SHAPE RETRIEVAL USING CONTOUR FEATURES AND DISTANCE OPTIMIZATION

Daniel Carlos Guimarães Pedronette and Ricardo da S. Torres
Institute of Computing, University of Campinas (Unicamp), Campinas, Brazil

Keywords: Shape Description, Content-based image retrieval, Distance optimization.

Abstract: This paper presents a shape descriptor based on a set of features computed for each point of an object contour. We also present an algorithm for distance optimization based on the similarity among ranked lists. Experiments were conducted on two well-known data sets: MPEG-7 and Kimia. Experimental results demonstrate that the combination of the two methods is very effective and yields better results than recently proposed shape descriptors.

1 INTRODUCTION

The huge growth of image collections and multimedia resources available and accessible through various technologies is not new for years. The same can be said of the research motivated by the need of methods for indexing and retrieval these data. For two decades, several models of Content-Based Image Retrieval - CBIR have been proposed using features such as shape, color, and texture for retrieving images.

Shape is clearly an important cue for recognition since humans can often recognize characteristics of objects solely on the basis of their shapes. This distinguishes shape from other elementary visual features, which usually do not reveal object identity (Adamek and OConnor, 2004). Several shape descriptors proposed in the literature analyze certain features of shapes, measuring properties associated with each pixel of the object contour, such as angle (Arica and Vural, 2003) and area of regions (Alajlan et al., 2007). Those works have shown that many of those features are able to characterize the shape complexity of objects. However, most approaches uses only one feature.

In this paper, we propose a new shape description model that allows the combination of several features computed for each pixel of an object contour. We also propose a new distance optimization method for improving the effectiveness of CBIR systems. This method exploits the similarity among ranked lists to redefine the distance among images.

Several experiments were conducted on two widely used image collections: MPEG-7 and Kimia. Experiment results demonstrate that the combination of the proposed methods yields effectiveness performance that are superior than several shape descriptors recently proposed in the literature.

2 BASIC CONCEPTS

Several feature functions proposed in this paper compute feature values for each point of the object contour by analyzing neighbour pixels. The concept of neighborhood is also used to classify features as local, regional, or global. Two pixels are considered neighbors if their distance is not greater than a *radius of analysis*, which is defined below.

Definition 1. *The **radius of analysis** $r_a = max(D_{center}) \times \chi$, where D_{center}=$\{ d_{c_0}, d_{c_1}, ..., d_{c_n} \}$ is a set in domain $\mathbb{R}$, and each element d_{c_i} is defined by the Euclidean distance of each contour pixel to the center of mass of the object, and χ is a constant.*

Definition 2. *A **contour feature** f_j can be defined as set of real values f_j=$\{f_{0j}, f_{ij}, \ldots, f_{nj}\}$ (one for each contour pixel p_i) that represents features that can be extracted by a feature function $F_j : p_i \rightarrow f_{ij}$, where j identifies the feature, i defines the pixel p_i of the object contour and $f_{ij} \in \mathbb{R}$ is the value of feature f_j for the pixel p_i.*

Carlos Guimarães Pedronette D. and da S. Torres R. (2010).
SHAPE RETRIEVAL USING CONTOUR FEATURES AND DISTANCE OPTIMIZATION.
In *Proceedings of the International Conference on Computer Vision Theory and Applications*, pages 197-202
DOI: 10.5220/0002837201970202
Copyright © SciTePress

3 SHAPE DESCRIPTION BASED ON CONTOUR FEATURES

The proposed descriptor is based on the combination of features that describe the contour of an object within an image. These features can be classified as local, regional, or global, according to the proximity of the pixels that are analyzed.

