping.

Features can be used to divide the curves into shape elements, or primitives. If a single curve can be decomposed into shape primitives, the matching algorithm should be constrained to preserve their order. But in the absence of any ordering information (like in stereo matching of many small fragments of curves), the matching algorithm may be called “spread primitive matching”. In this category we find algorithms that seek isomorphism between attributed relational graphs [8, 86, 16], and algorithms that look for the largest set of mutually compatible matches. Here, compatibility means an agreement on the induced coordinate transformation, and a few techniques exist to find the largest set of mutually compatible matches (e.g., clustering in Hough space [131], geometrical hashing [84], and clique finding in an association graph [11, 20, 58, 81]). Note that at the application level, finding isomorphism between attributed relational graphs is the same problem as finding isomorphism between shock graphs (discussed above), although in the last case an additional constraint may apply [105].

For our purpose of matching complex outlines, it is advantageous to use the natural order of primitives. This results in a great simplification, and there is no need to solve the difficult graph isomorphism problem. Moreover, the relations encoded by the attributed relational graphs need to be invariant with respect to $2D$ image transformations, and as a result they are usually non local.

A syntactical representation of a curve is an *ordered* list of shape elements, having attributes like length, orientation, bending angle etc. Hence, many syntactical matching methods are inspired by efficient and well known string comparison algorithms, which use edit operations (substitution, deletion and insertion) to transform one string to the other [143, 91, 46]. The vision problem is different from the string matching problem in two major aspects, however: first, in vision invariance

to certain geometrical transformations is desired; second, a resolution degradation (or smoothing) may create a completely different list of elements in the syntactical representation.

There are no syntactic algorithms available which satisfactorily solve both of these problems. If invariant attributes are used, the first problem is immediately addressed, but then the resolution problem either remains unsolved [2, 43, 89] or may be addressed by constructing for each curve a cascade of representations at different scales [4, 95, 139]. Moreover, invariant attributes are either non-local (e.g., length that is measured in units of the total curve length), or they are non-interruptible (see discussion in section 3.3.7). Using *variant* attributes is less efficient, but provides the possibility to define a merge operator which can handle noise [111, 135, 136], and might be useful (if correctly defined) in handling resolution change. However, the methods using variant attributes could not ensure rotation and scale invariance.

3.2 Flexible syntactic curve matching: algorithm

In this section we present a local syntactic matching method which can cope with both occlusion and irrelevant changes due to image transformation, while using variant attributes. These attributes support a simple smoothing mechanism, hence we can handle true scale (resolution) changes. The algorithm is outlined in Section 3.2.1, while the missing details are given in Section 3.3. We are primarily concerned with the amount of flexibility that our method achieves, since we aim to apply it to weakly similar curves. Section 3.2.2 shows extensive experiments with real images, where excellent matching is obtained between weakly similar shapes. We demonstrate silhouette matching under partial occlusion, under substantial change of viewpoint, and even when the occluding contours describe different (but related) objects, like two different cars or mammals. Our method is efficient and fast, taking only a few seconds to match two curves.

3.2.1 The proposed matching method

The occluding contours of objects are first extracted from the image and a syntactic representation is constructed, whose primitives are line segments, and whose attributes are length and absolute orientation. Our algorithm then uses a variant of the edit matching procedure combined with heuristic search. Thus we define a novel similarity measure between primitives to assign cost to each edit operation, a novel merge operation, and introduce penalty for interrupting a contour (in addition to the regular deletion/insertion penalty).

More specifically, let A and A' be two syntactic representations of two contours; $A = \{a_1, a_2, \dots, a_N\}$ is a cyclically ordered list of N line segments, and $A' = \{a'_1, a'_2, \dots, a'_{N'}\}$ is another cyclic list of N' segments.

Let a_i be a segment of A and a'_j be a segment of A' . Matching these segments uniquely determines the relative global rotation and scale (2D alignment transformation) between the curves. We assume that the optimal alignment is well approximated by at least one of the NN' possible selections of a_i and a'_j . In fact, we will discuss in Section 3.3.2 a method to prune many of them, leaving us with a set $\Psi \subseteq A \times A'$ of candidate global alignments, such that usually $|\Psi| \ll NN'$.

A member $\{a_i, a'_j\}$ of Ψ (abbreviated $\{i, j\}$ for convenience) denotes a starting point for our syntactic matching algorithm. The algorithm uses edit operations to extend the correspondence between the remaining unmatched segments, preserving their cyclic order. Total cost is minimized using dynamic programming, where the cost of a match between every two segments depends on their attributes, as well as on the attributes of the initial chosen pair (a_i and a'_j). This implicitly takes into account the global alignment operation. The various edit operations and their cost functions are described in Section 3.3.3; the cost of the edit operations can be either negative or positive.¹

We are searching for the member $\{i, j\} \in \Psi$ for which the extended correspondence list has minimal edit cost. Brute force implementation of this search is computationally infeasible. The syntactic matching process is therefore interlaced with heuristic search for the best initial pair in Ψ . Namely, a single dynamic programming extension step is performed for the best candidate in Ψ (possibly a different candidate in each extension matching step), while maintaining the lowest cost achieved by any of the sequences. When no candidate in Ψ has the potential to achieve a lower cost, the search is stopped (see below).

The pseudo-code in Figure 3.1 integrates the components of our matching algorithm into a procedure which gets two syntactic representations and returns a segment correspondence and a dissimilarity value. The arrays which support the dynamic programming are not referenced in this pseudo-code, to increase its readability. We note that the procedure **CURVE-MATCHING** minimizes

¹Negative cost can be interpreted as positive “gain” or “reward”. Every matching prefix has a total (accumulated) cost, which should be as negative as possible. A prefix is never extended by a suffix with positive cost, since this would increase the cost; hence *partial* matching is achieved by leaving the last segments unmatched. Note that if all costs are negative then the minimal cost must be obtained by matching all the segments of the shorter sequence, and if all costs are positive then the minimal cost is trivially obtained by leaving all segments unmatched. The average cost value determines the asymptotic matching length for random sequences [23].

```

procedure CURVE-MATCHING:
input:    two curve representations  $A$  and  $A'$ .
output:  correspondence between line segments and dissimilarity value.

 $\Psi, cost^*, \{i^*, j^*\} \leftarrow \text{INITIALIZE}(A, A')$ 
 $\{i, j\}, potential \leftarrow \text{PICK-CANDIDATE}(\Psi)$ 
While  $cost^* > potential$ 
     $cost^*, \{i^*, j^*\} \leftarrow \text{SYNTACTIC-STEP}(A, A', \{i, j\})$ 
     $\Psi \leftarrow \text{PREDICT-COST}(\{i, j\})$ 
     $\{i, j\}, potential \leftarrow \text{PICK-CANDIDATE}(\Psi)$ 
end-loop
 $correspondence \leftarrow \text{TRACE}(\{i^*, j^*\})$ 
 $dissim \leftarrow \text{DISSIMILARITY}(A, A', correspondence)$ 
return  $correspondence$  and  $dissim$ .

```

Figure 3.1: Pseudo code for CURVE-MATCHING procedure

the edit cost which is typically negative, thus, in effect, CURVE-MATCHING is maximizing the “gain” of matching.

The procedure INITIALIZE performs the initial pruning of pairs of starting points, and returns the set of candidates Ψ sorted by increasing potential values (see below). Full description is given in Section 3.3.2. Its implementation performs a few syntactic matching steps for all NN' possible pairs, and computes the intermediate edit cost corresponding to this partial matching. The minimal intermediate edit cost achieved by any of the candidates is returned as $cost^*$, and the candidate pair which achieves $cost^*$ is returned as $\{i^*, j^*\}$.

The procedure PICK-CANDIDATE selects a particular member of Ψ , to be fed into the syntactic algorithm. To understand its operation, we need to define the concept of potential: For each candidate $\{i, j\}$ that has been extended to a correspondence list of some length, we compute a lower bound on its final edit cost. This bound is based on the intermediate edit cost which has already been achieved, and the cost of the best (lowest cost) possible matching of the remaining unmatched segments. We call this bound the *potential* of the candidate $\{i, j\}$. The procedure PICK-CANDIDATE returns the member of Ψ which has best (minimal) potential.

Technically, we store Ψ as an ordered list, sorted by increasing potential value. Each member of Ψ is a candidate $\{i, j\}$ and its current potential. The procedure PICK-CANDIDATE then returns the first member of Ψ . The list Ψ is initially sorted by INITIALIZE, and its order is maintained by

the procedure **PREDICT-COST** discussed below.

The search for the best candidate $\{i^*, j^*\}$ continues as long as there exists a candidate whose potential is lower than the best cost achieved so far ($cost^*$). It is implemented by the loop which iterates as long as $cost^* > potential$. Note that $cost^*$ cannot increase and $potential$ cannot decrease during the search.

The procedure **SYNTACTIC-STEP** is the core of our algorithm. It is given as input two cyclically ordered sequences $A = \{a_1, \dots, a_N\}$ and $A' = \{a'_1, \dots, a'_{N'}\}$, which are partially matched from position i of A and onward, and from position j of A' and onward. It uses dynamic programming to extend the edit transformation between A and A' by one step. Since our editing cost operation typically takes negative values, the edit cost of $\{i, j\}$ could become better (lower) than $cost^*$. In this case $\{i^*, j^*\}$ is set equal to $\{i, j\}$, and $cost^*$ is set equal to the newly achieved edit cost (otherwise $\{i^*, j^*\}$ and $cost^*$ remain unchanged). Sections 3.3.3 and 3.3.4 give the full description of the procedure **SYNTACTIC-STEP**.

Extending the editing sequence of $\{i, j\}$ is likely to increase its potential, making it a less attractive candidate. This is because the potential of $\{i, j\}$ is partially determined by a lower bound on the final edit distance between the yet unmatched segments, and the edit operation just added can only tighten this bound by decreasing the number of unmatched segments. The procedure **PREDICT-COST** re-estimates the final cost, and corrects the potential of $\{i, j\}$. Since Ψ is kept as an ordered list, **PREDICT-COST** pushes the candidate $\{i, j\}$ down to maintain the order of the list. Section 3.3.5 gives full details of the potential estimation.

Assuming that the reader is familiar with conventional dynamic programming implementations, it is sufficient to describe the procedure **TRACE** as the procedure which reads the lowest cost path from the dynamic programming array². When this procedure is applied to the array associated with the best candidate $\{i^*, j^*\}$, the lowest cost editing sequence is obtained. In our implementation, in order to keep space complexity low, we keep just the last few rows and columns for each array, hence the procedure **TRACE** needs to repeat the syntactic matching for the best pair $\{i^*, j^*\}$. See Section 3.3.4 for a description of the dynamic programming implementation.

Finally, using the correspondence we found, we refine the global $2D$ alignment by minimizing the sum of residual distances between matched segments endpoints. The procedure **DISSIMILARITY**

²Note, however, that using both positive and negative costs allows for partial matching, hence the path can terminate at any entry of the dynamical programming array and not necessarily at the last row or column.

performs the minimization, and uses the residual distances to define a robust measure of dissimilarity between the curves (details in Section 3.3.6).

Our approach thus combines syntactic matching with a proximity measure (in this sense our method resembles that of [2]). That is, we establish feature correspondence using syntactic matching, and then evaluate the dissimilarity according to the residual distances between matched points. We *do not* use the edit distance as a measure of dissimilarity, mainly due to the fact that this quantity depends on the somewhat arbitrary parameters of the edit operation and segment similarity, whereas typically the best *matching* result is not sensitive to these exact parameters. That is, the same matching is obtained for a range of edit parameter values, although the edit distance may be different. Another advantage to combining syntactic and proximity criteria is that in many cases the combination provides a mechanism for outliers removal, as is demonstrated in Section 3.3.6.

3.2.2 Matching results

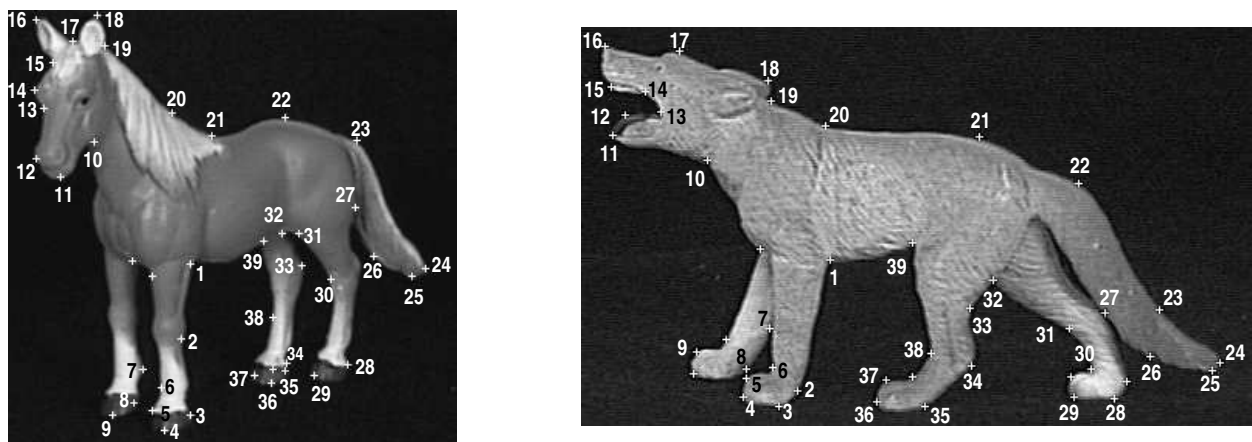


Figure 3.2: Qualitative matching between pictures of toy models of a horse and a wolf. Note the correct correspondence between the feet of the wolf to those of the horse, and the correspondence between the tails. The results are shown without outliers pruning. In this example, all the features to which no number is attached had been merged; e.g. the segment 9-10 on the horse outline was matched with 3 segments on the wolf outline.

We now present a few image pairs and triplets together with the matching results, which demonstrate perceptually appealing matching. In Section 3.2.3 below we apply our matching algorithm to a database of 31 silhouettes given to us by Sharvit & Kimia, and compare our dissimilarity values to those reported in [122]. The classification results which will be shown in Chapter 4, using the matching of a few thousands image pairs, may be regarded as an objective examination of the matching quality.

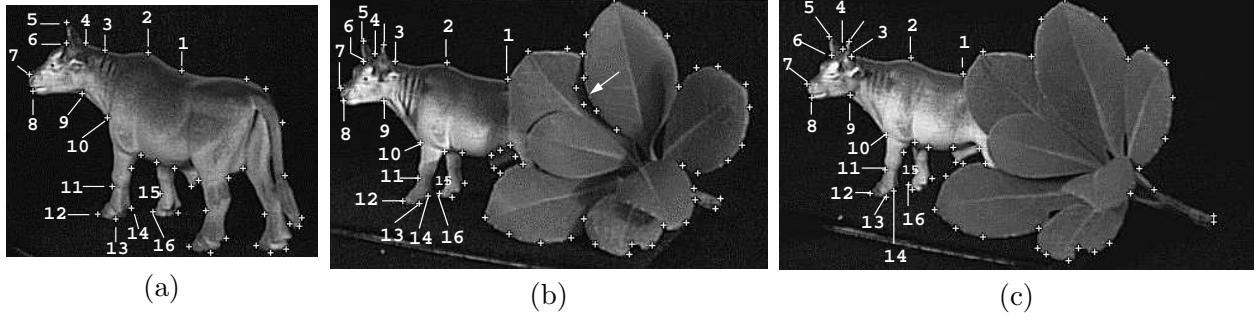


Figure 3.3: Dealing with occlusion: partial matching between three images. Points are mapped from (a) to (b) to (c) and back to (a). Only points which are mapped back to themselves are accepted (order is not important). The points on the tail in (a) are matched with the shadow (pointed by the arrow) in (b), but matching (b) with (c) leaves the shadow unmatched. Hence the tail is not matched back to itself, and the correspondence with the shadow is rejected.

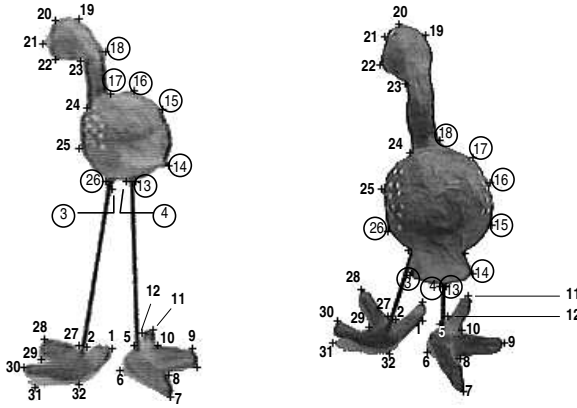


Figure 3.4: Matching two views of an object subjected to large foreshortening. Rejected pairs (in circles) were detected in four iterations of eliminating the (10%) most distant pairs and re-aligning the others. 33 and 35 features were extracted on the two outlines; 32 pairs were initially matched, and 9 pairs were rejected (28%).

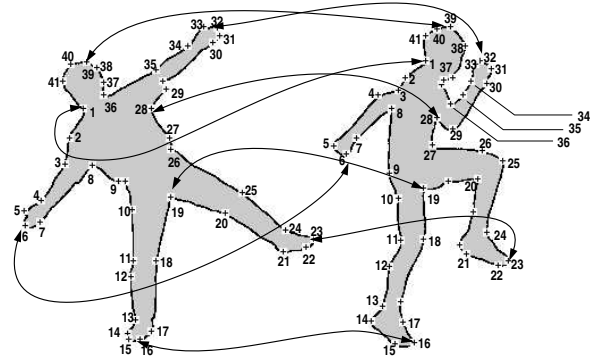


Figure 3.5: Matching of human limbs at different body configurations. In this case the outlines were extracted with snakes rather than by gray level clustering (see acknowledgments). Original images are not shown.