3.1 Feature Vector Extraction

Let S_F=$\{F_0, F_1, \ldots, F_m\}$ be a set of feature functions based on contour, S_f=$\{f_0, f_1, \ldots, f_m\}$ be a set of features based on contour, and $C_{D_I} = \{p_0, p_1, \ldots, p_n\}$ be a set of pixels that defines the contour of an object. The **feature vector** $v_{\widehat{I}} = (v_1, v_2, \ldots, v_d)$ has $d = n \times m$ dimensions and can be represented by a matrix f_v, where each cell is given by the value of feature F_j applied to the pixel p_i, i.e., $f_v[i, j]$=$F_j(p_i)$. The feature vector extraction can be performed by applying a feature function F_j to each contour pixel p_i.

3.2 Local Features

Local features aims to characterize relevant properties in a small neighborhood of a contour pixel p_i. The extraction of local features is based on analyzing pixels wich are within the area defined by the *radius of analysis* (see Definition 1) from the pixel p_i. Changes in other regions of the object does not influence the values of local features.

Normal Angle. The *normal angle* Θ_n is defined as the angle between the *normal vector* and the horizontal line.

Concavity. this feature aims to characterize the concavity/convexity around the current pixel p_i. Pixels in concave regions present high values, whereas pixels in convex regions present low values. The concavity is computed from radially sample lines, which are traced emerging from the current pixel p_i. Sample lines are traced starting from normal vector Θ_n to both side of this vector, with a angular increment of c_a° or $-c_a^\circ$ until increments reachs $\Theta_n + 180^\circ$ or $\Theta_n - 180^\circ$. For each line, it is verified if object pixels are found by applying function $f_{inObject}$. This function is defined as follows: let p_j be a pixel at a distance r_a from the pixel p_i at the direction of the angle Θ_j, and let $S_{line} = \{p_{k_0}, p_{k_1}, ..., p_{k_n}\}$ be the sample line (set of points) traced from p_j at Θ_j direction. Let O_{D_I} be the set of pixels that composes the analyzed object. Function $f_{inObject}$ is defined as follow:

$$f_{inObject}(p_i, \Theta_j) = \begin{cases} 2, \text{ if } p_j \in O_{D_I} \\ 1, \text{ if } p_j \notin O_{D_I}, \text{ and } \exists p_{k_l} \in O_{D_I} \\ 0, \text{ otherwise} \end{cases}$$

The value of concavity is the number of sample lines that are traced after function $f_{inObject}$ assumes value 2. Algorithm 1 shows how to compute the concavity feature $f_{concavity}[p_i]$ of a pixel p_i. Variable $flag$ is used to determine if pixel $p_j \in O_{D_I}$ was found.

Opposite Opening. This feature aims to identify contour segments that are associated with branches or stems of the object, such as legs of animals and tree branches. Only pixels of these segments have the value of the opposite opening different of zero. The method to compute *opposite opening* is similar to that used for computing concavity. When tracing the sample lines starting from the normal vector direction, pixels belonging to the object are found at a distance r_a ($flag = 1$). After further angular variations, sample lines do not intercept object pixels at a distance r_a. Only these sample lines are used for computing the value of the opposite opening. Algorithm 1 shows the main steps used to compute this feature.

3.3 Global Features

We can obtain some features analyzing global properties of objects. These features are classified as global fetaures due to the fact that changes in any region of the object affect their values.

Distance to Center of Mass. The distance to center of mass d_{c_i} is given by the Euclidean distance between current pixel p_i and the pixel that represents the center of mass p_c.

Angle to Center of Mass. Let $\vec{v_{ac}}$ be a vector with origin at current pixel p_i in direction to the pixel of the center of mass p_c, the *angle to center of mass* Θ_c is given by the angle between this vector and the horizontal line.

3.4 Regional Features

Regional features characterize shape properties that depend on both global features (such center of mass) and the local features (such as normal angle).

Opposite Distance. The *opposite distance* d_o is computed from a sample line, which is traced emerging from the current pixel p_i in opposite direction to the normal vector. Let p_f be the contour pixel defined by the first intersection of this sample line and the object contour. The opposite distance is obtained by computing the Euclidean distance between the pixel p_i and the pixel p_f. The opposite distance is normalized by the maximum distance value

Difference between Normal Angle and Angle to the Center of Mass. This feature is computed from absolute distante between values of normal angle Θ_n and the angle to the center of mass Θ_c.