In all the experiments reported in this paper we use the same parameter values (defined in Section 3.3): $w_1 = 1$, $w_2 = 0.8$, $w_3 = 8.0$ and $K = 4$ (with the exception of Figure 3.6, where $K = 5$). Each matching of an image pair took only a few seconds (see Chapter 4).

Figure 3.2 shows two images of different objects. There is a geometrical similarity between the two silhouettes, which has nothing to do with the semantic similarity between them. The geometrical similarity includes five approximately vertical swellings or lumps (which describe the four legs and the tail). In other words, there are many places where the two contours may be considered locally similar. This local similarity is captured by our matching algorithm.

The two occluding contours of the two animals and the feature points were automatically

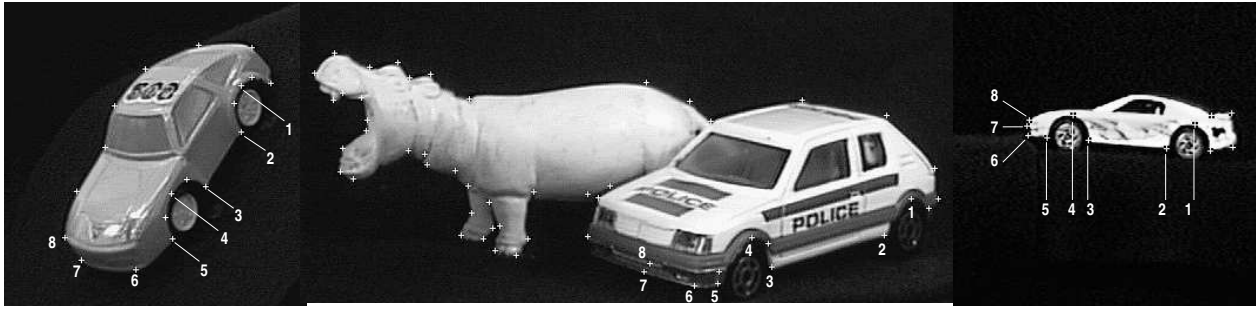


Figure 3.6: Combination of various sources of difficulty: different models, different viewpoints and occlusion. The merging utility is used to overcome the different number of feature points around the wheels; gap insertion is utilized to ignore the large irrelevant part.

extracted in the preprocessing stage. Corresponding points are marked in Figure 3.2 by the same numbers. Hence the tails and feet are nicely matched, although the two shapes are only weakly similar. The same matching result is obtained under arbitrarily large rotation and scaling of one image relative to the other.

Figure 3.3 demonstrates the local nature of our algorithm, namely, that partial matching can be found when objects are occluded. Since our method does not require global image normalization, the difference in length between the silhouette outlines does not impede the essentially perfect matching of the common parts. Moreover, the common parts are not identical (note the distance between the front legs and the number of ears) due to a small difference in viewpoint; this also does not impede the performance of our algorithm.

Figure 3.3 also demonstrates outliers pruning using three images. In image 3.3b there is a shadow between two of the leaves (pointed by the arrow), and as a result the outline penetrates inward. The feature points along the penetration are (mistakenly) matched with features along the tail in image 3.3a, since the two parts are locally similar. However, we use the procedure of mapping the points of 3.3a to 3.3b, then to 3.3c and back to 3.3a. Only points which are mapped back to themselves are accepted as correct matches; these matched are marked by common numbers in Figure 3.3.

Figures 3.4 and 3.6 show results when matching images taken from very different points of view. In Figure 3.4 two different views of the same object are matched, and the method of iterative elimination of distances is demonstrated (see Section 3.3.6). Figure 3.6 shows matching between three different cars, viewed from very different viewpoints and subjected to occlusion. Matching under a large perturbation of viewpoint can be successful as long as the silhouettes remain similar

“enough”. Note that preservation of shape under change of viewpoint is a quality that defines “canonical” or “stable” views. Stable images of 3D objects were proposed as the representative images in an appearance based approach to object representation [146].

The last example (Figure 3.5) shows results with an articulated object, matching human limbs at different body configurations.

3.2.3 Dissimilarity measurements: comparison












































																		
   	0 30 59 37	77 183 118 103	164 124 116 142	120 109 113 145	116 111 114 93	125 144 172 143 120												
	30 0 50 25	123 84 142 146	111 137 109 115	111 112 109 113	82 107 153 137	106 110 117 132 123												
	59 50 0 47	117 85 122 142	110 200 73 142	108 111 109 112	91 102 132 113	134 141 124 140 135												
	37 25 47 0	106 112 144 108	125 120 104 137	109 122 126 140	106 111 109 104	143 169 163 148 111												
   	77 123 117 106	0 21 59 47	105 127 163 139	105 140 143 123	150 145 112 101	144 187 163 134 115												
	183 84 85 112	21 0 89 78	124 116 134 84	157 125 108 128	64 132 140 129	177 142 171 140 150												
	118 142 122 144	59 89 0 65	103 140 116 179	122 141 132 122	85 107 127 95	134 117 133 162 139												
	103 146 142 108	47 78 65 0	114 92 127 82	97 136 118 95	81 180 128 131	128 91 116 149 117												
   	164 111 110 125	105 124 103 114	0 76 22 61	58 68 67 73	107 75 77 95	87 93 68 87 69												
	124 137 200 120	127 116 140 92	76 0 28 78	74 82 63 62	72 115 74 91	91 115 99 92 83												
	116 109 73 104	163 134 116 127	22 28 0 86	80 72 81 127	92 77 94 113	93 113 82 114 74												
	142 115 142 137	139 84 179 82	61 78 86 0	94 65 71 85	63 85 80 107	123 127 98 93 88												
   	120 111 108 109	105 157 122 97	58 74 80 94	0 24 26 51	77 92 74 83	91 141 66 108 93												
	109 112 111 122	140 125 141 136	68 82 72 65	24 0 45 67	64 46 66 122	106 77 92 92 117												
	113 109 109 126	143 108 132 118	67 63 81 71	26 45 0 53	96 65 80 65	101 106 101 97 83												
	145 113 112 140	123 128 122 95	73 62 127 85	51 67 53 0	91 83 67 81	70 72 62 98 55												
   	116 82 91 106	150 64 85 81	107 72 92 63	77 64 96 91	0 21 28 20	143 140 132 124 86												
	111 107 102 111	145 132 107 180	75 115 77 85	92 46 65 83	21 0 15 29	125 122 117 134 119												
	114 153 132 109	112 140 127 128	77 74 94 80	74 66 80 67	28 15 0 21	121 174 141 158 124												
	93 137 113 104	101 129 95 131	95 91 113 107	83 122 65 81	20 29 21 0	123 90 123 138 110												
    	125 106 134 143	144 177 134 128	87 91 93 123	91 106 101 70	143 125 121 123	0 14 20 20 19												
	144 110 141 169	187 142 117 91	93 115 113 127	141 77 106 72	140 122 174 90	14 0 16 19 15												
	172 117 124 163	163 171 133 116	68 99 82 98	66 92 101 62	132 117 141 123	20 16 0 30 25												
	143 132 140 148	134 140 162 149	87 92 114 93	108 92 97 98	124 134 158 138	20 19 30 0 34												
	120 123 135 111	115 150 139 117	69 83 74 88	93 117 83 55	86 119 124 110	19 15 25 34 0												

Table 3.1: The dissimilarity values computed between 25 silhouettes (multiplied by 1000 and rounded). For each line, the columns that correspond to the three nearest neighbors (and the self zero distance) are highlighted. The first, second and third nearest neighbor are in the same class a fraction of 25/25, 21/25 and 19/25 of the times respectively.

In this section we use our matching algorithm to compute a dissimilarity value, as will be explained in Section 3.3.6. Good matching is essential for correct dissimilarity estimation, whose values we use for quantitative comparisons with other methods. We use the image database created by Sharvit & Kimia (see [122]), which consists of 31 silhouettes of 6 classes of objects (including fish,airplanes, tools).

In [122], 25 images were selected out of this database, and their pairwise similarities were computed. The measure of quality was the number of instances (out of 25) in which the first, second and third nearest neighbor of an image was found in its own class. We follow the same procedure, and show in Table 3.1 the dissimilarity values which we obtain for 25 silhouettes.³

We find that the fraction of times (out of 25) that the first nearest neighbor of an image belongs to its own class is 25/25, namely, it is always the case. For the second and third nearest neighbors the results are 21/25 and 19/25. In comparison, the results reported in [122] are 23/25, 21/25 and 20/25 respectively, whereas our results for their choice of 25 images are the fractions 25/25, 20/25 and 17/25 respectively. It is to be noted, however, that in the framework of [122] one can use additional information, specifically, whether the two graphs that represent a pair of shapes have similar topology.

We conclude that the two methods are comparable in quality when isolated silhouettes are matched. This is in spite of our using boundary representation, whereas symmetry representation (shock graph) is used in [122]. The shock graph representation is inherently more sensitive to occlusions, while shock graph matching requires solving the difficult NP-complete graph isomorphism problem (see Section 3.1). Moreover, our method can easily be adjusted to handle open curves, by avoiding the assumption that the syntactic representation is cyclic. On the other hand, shock graphs must distinguish between interior and exterior.

Recently, progress has been made toward computing the edit distance between shock graphs [134] using a polynomial time algorithm that exploits their special structure. So far, however, the algorithm is not capable of dealing with invariance to image transformations, and no quantitative measures have been reported.

3.3 Flexible syntactic matching: details

In this section we give the details of the various procedures and steps involved in the curve matching algorithm, outlined in Section 3.2.1. Contour representation is discussed in Section 3.3.1. The initial pruning of candidate global alignments is discussed in Section 3.3.2, where we describe the procedure INITIALIZE. The syntactic edit operations which are used by SYNTACTIC-STEP, and

³As table 3.1 shows, we have at least 4 images in each class. In [122] the same images were chosen, with the exception that one of the classes consisted of only 3 images, while the fish class contained 5 images. However, for members of a class consisting of only 3 images, the three nearest neighbors can no longer be all in the same class. Hence we modified slightly the choice of selected images.

their respective costs, are discussed in Section 3.3.3. The details of the dynamic programming procedure, which **SYNTACTIC-STEP** uses to minimize the edit distance, are given in Section 3.3.4. In Section 3.3.5 we define the potentials which are used to guide the search for best starting point and describe the procedure **PREDICT-COST**. Finally, the procedure **DISSIMILARITY** is discussed in Section 3.3.6.

3.3.1 Preprocessing and contour representation

In the examples shown in this paper, objects appear on dark background, and segmentation is successfully accomplished by a commercial k-means segmentation tool. A syntactic representation of the occluding contour is then automatically extracted: it is a polygon whose vertices are either points of extreme curvature, or points which are added to refine the polygonal approximation. Thus the primitives of our syntactic representation are line segments, and the attributes are length and absolute orientation. The number of segments depends on the chosen scale and the shape of the contour, but typically it is around 50. Coarser scale descriptions may be obtained using merge operations.

Feature points (vertices) are initially identified at points of high curvature, according to the following procedure: at every contour pixel p an angle ρ is computed between two vectors u and v . The vector u is the vectorial sum of m vectors connecting p to its m neighboring pixels on the left, and the vector v is similarly defined to the right. Points where ρ is locally minimal are defined as feature points. The polygonal approximation (obtained by connecting these points by straight lines) is compared with the original outline. If the distance between the contour points to the polygonal segments is larger than a few pixels, more feature points are added to refine the approximation.

3.3.2 Global alignment: pruning the starting points

The procedure **INITIALIZE** receives two cyclic sequences A and A' of lengths N and N' respectively, as defined in Section 3.2.1. Initially there are NN' possible starting points for the syntactic matching procedure; recall that each starting point corresponds to the matching of one segment a_i in A to another segment a_j in A' , thus defining the global $2D$ alignment between the two curves. However, the number of successful starting points is much smaller than NN' , and they tend to correspond to similar global $2D$ alignment transformations for two reasons: (i) Low cost transformations tend

to be similar since any pair of segments $\{a_i, a'_j\}$, which belongs to a good correspondence list, is likely to be a good starting point for the syntactic algorithm. (ii) The overall number of good starting points tends to be small since most of the pairs in $A \times A'$ cannot be extended to a low cost correspondence sequence; the reason is that it might be possible to find a random match of short length, but it is very unlikely to successfully match long random sequences.

These observations are used by the procedure **INITIALIZE** to reduce significantly the number of candidate starting points. The procedure uses as a parameter the number t of edit operations which are performed for every one of the NN' possible starting points (in our experiments we use $t=5$ or 10). The pruning proceeds using the relation between starting points and global $2D$ alignments, as follows.

Every starting point $\{a_i, a'_j\}$ is associated with a global $2D$ alignment, and in particular with a certain rotation angle that maps the direction of a_i to that of a'_j . Let n be $\min(N, N')$, and observe the distribution of the n rotation angles which achieved the best n edit distances after t steps. If these angles are distributed sharply enough around some central value c , we conclude that c is a good estimator for the global rotation. Then, we discard every starting point in $A \times A'$ whose associated rotation is too far from c . The remaining set of candidates is the set Ψ (see Figure 3.7).

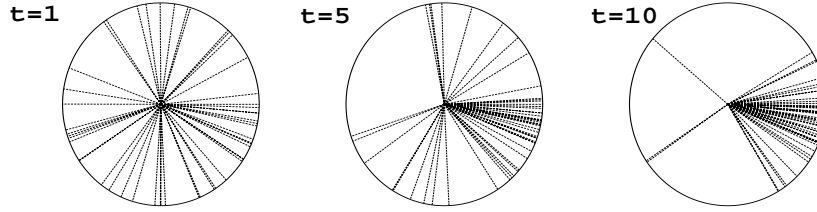


Figure 3.7: An example of initial pruning. Two curves with $n = \min(N, N') = 50$ are matched syntactically using t edit steps (see text for details). The 50 candidates which achieve best (minimal) edit cost are examined, to see whether the distribution of their associated rotations shows central tendency. Here we sample the distribution after 1, 5 and 10 syntactic steps. In this example 5-10 steps are sufficient for a reliable estimation of the global rotation angle c . We proceed by eliminating the candidates whose associated global rotation is too far from c .

For each candidate that remains in Ψ the procedure **INITIALIZE** computes its future potential, as discussed in Section 3.3.5, and sorts the list Ψ by increasing potential values. The minimal edit distance (cost) that has been achieved during the first t steps is returned as $cost^*$, and the candidate possessing this cost is returned as $\{i^*, j^*\}$.

3.3.3 Syntactic operations which determine the edit distance

The goal of classical string edit algorithms is to find a sequence of elementary edit operations, which transform one string into another at a minimal cost. The elementary operations are substitution, deletion and insertion of string symbols. Converting the algorithm to the domain of vision, symbol substitution is interpreted as matching two shape primitives, and the substitution cost is replaced by the dissimilarity between the matched primitives. The dissimilarity measure is discussed in Section 3.3.3. Novel operations involving gap insertion and the merging of primitives are discussed in Sections 3.3.3 and 3.3.3.

Similarity between primitives

We define now the similarity between line segments a_k and a'_l . The cost of a substitution operation is this value with a minus sign, hence the more similar the segments are, the lower their substitution cost is. We denote the attributes of a_k, a'_l - orientation and length - by (θ, ℓ) and (θ', ℓ') respectively. The ratio between the length attributes is denoted relative scale $c = \ell/\ell'$.

The term “reference segments” refers to the starting point segments, which implicitly determine the global rotation and scale that aligns the two curves (as discussed above). The reference segments are specified by the argument $\{i, j\}$ in the call to the procedure **SYNTACTIC-STEP**, and are denoted here a_0, a'_0 . The segment similarity also depends on the corresponding attributes - orientation, length and relative scale - of the reference segments: (θ_0, ℓ_0) , (θ'_0, ℓ'_0) and $c_0 = \ell_0/\ell'_0$.

We first define the component of similarity which is determined by the length (or relative scale) attribute of two matching segments. We map the two matched pairs of length values $\{\ell, \ell'\}$ and $\{\ell_0, \ell'_0\}$ to two corresponding directions in the (ℓ, ℓ') -plane, and measure the angle between these two directions. The cosine of twice this angle is the length-dependent component of our measure of segment similarity (Figure 3.8). This measure is numerically stable. It is not sensitive to small scale changes, nor does it diverge when $c_0 = \ell_0/\ell'_0$ is small. It is measured in intrinsic units between -1 and 1 . The measure is symmetric, so that the labeling of the contours as “first” and “second” has no effect.

Let δ be the angle between the vectors $[\ell, \ell']$ and $[\ell_0, \ell'_0]$. Our scale similarity measure is:

$$S_\ell(\ell, \ell' | \ell_0, \ell'_0) = \cos 2\delta = \frac{4cc_0 + (c^2 - 1)(c_0^2 - 1)}{(c^2 + 1)(c_0^2 + 1)} \quad (3.1)$$

S_ℓ thus depends explicitly on the scale values c and c_0 rather than on their ratio, hence it cannot be

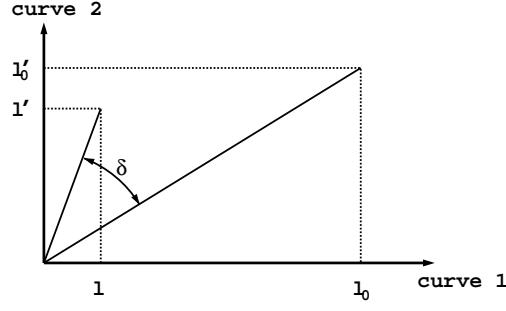


Figure 3.8: Length similarity is measured by comparing the corresponding reference length values $[\ell_0, \ell'_0]$ with the corresponding length values of the current segment $[\ell, \ell']$. Each length pair is mapped to a direction in the plane, and similarity is defined as $\cos(2\delta)$. This value is bounded between -1 and 1.

computed from the invariant attributes ℓ/ℓ_0 and ℓ'/ℓ'_0 . The irrelevance of labeling can be readily verified, since $S_\ell(c, c_0) = S_\ell(c^{-1}, c_0^{-1})$.

We next define the orientation similarity S_θ between two line segments whose attributes are θ and θ' respectively. The relative orientation between them is measured in the trigonometric direction (denoted $\theta \rightarrow \theta'$) and compared with the reference rotation ($\theta_0 \rightarrow \theta'_0$):

$$S_\theta(\theta, \theta' | \theta_0, \theta'_0) = \cos [(\theta \rightarrow \theta') - (\theta_0 \rightarrow \theta'_0)] \quad (3.2)$$

As with the scale similarity measure, the use of the cosine introduces non-linearity; we are not interested in fine similarity measurement when the two segments are close to being parallel or anti parallel. Our matching algorithm is designed to be flexible, in order to match curves that are only weakly similar; hence we want to encourage segment matching even if there is a small discrepancy between their orientations. Similarly, the degree of dissimilarity between two nearly opposite directions should not depend too much on the exact angle between them. On the other hand, the point of transition from acute to obtuse angle between the two orientations seems to have a significant effect on the degree of similarity, and therefore the derivative of S_θ is maximal when the line segments are perpendicular.

Finally, the combined similarity measure is defined as the weighted sum:

$$S = w_1 S_\ell + S_\theta \quad (3.3)$$

The positive weight w_1 (which equals 1 in all our experiments) controls the coupling of scale and orientation similarity.

Gap opening

In string matching, the null symbol λ serves to define the deletion and insertion operations using $a \rightarrow \lambda$ and $\lambda \rightarrow a$ respectively, where a denotes a symbol. In our case, a denotes a line segment and λ is interpreted as a “gap element”. Thus $a \rightarrow \lambda$ means that the second curve is interrupted and a gap element λ is inserted into it, to be matched with a . We define, customarily, the same cost for both operations, making the insertion of a into one sequence equivalent to its deletion from the other.

The cost of interrupting a contour and inserting ξ connected gap elements into it (that are matched with ξ consecutive segments on the other curve) is defined as $w_3 - w_2 \cdot \xi$, where w_2, w_3 are positive parameters. Thus we assign a penalty of magnitude w_3 for each single interruption, which is discounted by w_2 for every individual element insertion or deletion. This predefined quantity competes with the lowest cost (or best reward) that can be achieved by ξ substitutions. A match of ξ segments whose cost is higher (less negative) than $w_3 - w_2 \cdot \xi$ is considered to be poor, and in this case the interruption and gap insertion is preferred.

In all our experiments we used $w_2 = 0.8$ and $w_3 = 8.0$. (These values were determined in an ad-hoc fashion, and not by systematic parameter estimation which is left for future research). These numbers make it possible to match a gap with a long sequence of segments, as required when curves are partially occluded. On the other hand, isolated gaps are discouraged due to the high interruption cost. Our algorithm therefore uses deletions in cases of occlusion, while for local mismatches it uses the merging operation, which is described next.

Segment merging

One advantage of using variant attributes (length and absolute orientation) is that segment merging becomes possible. We use segment merging as the syntactic homologue of curve smoothing, accomplishing noise reduction by local resolution degradation. Segment merging, if defined correctly, should simplify a contour representation by being able to locally and adaptively change its scale from fine to coarse.

We define the following merging rule: two adjacent line segments are replaced by the line connecting their two furthest endpoints. If the line segments are viewed as vectors oriented in the direction of propagation along the contour, then the merging operation of any number of segments

is simply their vectorial sum.⁴ The cost of a merge operation is defined as the dissimilarity between the merged segment and the one it is matched with. A comparison of this rule with the literature is given in the appendix.

3.3.4 Minimizing edit cost via dynamic programming

Procedure SYNTACTIC-STEP is given as input two cyclically ordered sequences $A = \{a_1, \dots, a_N\}$ and $A' = \{a'_1, \dots, a'_{N'}\}$, which are partially matched from position i of A and onward, and from position j of A' and onward. It uses dynamic programming to extend the edit transformation between A and A' by one step. The edit operations and their cost functions were discussed above in Section 3.3.3.

The notation used in Section 3.2.1 hides, for simplicity, the workspace arrays which are used for path planning. Let us assume that the procedure SYNTACTIC-STEP can access an array R which corresponds with the starting point $\{i, j\}$. The entry $R[\mu, \nu]$ holds the minimal cost that can be achieved when the μ segments of A following a_i are matched with the ν segments of A' following a'_j . We will see later that it is not necessary to keep the complete array in memory.

A syntactic step is the operation of getting an array R which is partially filled, and extending it by filling some of the missing entries. Our choice of extension is called “block completion”. Let us assume for simplicity that the reference segments $\{i, j\}$ are the first ones. There is no loss of generality here, since the order in A, A' is cyclic. Initially the computed portion of the array R corresponds to a diagonal block, whose two corners are $R[1, 1]$ and $R[\mu, \nu]$. When the procedure SYNTACTIC-STEP is applied to R , it extends R by filling in another row (of length ν), and/or another column (of length μ).

The decision whether a row or a column is to be added depends on the previous values of R , and is related to the potential computation that is discussed below in Section 3.3.5. Roughly speaking, we add a row (column) if the minimum of R is in the last computed row (column), and we add both if the minimum is obtained in the last computed corner.

When extending the block of size $\mu \times \nu$, every new entry $R[k, l]$ is computed according to the

⁴Implementation note: it turns out to be more convenient to keep for every line segment the attributes $(\ell, \sin \theta, \cos \theta)$ instead of (ℓ, θ) . It is straightforward to express the orientation similarity S_θ using the new attributes (and S_ℓ is, of course, unaffected). The advantage is that merging segments does not involve the computation of any trigonometric functions. Thus after merge, using the input attributes $(\ell_i, \sin \theta_i, \cos \theta_i)$ we define $x = \sum_i \ell_i (\cos \theta_i)$ and $y = \sum_i \ell_i (\sin \theta_i)$, and then $\ell = \sqrt{x^2 + y^2}$, $\sin \theta = x/\ell$ and $\cos \theta = y/\ell$.

following rule:

$$R[k, l] = \min\{r_1, r_2, r_3\} \quad (3.4)$$

where

$$\begin{aligned} r_1 &= \min_{\alpha, \beta \in \Omega} \{R[\alpha, \beta] - S(\overline{\alpha k}, \overline{\beta l})\} \\ r_2 &= \min_{0 < \alpha < k} \{R[\alpha, l] + w_3 - w_2 \cdot (k - \alpha)\} \\ r_3 &= \min_{0 < \beta < l} \{R[k, \beta] + w_3 - w_2 \cdot (l - \beta)\} \end{aligned}$$

\overline{xy} denotes the vectorial sum of the segments $(x + 1), \dots, y$, and the domain Ω is defined below.

Unlike in the “classical” editing algorithm, the term r_1 is computed over a domain Ω , generalizing the simple substitution operation to the substitution of merged segments. We define Ω as a triangular subregion of R which contains $K(K - 1)/2$ entries (Figure 3.9). Namely:

$$\Omega = \{\alpha, \beta \mid 0 < \alpha < k, 0 < \beta < l, (k - \alpha) + (l - \beta) \leq K\}$$

and the computation of r_1 involves $K(K - 1)/2$ evaluations of alternatives merges. The minimal value of K is 2, which is the “classical” substitution (diagonal) step.

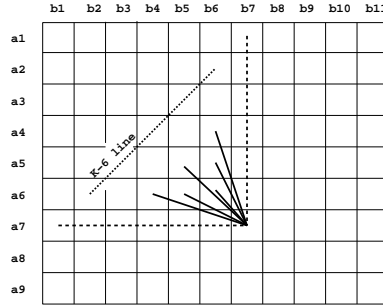


Figure 3.9: The entry $R[k, l]$ is updated according to Equation (3.4). Solid lines represent merge steps (with the “classical” substitution as a special case). Dashed lines represent the interruption and deletion of ξ connected elements.

The minimization domains for r_2 and r_3 are illustrated in Figure 3.9 by the horizontal and vertical dashed lines, generalizing the single element deletion operation to the deletion of ξ elements. This operation is associated with an interruption penalty (w_3) and a guaranteed reward ($w_2\xi$), which competes with the reward of substitutions.

We note that in order to find the minimal value for r_2 along the vertical line, it is not necessary to compare all its k possible values. It is sufficient to keep one index (α_0) for each column l , such that $r_2 = R[\alpha_0, l] + w_3 - w_2(k - \alpha_0)$. The initial value of α_0 is 1, and after entry $R[k, l]$ has been

evaluated, the value of α_0 should be set to k iff $R[k, l] \leq R[\alpha_0, l] - w_2(k - \alpha_0)$. A similar argument applies to the computation of r_3 . Hence both the computation of r_2 and r_3 have complexity $O(1)$.

Finally, according to the block completion scheme of the array R defined above, it is easily verified that only the last $\lfloor K/2 \rfloor$ rows and columns of the block need to be stored in memory. On the other hand, if only these entries are kept, the complete path cannot be restored after the minimal edit cost is found. Hence in our implementation the procedure **TRACE** (see Section 3.2.1) repeats the syntactic procedure for the array possessing the best result, while keeping all of it in memory. It then follows the best path, which ends at the entry that achieved minimal edit cost, and returns it as the final matching.

3.3.5 Potential: bounding the edit distance

In order to guide the search for the best starting point, which is interlaced with the syntactic steps, it is necessary to bound the final minimal cost of a partially computed matching. We use a tight lower bound on the edit cost estimation, termed “potential”. The optimality of the search is thus guaranteed. In this section we define the potential function, and explain how it is used by the procedure **PREDICT-COST**.

Let us consider an entry $R[k, l]$ in the dynamic programming array R . Without loss of generality, due to the cyclic segment order, we can assume that the forward path leaving this entry lies in the rectangular block defined by the corners $R[k, l]$ and $R[N, N']$. We define η and ζ to be the dimensions of this rectangle: $\eta = \min(N - k, N' - l)$, and $\zeta = \max(N - k, N' - l)$. We also define ϵ to be the maximal similarity between segments ($\epsilon = 1 + w_1$ from Equation (3.3)). The lowest cost substitution thus has the negative cost $-\epsilon$, which acts as a reward.

The values of the parameters w_1, w_2, w_3 are chosen to guarantee that a diagonal path consisting of the lowest cost substitutions has the minimal possible cost⁵. Hence the lowest cost forward path originating from $R[k, l]$ must contain a diagonal segment of maximal length, namely of length η . The cost assigned with this part is $-\epsilon\eta$. In addition to the diagonal part, the forward path may contain a vertical or horizontal part of length $\zeta - \eta$. In general, this part can decrease the cost only if $w_3 - w_2(\zeta - \eta)$ is negative. However, in the case where the value of $R[k, l]$ is set equal to r_2 or r_3 in Equation (3.4), it is the case that the entry $[k, l]$ is already considered to be inside a gap. In

⁵A sufficient condition is that $w_2 < \epsilon/2 = \frac{1+w_1}{2}$. If w_3 is nonzero, it is sufficient that $w_2 - \epsilon/2 < w_3/x$, where x is the largest possible diagonal path, namely $x = \min(N, N')$

this case we can first extend the gap by $\zeta - \eta$ moves, and only then continue with η diagonal steps. Thus the interruption penalty is avoided, and the forward path cost becomes $-\epsilon\eta - w_2(\zeta - \eta)$. Combining this bound with the value of $R[k, l]$, we get a lower bound f_{kl} on the minimal edit cost which can be achieved passing through $R[k, l]$:

$$f_{kl} = \begin{cases} R[k, l] - \epsilon\eta + \min\{0, w_3 - w_2(\zeta - \eta)\} & \text{if } R[k, l] = r_1 \text{ in Equation (3.4)} \\ R[k, l] - \epsilon\eta - w_2(\zeta - \eta) & \text{if } R[k, l] = r_2, r_3 \text{ in Equation (3.4)} \end{cases}$$

We refer to f_{kl} as an entry potential, and define the potential of the array R as the minimum over active entry potential. An entry is active if there is a future path computation that will use its value. Hence the entries occupying the last $\lfloor K/2 \rfloor$ rows and columns of the evaluated block are active.

The procedure **PREDICT-COST** receives as input a candidate pair of starting points segments denoted $\{i, j\}$. It computes its potential, which is the potential of the array R associated with this pair (evidently the procedure must access the dynamic programming array R). The procedure **PREDICT-COST** then updates the list Ψ with the new potential value. In our implementation we keep the list Ψ sorted by increasing potentials, and the procedure **PREDICT-COST** places the candidate $\{i, j\}$ in its proper position, after computing its potential.

3.3.6 Computing curve similarity after matching

When the line segments are syntactically matched, the procedure **DISSIMILARITY** transforms the residual distances between their endpoints into a robust dissimilarity value. We may use the edit distance itself, but this quantity has two drawbacks. It depends on the somewhat arbitrary parameters of the edit operations (whereas typically the best *matching* result is not sensitive to these parameters), and it decreases without bound with increasing edited length. Normalizing the edit distance with respect to the curve length is not a trivial task [91].

Instead of the minimal edit distance we use its corresponding matching sequence to compute a robust proximity measure. A closed form expression for the similarity transformation (rotation, translation and scale) that minimizes the sum of squared distances between matched point sets is given in [148]. Applying this transformation to our matched points refines the alignment that was assumed during matching, since at that time the alignment was chosen from a finite set of NN' candidates. After optimal alignment we define the dissimilarity d to be equal to the k^{th} shortest residual distance, where $k = \frac{1}{2} \min(N, N')$.

In the case of complete matching, the number of matched features is $\min(N, N')$, and d becomes the median of the residual distances. The median is known to be a robust estimator, since it is not affected by extreme outliers. When many feature points are left unmatched, as happens when the two curves are not similar, d becomes larger than the median (since k is defined by the total numbers of points, N and N'). This increases our dissimilarity measure, reflecting the fact that the matching is partial. If the number of matched features is smaller than k , the dissimilarity is defined to be ∞ .

We also report on two other techniques which we have found useful for identifying and removing outliers. The first is iterative elimination: in every iteration the matched pairs that are most distant (after optimal $2D$ alignment) are eliminated, and the rest are re-aligned. We chose to eliminate 10% of the pairs at every iteration. The motivation is the following: features are matched when the local pieces of curve around them have similar shape; if after alignment they are also proximal, meaning that they agree with the global alignment, then the match is likely to be correct.

The other pruning technique can be used when three related images are available (rather than two). Assume that a feature point p on contour 1 is matched with point p' on contour 2, and p' is matched with p'' on contour 3. If the matching between 1 and 3 supports the mapping between p and p'' , then the correspondence list $(p \leftrightarrow p', p' \leftrightarrow p'', p \leftrightarrow p'')$ is accepted.

3.3.7 Discussion and comparison to other methods

Below we discuss some issues relating to complexity, invariance and direct proximity computation. A detailed comparison of our syntactic operations to other methods is given in the appendix.

Complexity

The algorithm develops $|\Psi|$ dynamic programming arrays, and when it terminates each array has been completed up to a block of some size. Let n^2 be the average number of entries in a completed block. The total number of computed entries is therefore $|\Psi|n^2$. Clearly n is a fraction of N . From the considerations discussed in Section 3.3.2 it follows that $|\Psi|$ is typically of the order of N as well. It is possible, although not used here, to constrain the procedure INITIALIZE to return a set Ψ of size $\min(N, N')$ exactly. The overall number of computed entries is therefore $O(N^3)$.

Every single entry computation is of complexity $O(K^2)$, since $K(K-1)/2$ alternative evaluations are required to compute r_1 in Equation (3.4). Note that K is usually a small constant (we used

$K=4$). An entry computation is followed by updating the array potential. We maintain for each row and column the best score achieved there, hence the array potential is computed in $O(K)$ time after a block is completed.

How to achieve invariance

Available syntactic matching methods usually achieve scale and rotation invariance (if at all) by using invariant attributes. The benefit of using invariant attributes is efficiency. The drawback is that invariant attributes cannot be smoothed by merging, and they are either non local or non interruptible. For example, in [89] the orientation of a line segment is measured with respect to its successor, hence the opening of a gap between segments introduces ambiguity into the representation (see Figure 3.10). In [43] the attributes which describe curve fragments are Fourier coefficients, and in [2] an attribute called “sphericity” is defined. Both are invariant attributes, but non-interruptible⁶.

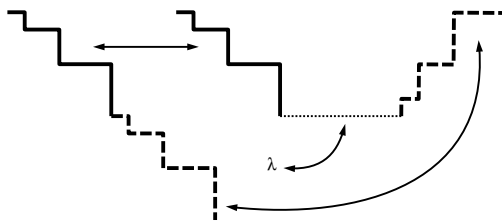


Figure 3.10: When the primitive attribute is measured relative to a preceding primitive, interrupting the sequence creates problems. Here, for example, the orientation information is lost when the dotted segment is matched with a gap element. As a result, the two contours may be matched almost perfectly to each other and considered as very similar.

Moreover, it seems to be impossible to find operators on invariant attributes that are equivalent to smoothing in real space. Instead, a cascade of different scale representations must be used [139], where few fragments may be replaced by a single one which is their “ancestor” in a scale space description. This requires massive preprocessing, building a cascade of syntactical representations for each curve with consistent fragment hierarchy.