Opening. This feature aims to characterize information about the extent of the branches in a neighborhood of current pixel p_i. Similarly to the computation of the concavity, radially sample lines are traced and, for each angular increment, a test is performed. The value of opening $f_{opening}$ is equal to the value of concavity $f_{concavity}$ (where a pixel belonging to the object is located from a distance r_a of the pixel p_i) plus the number of sample lines that intercept the object for a distance greater than r_a. Algorithm 1 presents steps to perform the computation of this feature.

Algorithm 1. Computation of Concavity, Opening, and Opposite Opening.

Require: Current pixel p_i
Ensure: Computed features for p_i: concavity, opening, and opposite opening
1: $f_{concavity}[p_i] \leftarrow 0$
2: $f_{opening}[p_i] \leftarrow 0$
3: $f_{oppositeOpening}[p_i] \leftarrow 0$
4: $flag \leftarrow 0$
5: **for** $\Theta_j = \Theta_n$ *to* $(\Theta_n +/- 180°)$ **do**
6: **if** $f_{inObject}(p_i, \Theta_j) = 2$ **then**
7: $flag \leftarrow 1$
8: **end if**
9: $f_{concavity}[p_i] \leftarrow f_{concavity}[p_i] + flag$
10: **if** $flag$ **or** $f_{inObject}(p_i, \Theta_j) = 1$ **then**
11: $f_{opening}[p_i] \leftarrow f_{opening}[p_i] + 1$
12: **end if**
13: **if** $flag$ **and** $f_{inObject}(p_i, \Theta_j) \neq 2$ **then**
14: $f_{oppositeOpening}[p_i] \leftarrow f_{oppositeOpening}[p_i] + 1$
15: **end if**
16: $\Theta_j \leftarrow \Theta_j +/- c_a$
17: **end for**

In addition to the feature vectors, three other measures are computed for a given object: area, perimeter, and the maximum distance from the center of mass.

The feature vectors are sampled to the same size for all images in the collection. We used feature vectors with 1000 elements.

3.5 Histogram-based Features

A great challenge for matching feature vectors relies on the fact that many objects, though with similar shapes, present different sizes of contour for corresponding segments. One solution to alleviate this problem consists in using histogram-based features for characterizing the shape complexity. Histograms are computed as follows: considering the interval $[0, f_{max}]$, where f_{max} represents the maximum value of a particular feature, we divide this interval in h sub-intervals (in our experiments, $h = 11$). For each sub-interval, we compute the number of contour pixels whose feature value belongs to the sub-interval range. Thus, using the feature vector f_v as input, we can compute a histogram for each feature f_j of each object. Only for Normal Angle and Angle to Center of Mass, which are sensitive to rotation, histograms are not computed.

3.6 Irrelevant Contour Segments

There are segments of contour which have some instability in the shape contour. Due to small changes in the angle/perspective that the image is seen, such segments may disappear in the visual perception of the contour of the object. Figure 1 shows examples of objects (part A), and the irrelevant segments of contour in red (part B).

These segments can be detected by computing a *moving average* of the *opening* feature for the pixels of the contour. When the value of the moving average of the opening is too high (greater than a threshold $th_{opening}$), these points are discarded for the purpose of matching and distance computation. The size s_m of the moving average used for the experiments was s_m = 10 and threshold $th_{opening}$ = 45.0.

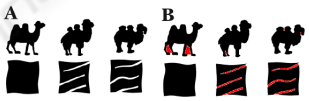

Figure 1: Irrelevant segments of contour.

4 DISTANCE COMPUTATION

4.1 Matching for Rotation Invariance

Feature vectors of similar objects may differ due to rotations. In order to solve this problem, a *matching* is performed. One feature vector is taken as reference, and the second feature vector is shifted considering different *offset* values. For each shift, the distance between the two feature vectors is computed. We call of *ideal offset* o_m for the matching that produces the minimun value for the computed distance.

The features used for similarity distance computation during the matching are the *distance to the center of mass* and *opposite distance*. The similarity distance for matching is defined as follows: $dist_{matching} = \overline{dist_c} + \overline{dist_o}$, where $dist_c$ and $dist_o$ define the value of distance for features *distance to the center of mass* and *opposite distance*, respectively. They are computed by applying the $L1$ distance.

4.2 Feature and Histogram Distances

The L1 distance is used after applying the best offset for computing the distance for a feature f_j between two objects. Distances are normalized as follows: $\overline{dist_{f_j}} = (avg(f_j)/max(f_j)^2) \times dist_{f_j}$, where f_j defines the values of feature j for each pixel p_i of the contour, $avg(f_j)$ defines the average of f_j, and $max(f_j)$ defines the maximum value of f_j, considering all images in the collection.

The L1 distace is also used to compute the distance between two histograms.

4.3 Final Distance

The **final distance** $dist_{final}$ is computed as follow: $dist_{final} = (dist_{aa} \times w_{aa}) + (dist_{ca} \times w_{ca}) + (dist_{gr} \times w_{gr}) + (dist_h \times w_h)$, where: $dist_{aa}$ is the sum of distances for angle to center of mass and normal angle; $dist_{ca}$ is the sum of distances for concavity, opening and opposite opening; $dist_{gr}$ is the sum of distances for the opposite distance, distance of center of mass, and distance between normal angle and angle to the center of mass; $dist_h$ is the sum of histogram distances.

The final distance depends also on the similarity of overall measures of the objects, such as *area* and *perimeter*. We propose to adjust the value of the final distance given these similarity values. Our strategy works as follows: the values of area and perimeter are normalized using the maximum distance of center of mass (for scale invariation). For each normalized values (area and perimeter), we check if the diffence of the area (perimeter) values is greater than a threshold th_{area} ($th_{perimeter}$). If so, the value of the final distance receives a penality $p > 1$, that is $dist_{final} \leftarrow dist_{final} \times p$. We used $w_{aa} = 12$, $w_{ca} = 3$, $w_{gr} = 60$, $w_h = 1$, $th_{area} = 0.7$, $th_{perimeter} = 3.5$, and $p = 1.1$.

5 DISTANCE OPTIMIZATION ALGORITHM

We can use the features defined in previous sections to process shape-based queries in an image collection $C = \{img_1, img_2, \ldots, img_n\}$. For a given query image img_q, we can compute the distance among img_q and all images of collection C. Next, collection images can be ranked according to their similarity to img_q, generating a ranked list R_q. We expect that similar images to img_q are placed at first positions of the ranked list R_q. In fact, it is possible to use the proposed features to compute the distances among all images of C. Let the matrix W be a distance matrix, where $W(k,l)$ is equal to the distance between images $img_k \in C$ and $img_l \in C$. It is also possible to compute ranked lists R_{img_k} for all images $img_k \in C$.

5.1 The Algorithm

Our strategy for distance optimization relies on exploiting the fact that if two images are similar, their ranked lists should be similar as well. Basically, we propose to redefine the distance among images, given the similarity of their ranked lists. A clustering approach is used for that. Images are assigned to the same cluster if they have similar ranked lists. Next, distances among images belonging to same cluster are updated (decreased). This process is repeated until the "quality" of the formed groups does not improve and, therefore, we have "good" ranked lists. We use a *cohesion* measure for determining the quality of a cluster.