In contrast, our algorithm is invariant with respect to scaling, rotation and translation without relying on invariant attributes, while remaining efficient and capable of comparing complex

⁶The Fourier coefficients are normalized individually, which means that if every fragment undergoes a different rigid or scaling transformation, the representation remains unchanged. The sphericity representation behaves in the same way. The relative size and orientation information is preserved as long as the sequence is not interrupted, since overlapping fragments are used. Note that in spite of this property the algorithms are applied to partial matching in the framework of model based recognition, since the solution that preserves the correct relative size and orientation information between primitives remains a valid solution, and the danger of finding an undesired solution (as is demonstrated in Figure 3.10) is small.

real image curves in a few seconds. Furthermore, a novel merging operation was defined, which accomplished curve simplification and helped in noise reduction and resolution change.

Direct proximity minimization

The conversion of residual distances into a dissimilarity measure is explained in section 3.3.6. The reader may wonder why we can't choose a proximity criterion, like the average distance between matched points or the Hausdorff distance, and minimize it directly. The Hausdorff distance may appear to be especially attractive, since it is not based on any prior feature pairing [3, 60].

We claim that direct minimization, which is heavily studied in the literature, is not adequate when the curves are only weakly similar. The reason is that proximity methods treat curves as two sets of points, and ignore more qualitative "structural" information.

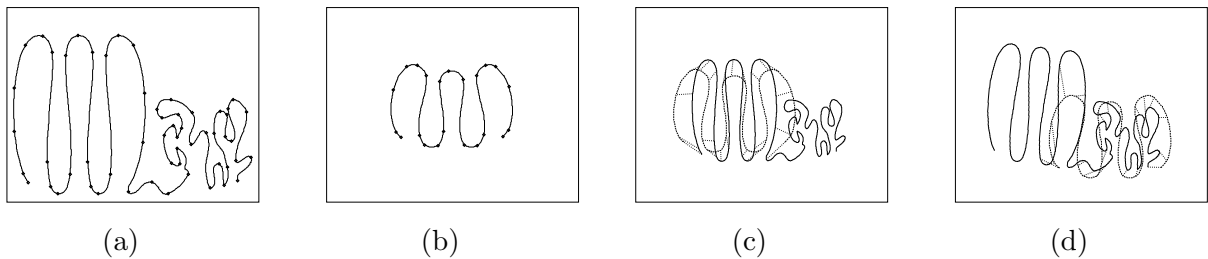


Figure 3.11: (a),(b) - Two weakly similar curves. The points are extracted automatically from polygonal approximations that do not depart from the original curves by more than 8 pixels. (c) The desirable matching result, as obtained by our matching algorithm. The average distance between matched feature points is 25 pixels. The directed Hausdorff distance is 34 pixels. (d) A non desirable pairing of points which yields better proximity value. The average distance between matched feature points is only 23 pixels. The directed Hausdorff distance is only 29 pixels.

An example is shown in Figure 3.11. In this example we compare two weakly similar curves, where a permissible alignment transformation includes translation, rotation and *uniform* scaling. Assuming that two sets of feature points had been extracted from the two curves, we investigate the proximity measure which is defined as the average distance between matched points. This measure is larger for the alignment shown in (c) than for the one shown in (d). Hence a proximity algorithm, which seeks the optimal alignment that achieves minimal residual distances, will consider (d) as a better alignment than (c)⁷.

⁷Note that the lower proximity value in (d) is not the result of the use of different global scaling, since the shorter curve appears in (c) and (d) at the same size. Moreover, the lower proximity measure of (d) is obtained even though many-to-one matches are avoided, and the order of points is kept. Without these constraints, it is easy to get arbitrarily small proximity values for an arbitrary correspondence by shrinking one set of points and matching it with only a few points (or even a single point) from the other set.

We continue with the previous example and investigate instead the use of the directed Hausdorff distance as the proximity criterion. The directed Hausdorff distance from a point set P to a point set Q is defined (with the Euclidean norm $\|\cdot\|$) as: $h(P, Q) = \max_{p \in P} \min_{q \in Q} \|p - q\|$; it is equal to the largest distance from some point in P to its nearest neighbor in Q . This measure is asymmetric, and therefore the symmetric expression $\max\{h(P, Q), h(Q, P)\}$ is often preferred. However, when there is a large image clutter, as in our case, the symmetric distance is not useful since its value is determined by the irrelevant part of the curve. Thus the directed distance, measured from the shorter curve to the longer one, is larger for the alignment shown in (c) than for the one shown in (d). A proximity algorithm that is based on minimizing the directed Hausdorff distance will consider (d) as a better alignment than (c).

In contrast, because it uses local structure, our syntactic matching algorithm provides the correct matching of the curves in Figure 3.11 (shown in part c).

Chapter 4

Results

In this chapter we apply our clustering algorithm to some fundamental computer vision problems, namely 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 sets examples used in Chapter 2 are no exception, but in this chapter we focus on real world computer vision problems, analyzing real images and contours extracted from images. In Sections 4.1 and 4.2 we consider segmentation of intensity and color images respectively. In Section 4.3 we discuss the case of “unstructured background”, which includes the problem of grouping edge elements. In Section 4.4 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.

4.1 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 (2.2) we have:

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

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 scale parameters. As in [109, 127], we determine the parameters a and b manually. To reduce the number of edges and get a sparse graph, we eliminate

edges whose weight is below some threshold and consider short range neighborhoods (see captions of Figures 4.1-4.4 for details).

The observation that the similarity graph is sparse is crucial for practical image segmentation applications. In [127] a sparse graph was obtained by randomly picking a small number of edges for each pixel in a limited range neighborhood. We use 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 the two methods¹. In any case, we exclude edges whose weight is below some threshold.

Figure 4.1 shows a gray level image taken during a baseball game. We use the same image used by Shi and Malik [127] to be able to compare the two methods. 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.

The eight images in the lower box of Figure 4.1 are shown to clarify the operation of the algorithm. These are intermediate results, and we use them to explain the origin of the peaks of $\Delta T(r)$, as well as to illustrate the operation of relabeling. Each column in the lower box is a pair of segmented images, computed at the transition level r and just before it, at the level $r - 1$. Hence we see that the origin of the peak at $r=3968$ is the splitting of the background cluster into two, one part containing the pixels of the lower player's body, while the other part containing the rest of the pixels. Similarly, the peak at $r=5615$ results from the detection of the upper player's body, and the peaks at $r=6366$ and 8872 result from splitting the background into different regions. Thus large changes in $\Delta T(r)$ reflect the segmentation of large objects in the scene.

The dark blue color in the eight intermediate images marks the pixels which are initially left unlabeled. In all our real image examples in this chapter, we regard clusters which contain less than 100 pixels as small clusters, and we do not label them. Namely, the parameter S of Section 2.7.3 is set to 100. The pixels which belong to these small clusters are assigned the dark blue color. The process of relabeling is intended to sustain the larger clusters, and relabel the smaller ones as discussed in Section 2.7.3. Thus the segmented images in the upper box of Figure 4.1 are obtained from the first row of images in the lower box, after the relabeling procedure has been applied. Note

¹The same image is used in [29] but in different resolution.

that this procedure can either connect unlabeled pixels to an existing colored grain, or it can create connected components among unlabeled pixels and assign to them a new tag. For example, the clusters constructing the two parts of the upper hand at level $r=6366$ are formed in this way.

We note that it is possible to get even finer level of resolution by setting the parameter S to a lower value. 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.

4.2 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, namely

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

but now $d(2)$ is measured in a $3D$ color space.

A color space is called “perceptually uniform” if a small perturbation to one of its components is equally perceptible across the space. A decade of psychophysical research at the Commission Internationale de L’Éclairage (CIE) during the 1960’s resulted in the definition of two color spaces, called CIE-LAB and CIE-LUV, which are approximately perceptually uniform. We define $d(2)$ to be the Euclidean distance in (either one of) these spaces. Our experiments do not suggest that one is better than the other, nor do we find consistent differences with respect to the measure used by [127] in HSV color space.

Figure 4.2 shows an outdoors scene with a rainbow and a cow. The arrangement of the images is the same as in Figure 4.1, with the impulse graph of $\Delta T(r)$ shown next to the image, and the three segmentation results which correspond to the three highest peaks in the impulse graph are 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 origin of the other peaks can be seen in the lower box, where the intermediate results at r and $r-1$ are shown². The analysis is in CIE-LAB color space.

²Three smaller peaks are seen at $r=1650$, 9330 and 10282. The segmented images at these values are not shown.

Figure 4.3 shows 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).

4.3 Unstructured background and grouping of edge elements

Our previous examples of image segmentation explicitly assume that the images can be partitioned into regions of coherent 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 they cannot be grouped together by standard clustering criteria. Nevertheless, 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 objects form many isolated clusters.

Image segmentation with unstructured background

To demonstrate this case we start with a synthetic example, where the task is brightness image segmentation. Here we follow the recent work of Perona and Freeman [109], who considered the segmentation of “structured” foreground from “unstructured” background. Figure 4.4a shows a synthetic image that is generated according to the parameters reported in [109]. Next to it, in the impulse graph of $\Delta T(r)$, we observe a large peak at $r=654$. The 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 some size threshold, leaving these pixels unlabeled.

The parameter S of minimal size of interest reflects prior information, and determines the resolution of the segmentation. Figures 4.5(a,b) show $\Delta T(r)$ obtained for larger values of this parameter. In this case the interesting clusters are large, and if this information is known in advance it can be used to clean up the impulse graphs. This will not be the case in our next example, of perceptual grouping of line segments. Figure 4.5(c,d) shows the partition obtained at

We note that the peak at $r=1650$ results from the splitting of the grass region, shown in yellow, into the yellow and dark red regions. The peaks at $r=9330$ and 10282 are due to the separation of the lower sky areas, later shown in light blue, red and light green. These peaks are relatively high since they indicate a separation from a large area (the sky).

the second highest peak.

The cross validation technique described in Section 2.6 can be applied here. The values which we get for the first five cross validation indices $\rho_1 \dots \rho_5$ are: 0.46, 0.02, 0.08, 0.02, 0.08. These values are obtained by averaging over 15 different random subsamplings, with standard deviations of 0.27, 0.05, 0.15, 0.03 and 0.21 respectively. Although the statistics is found to be noisy, its mean value clearly suggests that only the first partition can be accepted.

A comparison with the normalized cut algorithm [127] and with the factorization algorithm [109] is presented in Figure 4.6. The results are in agreement with those reported in [109], 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 4.7. Our heuristic, which identifies meaningful partition levels by measuring variations in $T(r)$, does not function as well under these circumstances, since it measures size variation. It is beyond the scope of this work to formulate an alternative principle. However, we can still use our method to generate a series of candidate partitions. The final solution should be selected from this set of candidate partitions, using an external criterion like good continuity, closure, or convexity.

Figure 4.7 shows a contour of a lemon superimposed on random background. The total number of line segments in this figure is 440, while the number of segments which construct the lemon contour is only 44 (signal to noise ratio of about 10%).³ The image and the similarity measure w_{ij} are adopted from [133]. 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

³In comparison, the signal to noise ratio in Figures 4.4 is about 100% (642 pixels in the central rectangles, 638 pixels in the random background).

segments. After the similarity between every two line segments is computed, we discard low weight edges; our elimination criterion uses the mutual-k principle [10], 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 than 10 elements, as we did in Figure 4.7, then there are only 7 candidate partitions to consider; and as noted above - the selection between them can rely on additional external perceptual criteria.

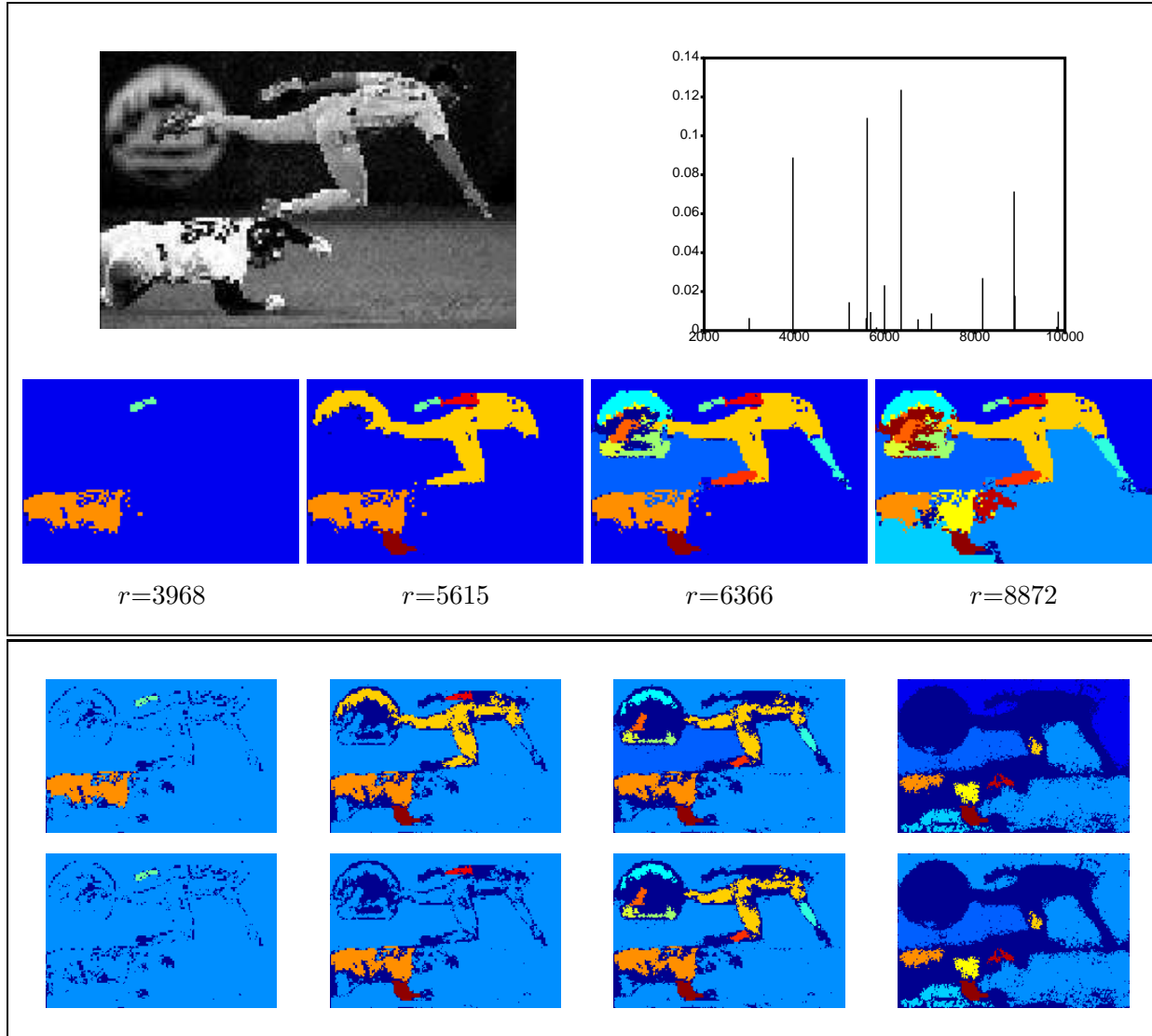


Figure 4.1: Brightness image segmentation. **Upper box:** the original image of size 147×221 , the impulse graph of $\Delta T(r)$ and the four segmentation results which correspond to the four highest peaks. **Lower box:** The intermediate results obtained before and after each one of the four highest transitions. The intermediate results are shown without relabeling, with dark blue color assigned to unlabeled pixels. **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 spacial 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. $M=1000$.

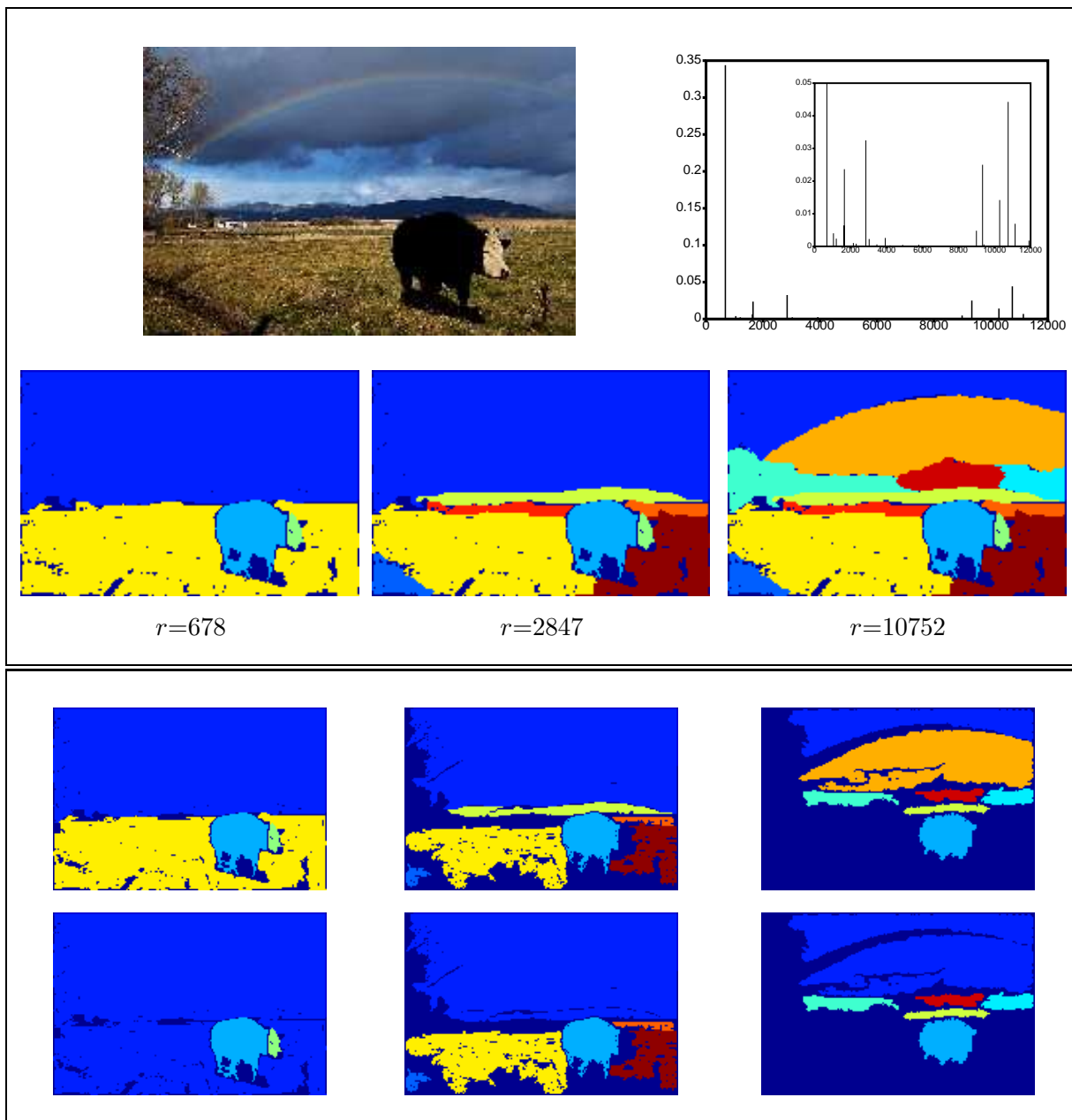


Figure 4.2: Color image segmentation in CIE-LAB color space. **Upper box:** the original image of size 122×183 , the impulse graph of $\Delta T(r)$ (insert shows an enlarged graph), and the three segmentation results which correspond to the three highest peaks. **Lower box:** The intermediate results obtained before and after each one of the three highest transitions. The intermediate results are shown without relabeling, with dark blue color assigned to unlabeled pixels. **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 spacial 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.