Let $C = \{img_1, img_2, \ldots, img_n\}$ be a set (or cluster) of images. Let R_{img_k} be the ranked list of query image $img_k \in C$ with *size* images. R_{img_k} is created using the distances among images. The *cohesion* of C is computed based on its first top_n results of the ranked lists R_{img_k}. It is defined as follows:

$$cohesion(C) = \frac{\sum_{j=0}^{size} \sum_{i=0}^{top_n} (top_n - i) \times (top_n / c) \times S(i)}{size^2},$$

where c is a constant [1] that defines a weight for a position in the ranked list and S is a function S: i $\rightarrow \{0,1\}$, that assumes value 1, if C contains the image ranked at position i of the ranked list defined by query image $img_j \in C$ or assumes value 0, otherwise. Algorithm 2 presents the proposed distance optimization method used to redefine distances among images.

Algorithm 2 . Distance Optimization Algorithm.

Require: Distance matrix W
Ensure: Optimized distance matrix W_o
1: $lastCohesion \leftarrow 0$
2: $currentCohesion \leftarrow computeCohesion(W)$
3: **while** $curCohesion > lastCohesion$ **do**
4: $Cls \leftarrow createClusters(W)$
5: $W \leftarrow updateDistances(W, Cls)$
6: $lastCohesion \leftarrow currentCohesion$
7: $currentCohesion \leftarrow computeCohesion(W)$
8: **end while**
9: $W_o \leftarrow W$

Function $computeCohesion(W)$ returns the average *cohesion* considering all clusters defined by ranked lists. Function $createClusters()$ is responsible for creating clusters. It is detailed in Section 5.2. Finally, function $updateDistances()$ verifies, for each pair of images, if they are in the same group. If so, the

[1] We use c=10 in our experiments.

distance between them is updated, i.e., multiplied by a constant $c_{mult} < 1$. In our experiments, $c_{mult} = 0.95$.

5.2 Clustering Algorithm

We use a graph-based approach for clustering. Let $G(V,E)$ be a directed and weighted graph, where a vertex $v \in V$ represents an image. The weight w_e of edge $e = (v_i, v_j) \in E$ is defined by the ranking position of image img_j (v_j) at the ranked list of img_i (v_i).

Definition 3. *Let (k,l) be a ordered pair. Two images img_i and img_j are (k,l)-**similar**, if $w_{e_{i,j}} \leq k$ and $w_{e_{j,i}} \leq l$, where $e_{i,j} = (img_i, img_j)$ is the edge between images img_i and img_j.*

Definition 4. *Let $S_p = \{(k_0,l_0),(k_1,l_1),\ldots,(k_m,l_m)\}$ be a set of ordered pairs. Two images, img_i and img_j, are **cluster-similar** according to S_p, if $\exists (k_a,l_a) \in S_p | img_i$ and img_j are (k_a,l_a)-similar.*

Figure 2 illustrates how to determine if two images are *cluster-similar*. In this example, $S_p = \{(1,8),(2,6),(3,5),(4,4)\}$. First, it is checked if img_1 and img_2 are $(1,8)$-similar. In this case, if the image ranked at the first position of ranked list R_{img_1} is at one of the eight first positions of ranked list R_{img_2}, images img_1 and img_2 are $(1,8)$-similar. If not, the second pair of S_p is used, and so on.

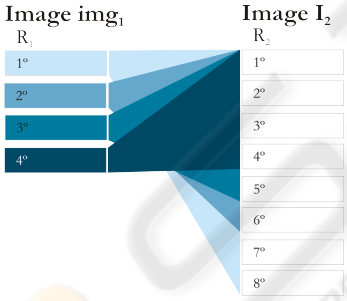

Figure 2: Example of *cluster-similarity* between images img_1 and img_2 with regard to $S_p = \{(1,8),(2,6),(3,5),(4,4)\}$.

Algorithm 3 shows the main steps for clustering images. The main step of the algorithm is the function *evaluateSimilarity*. This function is in charge of creating an initial set of clusters. Algorithms 4 and 5 show the main steps for creating image clusters. As it can be observed in step 7 of Algorithm 5, two images are assigned to the same cluster only if they are *cluster-similar* (Definition 4), according to a set S_p.