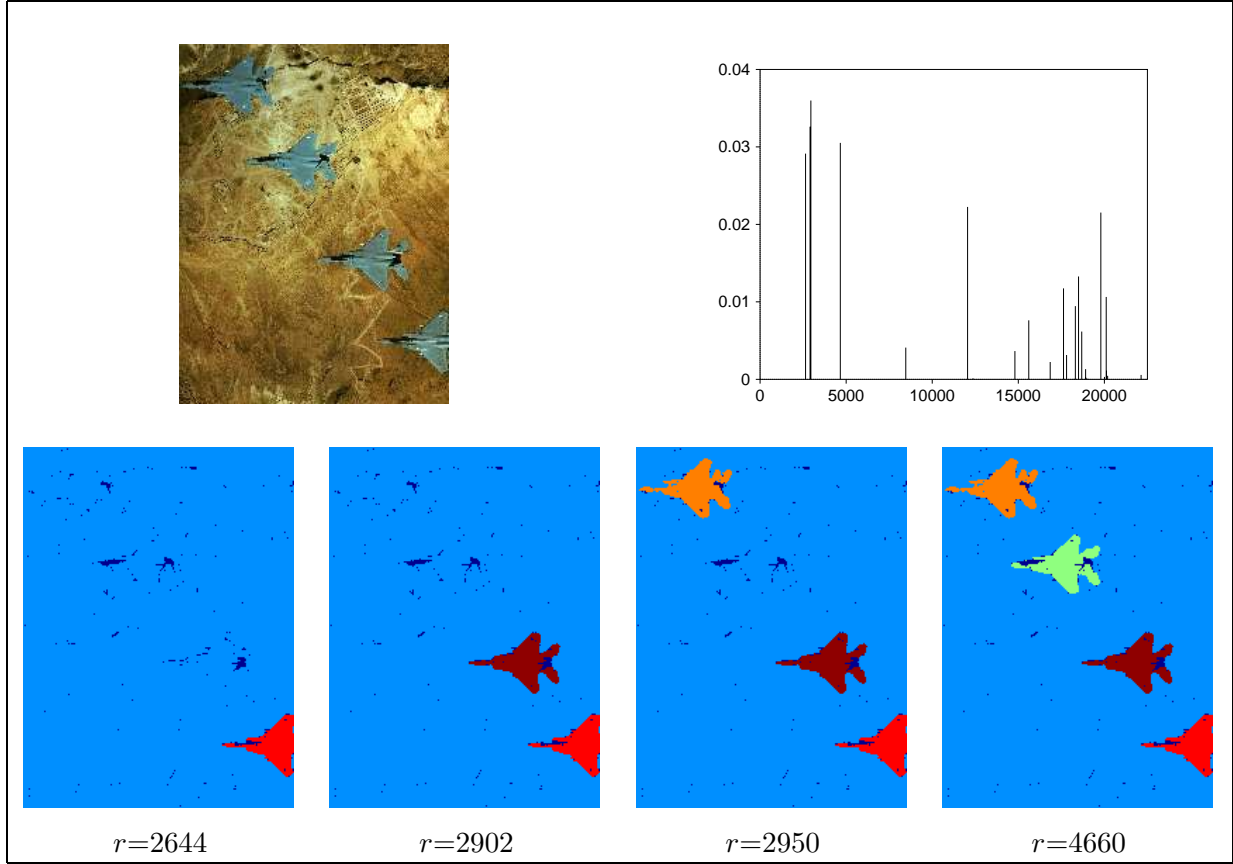


Figure 4.3: Color image segmentation in CIE-LUV color space. **Upper box:** the original image of size 256×192 , the impulse graph of $\Delta T(r)$ and the four segmentation results which correspond to the four highest peaks. **Parameter setting:** Colors are represented in CIE-LUV color space, $a=64$, $b=9$, edges with weight w_{ij} below 0.001 are eliminated, and only edges connecting each pixel with its four spacial 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. $M=500$.

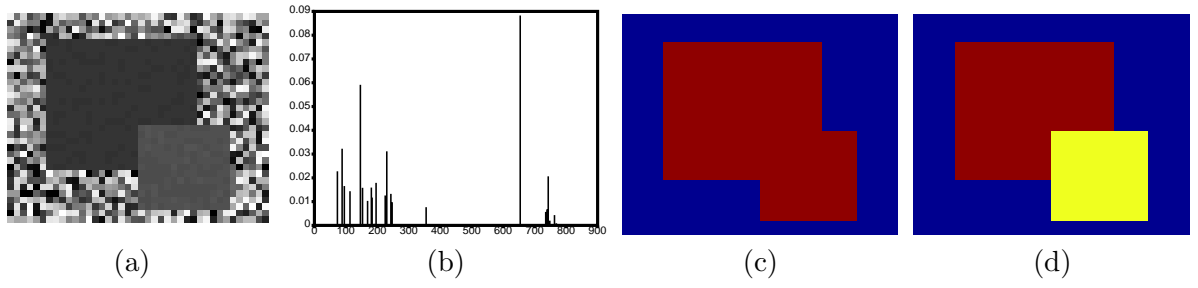


Figure 4.4: Separation of homogeneous objects from noisy background. (a) Synthetic 30×42 image, generated with the same parameters as in [109]: the brightness values are uniformly distributed between 0 and 1 for the background, between 0.2 and 0.21 for the larger rectangle, and between 0.3 and 0.31 for 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 [109]), only edges with the 8 nearest neighbors of each pixel are included.

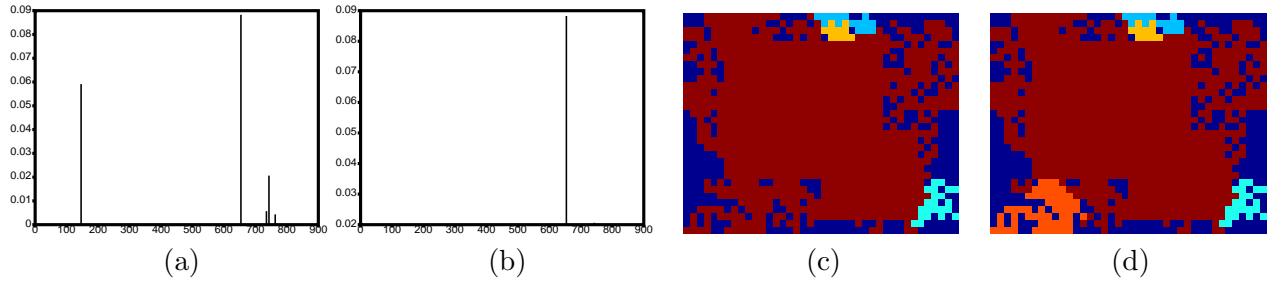


Figure 4.5: Separation of homogeneous objects from noisy background - II. (a,b) The graphs of $\Delta T(r)$ obtained when the minimal size of interest for a cluster is set to 30 and 100, respectively. (c,d) The segmentation results at $r=145, 146$ that correspond to the second largest peak of $\Delta T(r)$. The peaks at the lower r values correspond to spurious clusters which are formed in the background. For $r=146$, the peak results from the segmentation of the orange cluster at the lower left corner.

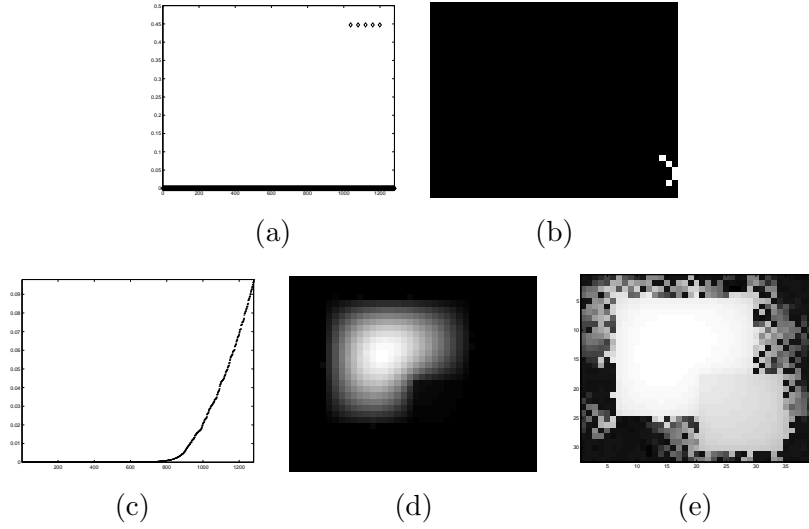


Figure 4.6: The performance of other algorithms applied to the data of Figure 4.4. The same preprocessing and exponential transformation from distances to weights was used (see text). **First row:** The normalized cut algorithm [127]. (a) One of the best Ncut partitions (there are many partitions which correspond to 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. **Second row:** The factorization method [109]. (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 intensity scale, where the entries are ordered like the image pixels. (e) A log function is applied to the values of the eigenvector, and the presentation is like in (d).

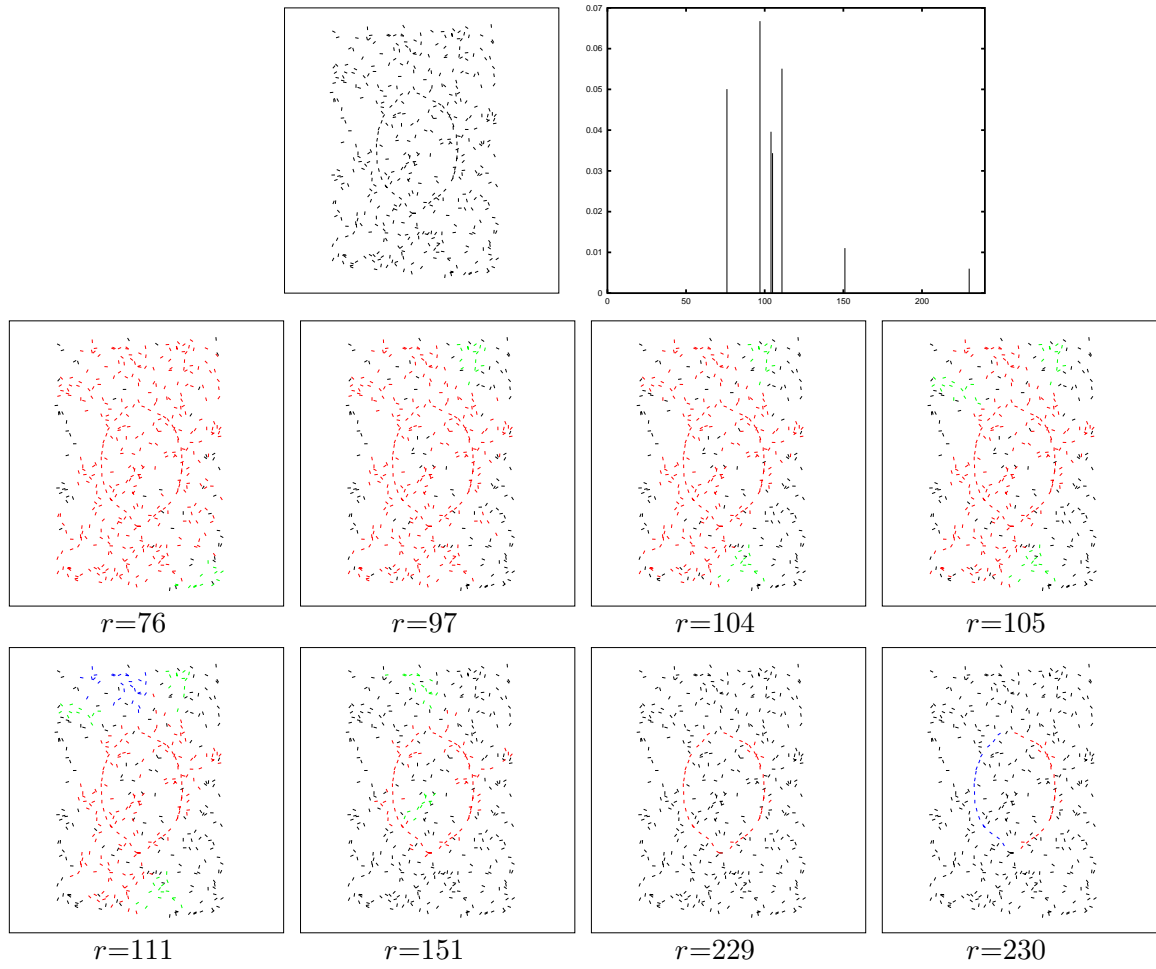


Figure 4.7: Perceptual grouping of line segments. The data is shown in the upper left image. If the minimal size of interest for a cluster is set to 10, then 7 transitions are detected, which are marked by the 7 peaks in the impulse graph of $\Delta T(r)$ (upper row, on the right). Each peak signals the segmentation of a set of at least 10 line segments (bottom rows). 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 in the bottom row, on the right ($r=229, 230$).

4.4 Organization of an image database

We now integrate the two steps of similarity estimation and pairwise clustering into one experiment of image database categorization. The database contains 121 images of 12 different objects; 90 of the images were collected by placing 6 toy models on a turntable, so that the objects could be viewed from different viewpoints. The other images are the 31 silhouettes discussed in Section 3.2.3. For each of the 6 toy models we collected 15 images, by rotating them in azimuth ($\vartheta = -20^\circ, -10^\circ, 0^\circ, 10^\circ, 20^\circ$) and elevation ($\varphi = -10^\circ, 0^\circ, 10^\circ$). We used models of a cow, wolf, hippopotamus, two different cars and a child.

The central images ($\vartheta = \varphi = 0$) in each of the three groups of pictures of animal models are side views (i.e., four legs, head and tail are visible). All the different 15 images of each animal model are somewhat similar in that the same parts are visible (though in some pictures some parts, such as 2 legs or a leg and a tail, are merged into one in the silhouette). Thus, 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 desirable 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 (see Section 3.3.1). The dissimilarities between the silhouettes are estimated using the algorithm described in Chapter 3. 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 4.8(a). Part of the data appears also in Table 3.1.

According to the general approach adopted in this work, 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 (2.2) we define

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

where a is a local scale parameter, here chosen to be the averaged d_{ij} to the second nearest neighbor⁴.

⁴Determining the local scale a by the second nearest neighbor results in a very fast decay rate of the weights w_{ij} . In this example we cannot determine a by remote neighbors, since some of the clusters of interest contains as few as four members. This is an unusual case in clustering applications. The larger clusters can be found even if a larger a is used. See our earlier report in [35], where the SPC algorithm [10] was used for the clustering.

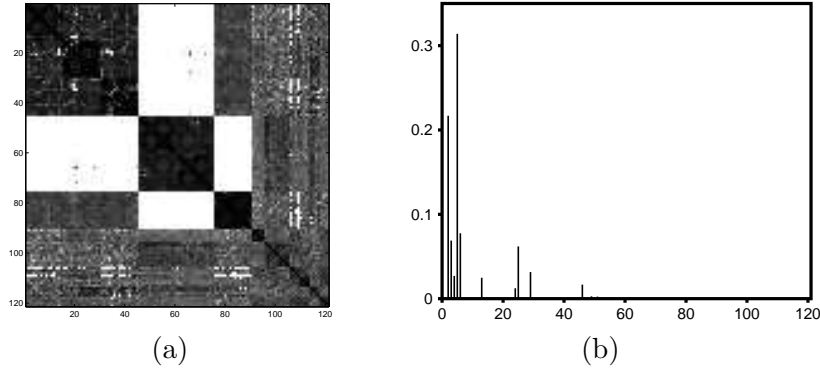


Figure 4.8: (a) The dissimilarity matrix d_{ij} computed for 121 images as described in Chapter 3, by silhouettes extraction, syntactic matching, and computation of residual distances between matched points. Larger dissimilarity values are represented by brighter intensities. The infinity value is represented 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 (Chapter 2): The impulse graph of $\Delta T(r)$ versus r , computed with 1000 iterations. The peaks correspond to the partitions shown in Figure 4.9. We have considered peaks above 0.01 to be meaningful.

After transforming d_{ij} to w_{ij} we threshold the graph ($\theta = 10^{-5}$) to obtain a sparse representation.

Figure 4.8(b) shows the graph of $\Delta T(r)$, while the corresponding hierarchical classification (dendrogram) is shown in Figure 4.9. 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, which means that our dissimilarity measure is continuous, assigning low (but reliable) values to weakly similar shapes.

Since humans can do so, we assume that an ideal shape classifier can put the images of every object in a different class. It is hard to test this hypothesis, since as humans we cannot ignore the semantic meaning of the shapes. Nevertheless, 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.

Our categorization is obtained using only intrinsic shape information. The relative size, orientation and position of the silhouettes within each category is arbitrary. Moreover, global information like the length of the occluding contour or the area it encloses are not used. Hence we expect that moderate occlusion will not affect the classification.

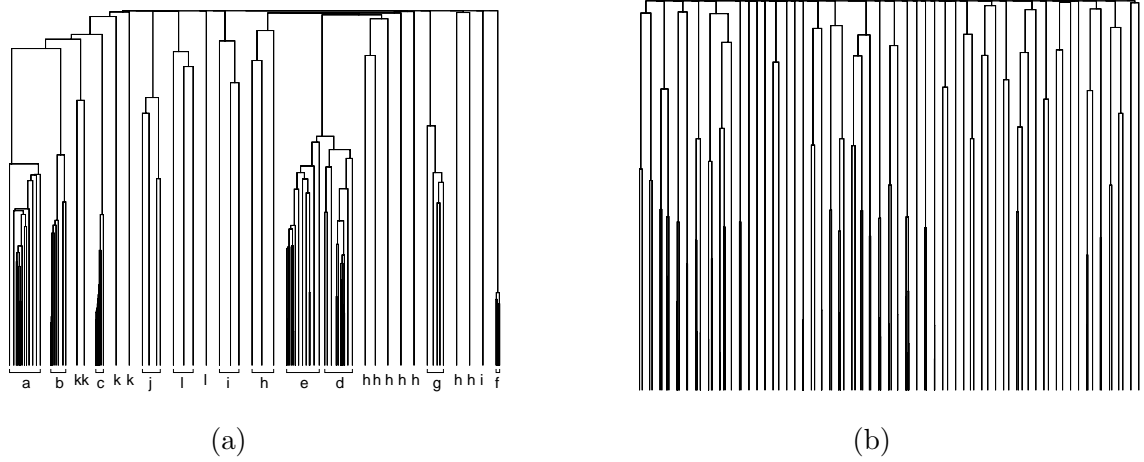


Figure 4.10: The application of two agglomerative techniques to the silhouettes data: (a) single linkage, (b) complete linkage. The horizontal spacing between endpoints (leaves) is proportional to the scale (vertical axis) in which they are merged together. Hence the lines within cluster are dense, while more space is left between clusters. While the complete linkage algorithm completely fails, the single linkage algorithm succeeds to find meaningful structure, although the natural levels are not salient. We conclude that the images form chained structures, and not compact (isotropic) ones. The labels near the leaves are: a-cow, b-hippo, c-wolf, d-car, e-sport car, f-child, g-hand palm, h-fish, i-plane, j-rabbit, k-tool, l-artificial shape.

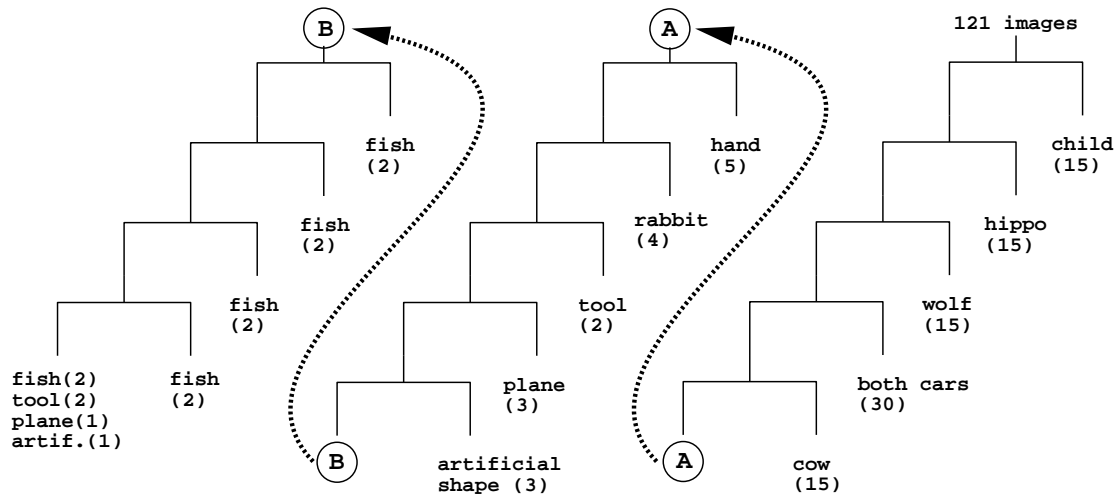


Figure 4.11: The factorization algorithm [109] recursively applied to our silhouettes data. The method does not offer hierarchical interpretation. The threshold at each branching point was selected manually. No threshold could be found to separate the 30 images of the two different cars. The divisive process at the later stages cuts spurious small clusters.

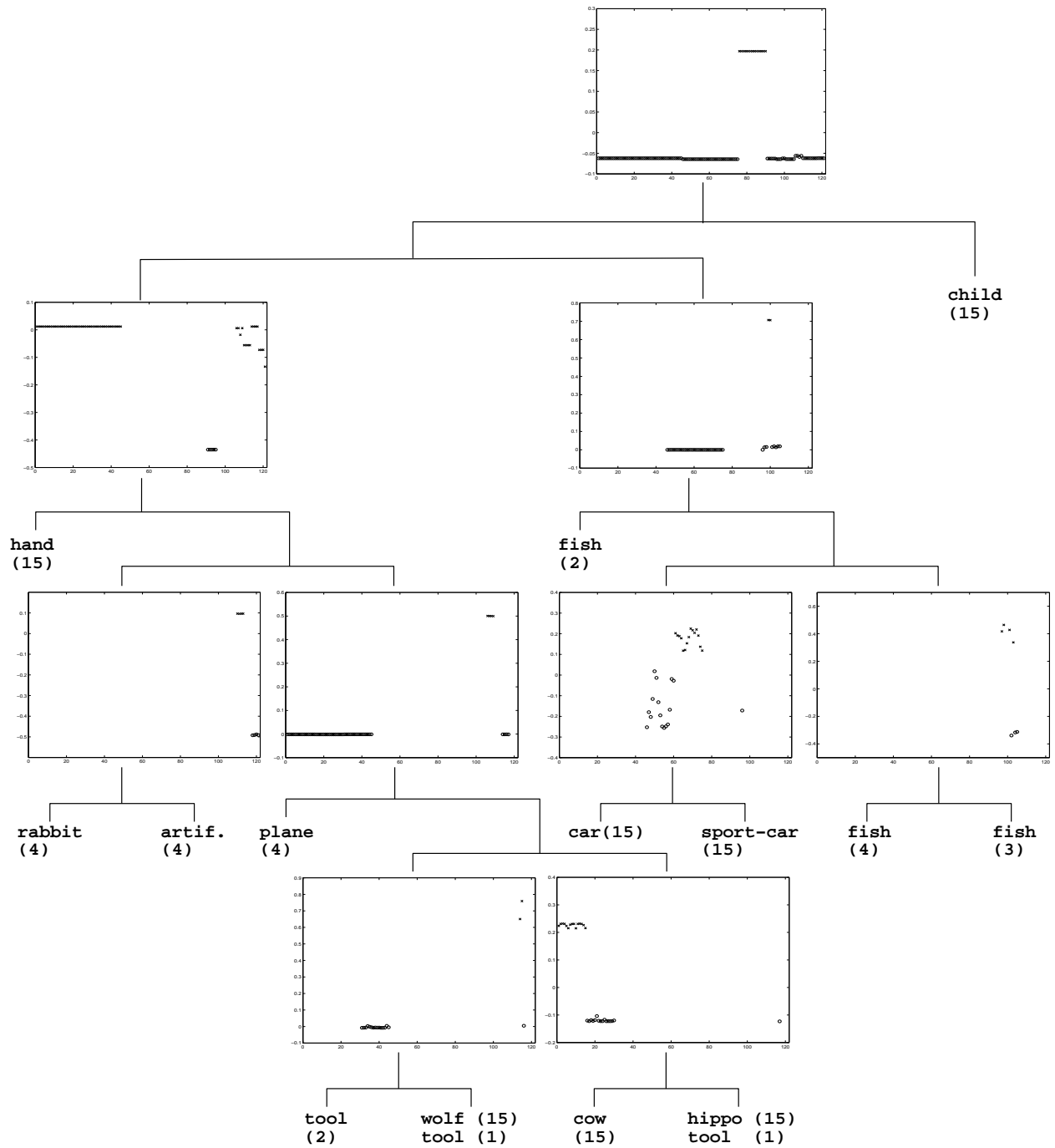


Figure 4.12: The normalize cut algorithm [127] recursively applied to our silhouettes data. In tree nodes which split into at least one terminating leaf we show the entries of the corresponding eigenvector. The branching is halted when, by inspection, the entries of the eigenvector do not show a clear separation into two groups. The images are ordered in the same order as in Figure 4.8a, namely images that belong to the same class are adjacent on the x-axis.

Comparisons

Our image classification procedure consists of two parts: estimation of pairwise dissimilarities and detection of clusters. The quality of the dissimilarity estimation was considered in Chapter 3. Here we claim that the veridical hierarchical clustering is an indirect evidence for the quality of our dissimilarity estimation. The reliable pairwise relations within classes and between related classes give rise to veridical partitions with a number of clustering methods.

In all our comparison experiments we use as input the same matrix w_{ij} , to eliminate any dependence on the data preprocessing. We apply agglomerative algorithms (Figure 4.10) and two spectral methods (Figures 4.12, 4.11). 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 (see also Table 3.1), 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 (Figures 4.12) is comparable with ours. While our algorithm computes the whole hierarchy at once, the normalized cut algorithm is applied recursively on each part. The stopping condition of this algorithm is not well formalized; the deviation 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 did not observe a clear partition by eye. In Figures 4.12 we show the eigenvectors that correspond to the final partition levels.

Next, we recursively apply the factorization method on our data. The partition tree is shown in Figures 4.11. Note that the 30 images of the two cars cannot be separated into two clusters by this method, since the principal eigenvector of the corresponding 30×30 matrix does not indicate a separation. This method does not suggest hierarchical interpretation of the data, and it breaks the fish cluster into small pieces, each containing just two members.

Chapter 5

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 unified approach. In particular, we have focused on the tasks of image segmentation, edge elements grouping, and image organization.

The first step in these tasks is to define the pairwise similarity between the visual entities: pixels, edgels, or shapes. While natural measures exist for pixel similarity, and extensive work is done in the area of edgels similarity, shape similarity is not well understood. Most of the algorithms in the literature which address the problem of shape similarity are designed for model based recognition tasks. Hence they are usually suitable for decision problems, for example when one needs to decide which model shape in the archive is identical to a given query shape.

Hierarchical image organization necessitates a graded similarity measure, and when the desired categories are shape categories, then a graded shape similarity measure is necessary. In addition, an appropriate similarity measure should be invariant with respect to scaling and rigid transformations of the image, and should be local (avoid using global invariants) in order to be useful in cases of occlusion. We have shown how a flexible curve matching algorithm can be used to define the dissimilarity between weakly similar shapes. Our flexible syntactic matching is based on a simple heuristic, guided by the principle that matched features should lie on locally similar pieces of curve. Naturally, this principle cannot guarantee that the results would always agree with our human intuition for “good” matching, but our examples in Section 3.2.2 demonstrate that satisfactory and intuitive results are usually obtained. We demonstrated excellent results, matching similar curves under partial occlusion, matching similar curves where the curves depict the occluding contours

of objects observed from different viewpoints, and matching different but related curves (like the silhouettes of different mammals or cars).

In order to achieve the large flexibility we introduced a non linear measure of similarity between line segments, which is not sensitive to either very small or very large differences in their scale and orientation. Our specific choice of segment similarity, combined with a novel merging mechanism and an improved interruption operation, added up to a robust and successful algorithm. The most important properties of our algorithm, which make it advantageous over other successful matching algorithms, are its relatively low complexity, its locality which allows us to deal with occlusions, and its invariance to $2D$ image transformations. Note that “successful” matching depends on the application at hand. Our method is not suitable for recovering depth from stereo, but it is well suited for more qualitative tasks, such as the organization of an image database, the selection of prototypical shapes, and image morphing for graphics or animation.

Getting back to the general scheme that underlies our work, the second step in our unified approach 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. We use a canonical weight transformation, which converts the perceptual measure of distance to exponentially decaying similarity weight¹. Exponential decay is not only supported psychophysically [125, 126, 28], it can be understood from the properties of the cut cost function that we use. This cost, 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, avoids this undesirable bias.

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 at 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.

¹The method that we chose to compute the similarities between edge elements directly provides us with a similarity (and not a dissimilarity) measure. It is inherently an exponentially decaying function.

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 2.5.3). 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 2.5.2 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 rely on the interpretation of $1-p_{ij}^r$ as the probability that edge (i, j) is a crossing edge in an “average cut” (Section 2.2). The N typical cuts corresponding to $r = 1 \dots N$ 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.

Since the estimation of the pairing probabilities does not require the computation of a particular minimum cut, the resulting algorithm turned out to be more efficient than explicit minimal cut computation. A deterministic state of the art algorithm which finds minimal 2-way cut in an N -nodes $|E|$ -edges graph is proposed in [99], and requires $O(N|E|)$ time. The state of the art probabilistic approach is based on packing of maximum spanning trees and random sampling [71], which takes $O(|E|\log^3 N)$ time (for bi-partition). To divide the graph by a cascade of bi-partitions it is better to compute maximal flows rather than to apply these algorithms recursively. Based on the Gomory-Hu theorem, the state of the art algorithms require $\Omega(N^2|E|)$ time [76]. On the other hand, our algorithm runs in $O(N \log^2 N)$ time on sparse graphs that have $|E| = O(N)$ edges. Complete graphs are not of interest in usual clustering application, but in this case the running time of our algorithm is $O(N^2 \log N)$. In both cases the space complexity is $O(|E|)$. It must be noted that 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.

In Chapter 4 we investigated experimentally the plausibility of our goal, and showed that using our clustering algorithm we can address different problems in computer vision. Some of our results were compared with other method, primarily with the recently proposed normalized cut algorithm [127] and the factorization algorithm [109]. This adds to some comparisons that are presented in Chapter 2. To be concrete, we compared the performance on synthetic point set examples in Figures 2.6 and 2.9; we presented in Figure 4.1 our segmentation results using the same image reported in [127]; we compared our results under unstructured background noise with those reported in [109] (Figure 4.6); and we analyzed our database of 121 images using other methods in Figures 4.10-4.11.

We conclude that there are cases where our method works and the normalized cut and the factorization method fail completely (Figures 2.6,2.9). In other cases, like in the image segmentation of Figure 4.1, 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. It must be noted that in addition the complexity of our algorithm appears to be lower: while our running time is $O(N \log^2 N)$, the complexity of normalized cut 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 than $O(\sqrt{N})$.

The factorization algorithm was proposed in [109] in conjunction with the figure-ground problem that aims to segment a structured “foreground” set of objects from an unstructured “background” set. The claim was that while the normalized cut algorithm fails in these circumstances, the factorization algorithm performs well. In Figure 4.6 we reconstructed the example used in [109], and showed that our algorithm works better than the factorization method, providing that we interpret small clusters as spurious structures and ignore them. When the unstructured background contains most of the data points, it might be that our current algorithm will only generate a few candidate solutions, and the selection between them will have to rely on additional external perceptual criteria (see the example of grouping edge elements in Section 4.3).

The comparisons between different methods using the image database data (Figures 4.10-4.11) showed that for this example the normalized cut and our algorithm produced comparable results. Both methods detected all the relevant partitions. This kind of database structuring could not have emerged without the reliable estimation of the dissimilarities between weakly similar images.

Hence the veridical hierarchical organization stands as an indirect and objective evidence to the quality of our matching and similarity estimation.

The factorization algorithm also succeeded to detect many of the clusters, but its overall performance was not as good as the previous algorithms, and it did not provide a hierarchical interpretation of the data. The single linkage methods performed less well, and the complete linkage algorithm failed completely due to the chained structure of the clusters.

Appendix A

An Alternative Implementation of The Typical Cut Algorithm

The efficient implementation of the first stage of our clustering algorithm is presented in Section 2.7.2, and is based on UNION and FIND operations. Using a mapping from node to meta nodes names (*names*), the inverse mapping *members* and the array *sizes*, the FIND operation was implemented in constant time and the UNION operation was implemented in $O(\log N)$ average time.

In the following proposed implementation we use alternative mappings between names, and implement the UNION operation in constant time, and the FIND operation in nearly constant time on average. Moreover, we maintain a data structure which supports direct indexing into the edges which connect two given meta nodes. We therefore believe that the implementation presented here is more efficient, although we cannot show a complexity bound which reflects this intuition.

The mapping *names* is implemented as a forest data structure. Hence the members of each meta node construct a tree, and if node i is the parent of node j then $names(j) = i$. The root index is the meta node name. A UNION operation is to make the root of the smaller tree be a child of the root of the larger, and this takes $O(1)$ time. A FIND operation for node i is to follow the path from $names(i)$ up to its root, and to return the root index. In addition, during FIND, we change each node encountered along the path to be a child of the root. This is known as path compression. It can be shown that in this implementation all the FIND operations which are involved with a single graph contraction (single iteration of the loop at lines 7–17, Figure 2.14) takes $O(N\alpha(N))$ time, where α is the inverse Ackermann's function. This function is not a constant, yet $\alpha(N) \leq 4$ for all integers N one is ever likely to encounter.

Our challenge is to construct a new implementation of **CONNECTING-EDGES**, which does not use the *members* mapping (this mapping cannot be maintained by a constant time union operation). We define a new data structure, called *neighbors*. For every meta node u , $neighbors(u)$ is a list of items of the form $\{v, E_{uv}\}$, where v is a meta node which is adjacent to u and E_{uv} is list of edge indices connecting u and v . There is some arbitrariness in this structure, since the adjacent meta nodes u and v may be listed either under $neighbors(u)$ or under $neighbors(v)$.

In principle, a **UNION** operation should be followed by a complicated update of the *neighbors* lists. We settle for a partial update. Namely, after the meta node v was renamed u , we concatenate the lists $neighbors(u)$ and $neighbors(v)$, but we do not search for members of the form $\{v, \dots\}$ in other lists. These members stay with a name v which is no longer valid, hence every access to a meta node name in the data structure *neighbors* will be followed by a **FIND** operation, that will correct the name if necessary.

The new **CONNECTING-EDGES** procedure is shown in Figure A.1. Lines 1–5 are identical with the previous version. Line 6 is the **UNION** operation. The loop in lines 8–15 query all the meta nodes which are either connected to u or to v . A possible name correction is performed in line 9, then the adjacent meta nodes are entered into a temporary list A (line 11). However, if u and v are contracted, it means that they were previously connected, meaning that in the list $neighbors(u)$ we may find a member called v and vice versa. These members should not be included in A . Instead, their lists of connecting edges are added to L (line 13).

Finally, line 16 calls for the procedure **PURIFY**. While constructing the list A it might happen that some of the members added to it have the same name. The procedure **PURIFY** sorts the list A by name, and replaces such members with a single member, whose edge list is the concatenation of the distributed lists.

In order to keep the length of the neighbors lists as short as possible, we use the following heuristic (which is not shown in Figure A.1). When a member $f = \{x, E_{ux}\} \in neighbors(u)$ is to be added to A , we first check whether the length of $neighbors(x)$ is shorter than the current length of A . In this case, instead of adding f to A , we rename it to be called u , and add it to $neighbors(x)$.

The reason why we could not find a complexity bound for this version of **CONNECTING-EDGES** is that we do not have a bound on the lengths of the neighbors lists. However, our experiments show that for sparse graphs these lists remain very short, of the order of the maximal node degree in the

```

procedure CONNECTING-EDGES:
input:    two meta nodes names  $u, v$ 
output:  list  $L$  of connecting edges (and modified mappings)
use mappings:   $names, neighbors$  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)  $names(v) \leftarrow u$  (the UNION operation)

(07) let  $A$  be a temporary empty list of neighbors
(08) for each member  $f = \{x, E_x\}$  in  $neighbors(u)$  and in  $neighbors(v)$ 
(09)      $x \leftarrow \text{FIND}(x)$ 
(10)     if  $x \neq u, v$  then
(11)         add the (corrected) member  $f$  to  $A$ 
(12)     else
(13)         add the edge-list  $E_x$  of  $f$  to  $L$ 
(14)     end-if
(15) end-loop
(16) PURIFY( $A$ )
(17)  $neighbors(u) \leftarrow A$ 
(18)  $neighbors(v) \leftarrow \text{NULL}$ 
(19) return  $L$ 

```

Figure A.1: An alternative implementation for merging meta nodes and finding the edges which connect them.

original graph G .

Appendix B

Syntactic Operations for Curve Matching: Comparison

The syntactic operations which underly our curve matching algorithm are defined in Section 3.3.3. Here we make a comparison with similar definitions from the literature.

How to measure similarity

Local and scale invariant matching methods usually use the normalized length ℓ/ℓ_0 . For example, the ratio between normalized lengths $\frac{\ell/\ell_0}{\ell'/\ell'_0}$ is used in [89, 86] (with global normalization the difference $|\ell/L - \ell'/L'|$ can be used [139, 135]). The ratio between normalized lengths may be viewed as the ratio between the relative scale $c = \ell/\ell'$ and the reference relative scale $c_0 = \ell_0/\ell'_0$. While this scale ratio is invariant, unlike our measure it is not bounded and is thus less stable.

We are familiar with only one other definition of a symmetric, bounded and scale invariant measure for segment length similarity [86]. However, their matching algorithm is not syntactic and is very different from ours. In addition, there is an important qualitative difference between the two definitions (see Figure B.1), where our measure is more suitable for flexible matching.

As for our measure of orientation difference, we note that a *linear* measure of orientation differences has been widely used by others [135, 136, 89, 16]. The non linear measure used by [86] differs from ours in exactly the same way as discussed above concerning length.

Reminiscent of our combined similarity measure (3.3), in [111] a coupled measure is used: the segments are superimposed at one end, and their dissimilarity is proportional to the distance between their other ends. However, this measure is too complicated for our case, and it has the

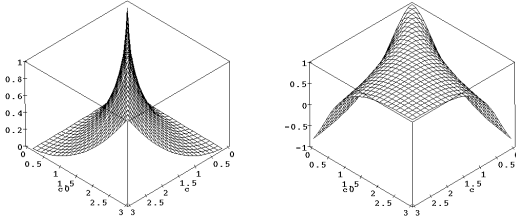


Figure B.1: Scale similarity: the scale $c = \ell/\ell'$ is compared with the reference scale $c_0 = \ell_0/\ell'_0$. Left: The binary relation used by Li [86] to measure scale similarity is $\exp(-|\log(c/c_0)|/\sigma)$ with $\sigma = 0.5$. Right: Our measure function (Equation (3.1)) is not sensitive to small scale changes, since it is flat near the line $c = c_0$.

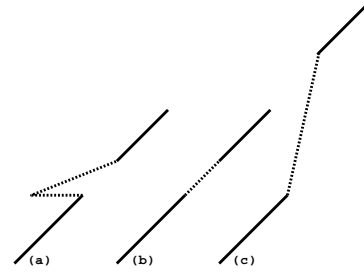


Figure B.2: Comparison between merging rules: (a) A polygonal approximation of a curve, with two dotted segments which are to be merged. (b) Merging result according to our scheme. A coarser approximation is obtained. (c) Merging according to Tsai and Yu [135]. The new polygon does not appear to give a good approximation.

additional drawback that it is sensitive to the arbitrary reference scale and orientation (in the character recognition task of [111] it is assumed that characters are of the same scale and properly aligned).

How to open gaps

All the syntactical shape matching algorithms that we are familiar with make use of deletions and insertions as purely local operations, as in classical string matching. That is, the cost of inserting a sequence of gaps into a contour is equal to the cost of spreading the same number of gap elements in different places along the contour. We distinguish the two cases, since the first typically arises from occlusion or partial matching, while the second arises typically from curve dissimilarity. In order to make the distinction we adopt a technique frequently used in protein sequence comparison, namely, we assign a cost to any event of contour interruption, in addition to the (negative) cost from deletion/insertion of any single element.

How to merge segments

A similar approach to segment merging (Section 3.3.3) was taken in [139], but their use of invariant attributes made it impossible to realize the merge operator as an operation on attributes. Specifically, there is no analytical relation between the attributes being merged to the attributes of the new primitive. Instead, a cascade of alternative representations was used, each one obtained by a different Gaussian smoothing of the two dimensional curve; a primitive sequence is replaced by its

“ancestor” in the scale space description¹.

Compare the polygonal approximation after merging with the polygon that would have been obtained had the curve been first smoothed and then approximated. The two polygons are not identical, since smoothing may cause displacement of features (vertices). However, a displaced vertex cannot be too far from some feature at the finest scale; the location error caused by “freezing” the feature points is clearly bounded by the length of the longest fragment in the initial (finest scale) representation. To ensure good multi scaled feature matching our sub-optimal polygonal approximation is sufficient, and the expensive generation of the multi scale cascade is not necessary. Instead, the attributes of the coarse scale representation may be computed directly from the attributes of the finer scale.

Merging was defined as an operation on attributes by [135], who also applied the technique to Chinese character recognition [136]. Their algorithm suffers from some drawbacks concerning invariance and locality²; below we concentrate on their merging mechanism, and compare it to our own.

Assume that two line segments characterized by (ℓ_1, θ_1) and (ℓ_2, θ_2) are to be merged into one segment (ℓ, θ) . In [135] $\ell = \ell_1 + \ell_2$, and θ is the weighted average between θ_1 and θ_2 , with weights $\ell_1/(\ell_1 + \ell_2)$ and $\ell_2/(\ell_1 + \ell_2)$, and with the necessary cyclic correction.³ Usually, the polygonal shape that is obtained using this simple ad-hoc merging scheme *cannot* approximate the smoothed contour very well. Satisfactory noise reduction is only achieved in one of the following two extreme cases: either one segment is dominant (much longer than the other one), or the two segments have similar orientation. If two or more segments having large variance are merged, the resulting curve may be very different from the original curve (see Figure B.2). The result of the segment merging in the finer scale is not an acceptable coarser approximation of the shape.

¹The primitive elements used in [139] are convex and concave fragments, which are bounded by inflection points. The attributes are the fragment length divided by total curve length (a non-local attribute), and the accumulated tangent angle along the fragment (a non-interruptible attribute). The algorithm cannot handle occlusions or partial distortions, and massive preprocessing is required to prepare the cascade of syntactical representations for each curve, with consistent fragment hierarchy.

²The primitives used by [135] are line segments, the attributes are relative length (with respect to the total length) and absolute orientation (with respect to the first segment). The relative length is, of course, a non-local attribute, and in addition the algorithm uses the total number of segments, meaning that the method cannot handle occlusions. The problem of attribute variance due to rotation remains in fact unsolved. The authors assume that the identity of the first segments is known. They comment that if this information is missing, one may try to hypothesize an initial match by labeling the segment that is near the most salient feature as segment number one.

³For example, an equal weight average between 0.9π (almost “west”) and -0.9π (almost “west” as well) is π (“west”) and not zero (“east”).

List of Publications

Refereed conferences

- Y. Gdalyahu and D. Weinshall: “Measures for Silhouettes Resemblance and The Most Representative Silhouette of a Curved Object”, in *Proc. of the fourth European Conference on Computer Vision (ECCV96)*, vol II, 363–375, Cambridge, April 1996.
- E. Shilat, M. Werman, and Y. Gdalyahu: “Ridge’s Corner Detection and Correspondence”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR97)*, 976–981, Porto-Rico, June 1997.
- D. W. Jacobs, D. Weinshall and Y. Gdalyahu: “Condensing Image Data-bases when Retrieval is based on Non-Metric Distance”, in *Proc. of the sixth International Conference on Computer Vision (ICCV98)*, 596–601, Bombay, January 1998.
- Y. Gdalyahu and D. Weinshall: “Flexible Syntactic Matching of Curves”, in *Proc. of the fifth European Conference on Computer Vision (ECCV98)*, vol II, 123–139, Freiburg, June 1998.
- Y. Gdalyahu and D. Weinshall: “Automatic Hierarchical Classification of Silhouettes of 3D Objects”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR98)*, 787–793, Santa Barbara, June 1998.
- Y. Gdalyahu, D. Weinshall and M. Werman: “A Randomized Algorithm for Pairwise Clustering”, in *Advances in Neural Information Processing Systems (NIPS98)*, 11, 424–430, Denver, December 1998.
- D. Weinshall, D. W. Jacobs and Y. Gdalyahu: “Classification in Non-Metric Spaces”, in *Advances in Neural Information Processing Systems (NIPS98)*, 11, 838–844, Denver, December 1998.
- Y. Gdalyahu, D. Weinshall and M. Werman: “Stochastic Image Segmentation by Typical Cuts”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR99)*, vol II, 596–601, Fort Collins, June 1999.

Journal submission

- Y. Gdalyahu and D. Weinshall: “Flexible Syntactic Matching of Curves and its Application to Automatic Hierarchical Classification of Silhouettes”, submitted to *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*.

- Dubnov S., El-Yaniv R., Gdalyahu Y., Schneiderman E., Tishby N. and Yona G., “A new non parametric clustering algorithm”, submitted to *Machine Learning Journal* special issue on unsupervised learning.
- Y. Gdalyahu, D. Weinshall and M. Werman: “Self organization in vision: stochastic clustering for image segmentation, perceptual grouping, and image database organization”, to be submitted to *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)* special issue on graph algorithms in computer vision..

Other publications

- D. Weinshall, M. Werman, Y. Gdalyahu: “Canonical Views, or the Stability and Likelihood of Images of 3D Objects”, in *Proc. of DARPA Image Understanding Workshop*, November 1994.
- Y. Gdalyahu and D. Weinshall: “Local Curve Matching for Object Recognition without Prior Knowledge”, in *Proc. of DARPA Image Understanding Workshop*, New-Orleans, May 1997.
- Y. Gdalyahu, D. Wenshall and M. Werman: “Stochastic Clustering and its Applications to Image Segmentation”, *IEEE Workshop on Graph Algorithms and Computer Vision (in conjunction with the International Conference on Computer Vision)*, Corfu, Greece, September 1999.

Bibliography

- [1] Alter T. and Basri R., “Extracting salient contours from images: An analysis of the saliency network”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 13-20, 1996.
- [2] Ansari N., Delp E., “Partial shape recognition: a landmark based approach”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 12, 470-489, 1990.
- [3] Arkin E., Paul Chew L., Huttenlocher D., Kedem K. and Mitchel J., “An efficiently computable metric for comparing polygonal shapes”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 13, 209-216, 1991.
- [4] Asada H. and Brady M., “The curvature primal sketch”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 8, 2-14, 1986.
- [5] Ayach N. and Faugeras O., “HYPER: a new approach for the recognition and positioning of two dimensional objects”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 8, 44-54, 1986.
- [6] Basri R., Costa L., Geiger D. and Jacobs D., “Determining the similarity of deformable shapes” *IEEE Workshop on Physics Based Modeling in Computer Vision*, 135-143, 1995.
- [7] Belongie S., Carson S., Greenspan H. and Malik J., “Color and texture-based image segmentation using EM and its application to content based image retrieval”, in *Proc. of International Conference on Computer Vision*, 675-682, 1998.
- [8] Bhanu B. and Faugeras O., “Shape matching of two dimensional objects”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 6, 137-155, 1984.
- [9] Blake A. and Zisserman A., “*Visual Recognition*”, The MIT press, 1987.
- [10] Blatt M., Wiseman S. and Domany E., “Data clustering using a model granular magnet”, *Neural Computation* 9, 1805-1842, 1997.
- [11] Bolles R. and Cain R., “Recognizing and locating partially visible objects: the focus feature method”, *International Journal of Robotics Research* 1, 57-81, 1982.
- [12] Boppana R., “Eigenvalues and graph bisection: An average case analysis”, in *Proc. IEEE Symposium on Foundations of Computer Science*, 280-285, 1987.

- [13] Boykov Y., Veksler O., and Zabih R., “Fast approximate energy minimization via graph cuts”, in *Proc. of International Conference on Computer Vision* vol I, 377-384, 1999.
- [14] Brint A. and Brady M., “Stereo matching of curves”, *Image and vision computing* 8, 50-56, 1990.
- [15] Cho K. and Meer P., “Image segmentation from consensus information”, *Computer Vision and Image Understanding* 68, 72-89, 1997.
- [16] Christmas W., Kittler J. and Petrou M., “Structural matching in computer vision using probabilistic relaxation”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 749-764, 1995.
- [17] Chung F.R.K, “*spectral graph theory*”, AMS Press, Providence Rhode Island, 1977
- [18] Cohen I., Ayache N. and Sulger P., “Tracking points on deformable objects using curvature information”, in *Proc. of European Conference on Computer Vision*, 458-466, 1992.
- [19] Costeira J. and Kanade T., “A multibody factorization method for motion analysis”, in *Proc. of International Conference on Computer Vision*, 1071-1076, 1995.
- [20] Davis L., “Shape matching using relaxation techniques”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 1, 60-72, 1979.
- [21] Deerwester S., Dumais S., Furnas G., Landauer T. and Harshman R., “Indexing by latent semantic indexing”, *J. of the American Society for Information Science*, 41, 391-407, 1990.
- [22] Del Bimbo A. and Pala P., “Visual image retrieval by elastic matching of user sketches”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19, 121-132, 1997.
- [23] Dembo A., Karlin S. and Zeitouni O., “Critical phenomena for sequence matching with scoring”, *Annals of Probability* 22, 1993-2021, 1994.
- [24] Donath W. and Hoffman A., “Lower bounds for the partitioning of graphs”, *IBM Journal of Research and Development* 17, 420-425, 1973.
- [25] Dubnov S., El-Yaniv R., Gdalyahu Y., Schneiderman E., Tishby N. and Yona G., “A new non parametric clustering algorithm”, submitted to *Machine Learning Journal*.
- [26] Duda O. and Hart E., “*Pattern classification and scene analysis*”, Wiley-Interscience, New York, 1973.
- [27] Edelman S. and Duvdevani-Bar S., “Similarity, connectionism, and the problem of representation in vision”, *Neural Computation* 9, 701-720, 1997.
- [28] 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.

- [29] Felzenszwalb P. and Huttenlocher D., "Image segmentation using local variation", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 98-104, 1998.
- [30] Fiduccia C. and Mattheyses R., "A linear time heuristic for improving network partitions", in *Proc. 19th Design Automation Conference*, 175-181, 1982.
- [31] Fiedler M., "Algebraic connectivity of graphs", *Czech. Math. Journal* 23, 298-305, 1973.
- [32] Flickner M., Sawhney H., Niblack W., Ashley J., Huang Q., Dom B., Gorkani M., Hafner J., Lee D., Petkovic D., Steele D. and Yanker P., "Query by image and video content: The QBIC system", *IEEE Computer* 28, 23-32, 1995.
- [33] Fukunaga K., *"Introduction to statistical pattern recognition"*, Academic Press, San Diego, 1990.
- [34] Funt B.V. and Finlayson G.D., "Color constant color indexing" *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 522-529, 1995.
- [35] Gdalyahu Y. and Weinshall D., "Automatic Hierarchical Classification of Silhouettes of 3D Objects", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 787-793, 1998.
- [36] Gdalyahu Y., Weinshall D. and Werman M., "A randomized algorithm for pairwise clustering", in *Advances in Neural Information Processing Systems* 11, 424-430, 1998.
- [37] Gdalyahu Y., Weinshall D. and Werman M., "Stochastic image segmentation by typical cuts", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition* vol II, 596-601, 1999.
- [38] Geiger D., Gupta A., Costa L. and Vlontzos J., "Dynamic programming for detecting, tracking and matching deformable contours", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 294-302, 1995.
- [39] Geller N., "On the citation influence methodology of Pinski and Narin", *Information Processing and Management*, 93-95, 1978.
- [40] 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.
- [41] Gibson D., Kleinberg J. and Raghavan P., "Clustering categorical data: an approach based on dynamical systems", in *Proc. 24th Int. Conf. Very Large Data Bases*, 311-322, 1998.
- [42] Golub G. and Van Loan C.F., *"Matrix Computation"*, Johns Hopkins University Press, 1989.
- [43] Gorman J., Mitchell O. and Kuhl F., "Partial shape recognition using Dynamic programming", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 10, 257-266, 1988.
- [44] Gowda K. and Krishna G., "Disaggregative clustering using the concept of mutual nearest neighborhood", *IEEE Trans. on Systems, Man, and Cybernetics* 8, 888-895, 1978.

- [45] Gray I. and Wilson D., “Nonnegative factorization of positive semidefinite nonnegative matrices”, *Linear algebra and its applications* 31, 119-127, 1980.
- [46] Gregor J. and Thomason M., “Dynamic programming alignment of sequences representing cyclic patterns”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 129-135, 1993.
- [47] Guedalia I., London M. and Werman M., “An on-line agglomerative clustering method for nonstationary data”, *Neural Computation* 11, 521-540, 1999.
- [48] Gupta A. and Jain R., “Visual Information Retrieval”, *Communications of the ACM* 40, 70-79, 1997.
- [49] Guy G. and Medioni G., “Inferring global perceptual contours from local features”, *International Journal of Computer vision* 20, 113-133, 1996.
- [50] Hafner J., Sawhney H.S., Equitz W., Flickner M. and Niblack W., “Efficient color histogram indexing for quadratic form distance functions”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 729-736, 1995.
- [51] Haralik R. and Shapiro L., “*Computer and Robot Vision*”, Addison-Wesley, Reading MA, 1992.
- [52] Hastie T. and Stuetzle W., “Principal curves”, *Journal of the American Statistical Association*, 84, 502-516, 1989.
- [53] Herault L. and Horaud R., “Figure-ground discrimination: a combinatorial optimization approach”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 899-914, 1993.
- [54] Hofman T., Puzicha J. and Buhmann J., “Unsupervised texture segmentation in a deterministic annealing framework”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 20, 803-818, 1998.
- [55] Hofmann T. and Puzicha J., “Statistical models for co-occurrence data”, Technical Memo, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Number AIM-1625, February 1998.
- [56] Hofmann T., “*Data clustering and beyond*”, PhD thesis, Bonn university, 1997.
- [57] Hofmann T. and Buhmann J., “Pairwise data clustering by deterministic annealing”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19, 1-14, 1997.
- [58] Horaud R. and Skordas T., “Stereo correspondence through feature grouping and maximal cliques”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 11, 1168-1180, 1989.
- [59] Hu T.C., “*Combinatorial Algorithms*”, Addison-Wesley Publishing Company, Reading MA, 1982.
- [60] Huttenlocher D., Klanderman G. and Rucklidge W., “Comparing images using the Hausdorff distance”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 850-863, 1993.

- [61] Huttenlocher D., Lilien R. and Olson C., "Object recognition using subspace methods", in *Proc. of European Conference on Computer Vision*, 536-545, 1996.
- [62] Huttenlocher D. and Ullman S., "Object recognition using alignment", in *Proc. of International Conference on Computer Vision*, 102-111, 1987.
- [63] Ishikawa H. and Geiger D., "Segmentation by grouping junctions", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 125-131, 1998.
- [64] Jacobs D. W., Weinshall D. and Gdalyahu Y., "Condensing image databases when retrieval is based on non-metric distances", in *Proc. of International Conference on Computer Vision*, 596-601, 1998.
- [65] Jain A. and Dubes R., *"Algorithms for clustering data"*, Prentice Hall, NJ, 1988.
- [66] Jaynes E., "On the rational of maximum entropy methods", *Proceedings of the IEEE* 70, 939-952, 1982.
- [67] Jerrum M. and Sinclair A., "The Markov chain Monte Carlo method: An approach to approximate counting and integration", in *Approximation algorithms for NP-hard problems*, D.S.Hochbaum ed., PWS Publishing, 1996.
- [68] Johnson W. and Lindenstrauss J., "Extension of Lipshitz mapping to Hilbert space", *Contemp. Math.* 26, 189-206, 1984.
- [69] Johnson E., Mehrotra A. and Nemhauser G., "Min-cut clustering", *Mathematical programming* 62, 133-151, 1993.
- [70] Kamger-Parsi B., Margalit M. and Rozenfeld A., "Matching general polygonal arcs", *CVGIP: image understanding* 53, 227-234, 1991.
- [71] Karger D., "Minimum cuts in near linear time", in *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, 56-63, 1996.
- [72] Karger D. and Stein C., "A new approach to the minimum cut problem", *Journal of the ACM* 43, 601-640, 1996.
- [73] Karp R. and Luby M., "Monte Carlo algorithms for planar multiterminal network reliability problems", *Journal of Complexity* 1, 45-64, 1985.
- [74] Kearns M. and Vazirani U., *"An introduction to computational learning theory"*, The MIT press, 1994.
- [75] Kernighan B. and Lin S., "An effective heuristic procedure for partitioning graphs", *The Bell System Technical Journal* 49, 291-308, 1970.
- [76] King V., Rao S. and Tarajan R., "A faster deterministic maximum flow algorithm", *Journal of Algorithms* 17, 447-474, 1994.

- [77] Kirkpatrick S., Gelatt C. and Vecchi M., "Optimization by simulated annealing", *Science* 220, 671-680, 1983.
- [78] Kleinberg J., "Authoritative sources in a hyperlink environment", in *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 668-677, 1998.
- [79] Kleinberg J. and Tomkins A., "Applications of linear algebra in information retrieval and hypertext analysis", in *Proc. ACM Symposium on Principles of Database Systems*, 185-193, 1999.
- [80] Klock H. and Buhmann J., "Multidimensional Scaling by Deterministic Annealing", in *Proc. of the International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 245-260, 1997.
- [81] Koch M. and Kashyap R., "Using polygons to recognize and locate partially occluded objects", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 9, 483-494, 1987.
- [82] Kruskal J., "A theorem about CONCOR", Technical report MH 2C-571, Bell laboratories, 1977.
- [83] Kruskal J.B. and Wish M., "Multidimensional Scaling", Sage Publications, Beverly Hills, CA, 1978
- [84] Lamdan Y., Schwartz J. and Wolfson H., "Affine invariant model based object recognition", *IEEE trans. on Robotics and Automation* 6, 578-589, 1990.
- [85] Lance G. and Williams W., "A general theory of classification sorting strategies: II. clustering systems", *Computer Journal* 10, 271-277, 1969.
- [86] Li S., "Matching: invariant to translations, rotations and scale changes", *Pattern Recognition* 25, 583-594, 1992.
- [87] Linial M., Linial N., Tishby N. and Yona G., "Global self organization of all known protein sequences reveals inherent biological signatures", *Journal of Molecular Biology* 268, 539-556, 1997.
- [88] Linial N., London E. and Rabinovich Y., "The geometry of graphs and some of its algorithmic applications", *Combinatorica* 15, 215-245, 1995.
- [89] Liu H. and Srinath M., "Partial shape classification using contour matching in distance transformation", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 12, 1072-1079, 1990.
- [90] Lu C. and Dunham J., "Shape matching using polygon approximation and dynamic alignment", *Pattern Recognition Letters* 14, 945-949, 1993.
- [91] Marzal A. and Vidal E., "Computation of normalized edit distance and applications", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 926-932, 1993.

- [92] McConnell R., Kwok R., Curlander J., Kober W. and Pang S., “ Ψ -S Correlation and dynamic time warping: two methods for tracking ice floes in SAR images”, *IEEE trans. on geoscience and remote sensing* 29, 1004-1012, 1991.
- [93] McQuitty L. and Clark J., “Clustering from iterative intercolumnar correlation analysis”, *Educational and Psychological Measurement*, 211-238, 1968.
- [94] Miller D. and Rose K., “Hierarchical, unsupervised learning with growing via phase transitions”, *Neural Computation* 8, 425-450, 1996.
- [95] Mokhtarian F. and Mackworth A., “Scale-based description and recognition of planar curves and two-dimensional shapes”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 8, 34-44, 1986.
- [96] Mokhtarian F., Abbasi S. and Kittler J., “Robust and Efficient Shape Indexing through Curvature Scale Space,” in *Proc. British Machine Vision Conference*, 53-62, 1996.
- [97] Motwani R. and Raghavan P., “*Randomized Algorithms*”, Cambridge University Press, New York 1995
- [98] Mumford D. and Shah J., “Boundary detection by minimizing functionals”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 22-26, 1985.
- [99] Nagamochi H. and Ibaraki T., “Computing edge connectivity in multigraphs and capacitated graphs”, *SIAM Journal of Discrete Mathematics* 5, 54-66, 1992.
- [100] Nastar C., Mitschke M. and Meilhac C., “Efficient Query Refinement for Image Retrieval”, in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 547-552, 1998.
- [101] Paatero P. and Tapper U., “Positive matrix factorization: a non negative factor model with optimal utilization of error estimates of data values”, *Environmetrics* 5, 111-126, 1994.
- [102] Pal N. and Pal S., “A review on image segmentation techniques”, *Pattern Recognition* 26, 1277-1294, 1993.
- [103] Papadimitriou C., Raghavan P., Tamaki H. and Vempala S., “Latent semantic indexing: a probabilistic analysis”, in *Proc. of the ACM Symposium on Principles of Databases Systems*, 159-168, 1998.
- [104] Pelillo M., “Relaxation labeling networks for the maximum clique problem”, *Journal of Artificial Neural Networks* 2, 313-328, 1995.
- [105] Pelillo M., Siddiqi K. and Zucker S., “Matching hierarchical structures using association graphs”, in *Proc. of European Conference on Computer Vision*, 3-16, 1998.
- [106] Pelillo M., “Replicator equations, maximal cliques, and graph isomorphism”, *Neural Computation* 11, 2023-2045, 1999.
- [107] Pentland A., Picard R. and Sclaroff S., “Photobook: Tools for Content-Based Manipulation of Image Databases”, *International Journal of Computer Vision* 18, 233-254, 1996.

- [108] Pereira F., Tishbi N. and Lee L., “Distributional clustering of English words”, in *Proc. of the Association for Computational Linguistics*, 183-190, 1993.
- [109] Perona P. and Freeman, W., “A factorization approach to grouping”, in *Proc. of European Conference on Computer Vision*, 655-670, 1998.
- [110] Pinski G. and Narin F., “Citation influence for journal aggregates of scientific publications: theory with application to the literature of physics”, *Information Processing and Management* 12, 297-312, 1976.
- [111] Rocha J. and Pavlidis T. “A shape analysis model with applications to a character recognition system”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 16, 393-404, 1994.
- [112] Rosch E. and Mervis C., “Family resemblance: studies in the internal structure of categories”, *Cognitive Psychology* 7, 573-605, 1975.
- [113] 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.
- [114] Rose K., “Deterministic annealing for clustering, compression, classification, regression, and related optimization problems”, *Proceedings of the IEEE* 86, 2210-2238, 1998.
- [115] Rosenfeld A., Hummel R. and Zucker S., “Scene labeling by relaxation operations”, *IEEE Trans. on Systems, Man, and Cybernetics* 6, 420-433, 1976.
- [116] Salce L. and Zanardo P., “Completely positive matrices and positivity of least squares solutions”, *Linear algebra and its applications* 178, 201-216, 1993.
- [117] 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
- [118] Sclaroff S. and Pentland A., “Modal matching for correspondence and recognition”, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 545-561, 1995.
- [119] 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.
- [120] Shapiro L. and Brady M., “Feature based correspondence: an eigenvector approach”, *Image and vision computing* 10, 283-288, 1992.
- [121] Sharon E., Brandt A. and Basri R., “Fast Multiscale Image Segmentation”, *preprint*.
- [122] Sharvit D., Chan J., Tek H. and Kimia B., “Symmetry based indexing of image databases”, *J. visual communication and image representation*, 1998.
- [123] 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.

- [124] Sheoard R. and Arabie P., "Additive clustering: Representation of similarities as combinations of discrete overlapping properties", *Psychological Review* 86, 87-123, 1979.
- [125] Shepard R., "Multidimensional scaling, tree fitting, and clustering", *Science* 210, 390-398, 1980.
- [126] Shepard R., "Toward a universal law of generalization for psychological science", *Science* 237, 1317-1323, 1987.
- [127] Shi J. and Malik J., "Normalized cuts and image segmentation", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 731-737, 1997.
- [128] Shokoufandeh A., Dickinson S., Siddiqi K. and Zuker S., "Indexing using a spectral encoding of topological structure", in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 491-497, 1999.
- [129] Siddiqi K., Shokoufandeh A., Dickinson S. and Zuker S., "Shock graphs and shape matching", in *Proc. of International Conference on Computer Vision*, 222-229, 1998.
- [130] Smith S., "Threshold validity for mutual neighborhood clustering", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 89-92, 1993.
- [131] Stockman G., Kopstein S. and Benett S., "Matching images to models for registration and object detection via clustering", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 4, 229-241, 1982.
- [132] Swain M.J. and Ballard D., "Color indexing", *International Journal of Computer Vision* 7, 11-32, 1991.
- [133] Thornber K. and Williams L., "Analytic solution of stochastic completion fields", *Biological Cybernetics* 75, 141-151, 1996.
- [134] Tirthapura S., Sharvit D., Klein P. and Kimia B., "Indexing based on edit distance matching of shape graphs", *SPIE Proc. on multimedia storage and archiving systems III*, 25-36, 1998.
- [135] Tsai W. and Yu S., "Attributed string matching with merging for shape recognition", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 7, 453-462, 1985.
- [136] Tsay Y. and Tsai W., "Attributed string matching by split and merge for on-line chinese character recognition", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 180-185, 1993.
- [137] Tversky A., "Features of similarity", *Psychological Review* 84, 327-352, 1977.
- [138] Ueda N. and Suzuki S., "Automatic shape model acquisition using multiscale segment matching", in *Proc. of International Conference on Pattern Recognition*, 897-902, 1990.
- [139] Ueda N. and Suzuki S., "Learning visual models from shape contours using multiscale convex/concave structure matching", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 337-352, 1993.

- [140] Umeyama S., "Parameterized point pattern matching and its application to recognition of object families", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15, 136-144, 1993.
- [141] Van Den Bout D. and Miller T., "Graph partitioning using annealed neural networks", *IEEE Trans. on Neural Networks* 1, 192-203, 1990.
- [142] Verri A., Uras C., Frosini P. and Ferri M., "On the use of size functions for shape analysis", *Biological Cybernetics* 70, 99-107, 1993.
- [143] Wang Y. and Pavlidis T., "Optimal correspondence of string subsequences", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 12, 1080-1086, 1990.
- [144] Wang S. and Swendsen R., "Cluster Monte Carlo algorithms", *Physica A* 167, 565-579, 1990.
- [145] Weinshall D., Jacobs D. and Gdalyahu Y., "Classification in non-metric spaces", in *Advances in Neural Information Processing Systems* 11, 838-844, 1998.
- [146] Weinshall D. and Werman M., "On View Likelihood and Stability", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19, 97-108, 1997.
- [147] Weiss Y., "Segmentation using eigenvectors: a unifying view", in *Proc. of International Conference on Computer Vision* vol II, 975-982, 1999.
- [148] Werman M. and Weinshall D., "Similarity and Affine Invariant Distance Between Point Sets", *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17, 810-814, 1995.
- [149] Williams L. and Jacobs D., "Stochastic completion fields: a natural model of illusory contour shape and salience", *neural computation* 9, 849-870, 1997.
- [150] Williams L. and Thornber K., "A comparison of measures for detecting natural shapes in cluttered background", NEC Research Institute Technical Report 97-2, January 24, 1997.
- [151] Wiseman S., Blatt M. and Domany E., "Superparamagnetic clustering of data", *Physical Review E* 57, 3767-3783, 1998.
- [152] 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.
- [153] Zhang J., "The convergence of mean field procedure for MRF's", *IEEE Trans. on Image Processing* 5, 1662-1665, 1991.