Function *mergeClusters*(*Clusters*) deals with clusters with only one image. Let R_{img} be the ranked list of the image of such a cluster. If the *cohesion* of the top_{nToAdd} images in R_{img} are greater than a threshold ($th_{cohesion}$), then *img* is added to the cluster that has more images of R_{img}. This function is also in charge of merging small-size clusters. A small-size cluster is added to a larger cluster. If the *cohesion* of this new group is greater than a threshold $th_{cohesion}$, these clusters are merged. Furthermore, the new group should have a *cohesion* greater than the average cohesion of the initial clusters. Weights are defined by the size of the initial clusters.

Algorithm 3. Clustering Algorithm.

Require: Graph G, $S_1 = \{(1,8),(2,6),(3,5),(4,4)\}$, $S_2 = \{(1,6),(2,4),(3,3)\}$.
Ensure: A set of clusters *Cls*.
1: $Cls \leftarrow null$
2: $Cls \leftarrow evaluateSimilarity(G(V,E),S_1,Cls)$
3: $Cls \leftarrow mergeClusters(Cls)$
4: $Cls \leftarrow divideClusters(Cls)$
5: $Cls \leftarrow evaluateSimilarity(G(V,E),S_2,Cls)$
6: $Cls \leftarrow mergeClusters(Cls)$

Algorithm 4. Algorithm evaluateSimilarity.

Require: Graph $G = (V,E)$ and S_p.
Ensure: Set of clusters *Cls*.
1: $Cls = \{\}$
2: **for all** i such that $0 \leq i \leq |V|$ **do**
3: $currentCluster = \{\}$
4: processImage (img_i,G,S_p)
5: $Cls \leftarrow Cls \cup currentCluster$
6: **end for**

Algorithm 5. Algorithm processImage.

Require: Image img_i, Graph $G = (V,E)$, and S_p.
1: **if** alreadyProcessed(img_i) **then**
2: return
3: **end if**
4: $currentCluster = currentCluster \cup img_i$
5: **for all** j such that $0 \leq j \leq |top_n|$ **do**
6: $img_j \leftarrow w_{e_{i,j}}$
7: **if** clusterSimilar(img_i,img_j,S_p) **then**
8: **if not** alreadyProcessd(img_j) **then**
9: processImage(img_j,G,S_p)
10: **else**
11: $currentCluster \leftarrow currentCluster \cup$ clusterOf(img_j)
12: **end if**
13: **end if**
14: **end for**

If the *cohesion* of a formed cluster is less than threshold $th_{cohesion}$, this cluster is splited and the status of its images is set to "non-processed" in the *divideClusters*(*Clusters*) function. These images are processed in steps 5 and 6 of Algorithm 3. In this case, $S_2 = \{(1,6),(2,4),(3,3)\}$ is used to defined the similarity among images.

Table 1: Results comparison in MPEG-7 dataset.

Descriptors	**Retrieval Rate**
BAS (Arica and Vural, 2003)	82.37%
CFD	**84.43%**
DSW (Alajlan et al., 2007)	85.03%
IDSC+DP (Ling and Jacobs, 2007)	85.40%
DSW+Global (Alajlan et al., 2007)	87.23%
Graph Trans. (Yang et al., 2008)	91.00%
CFD+DistOpt	**92.56%**

Table 2: Results of experiments in Kimia dataset.

Descriptors	1^o	2^o	3^o	4^o	5^o	6^o	7^o	8^o	9^o	10^o
SC (Sebastian et al., 2004)	97	91	88	85	84	77	75	66	56	37
CFD	99	98	98	99	97	90	86	86	68	56
IDSC+DP	99	99	99	98	98	97	97	98	94	79
Shape Tree (Felzenszwalb and Schwartz, 2007)	99	99	99	99	99	99	99	97	93	86
CFD + DistOpt	98	99	99	99	98	99	99	97	98	99
Graph Trans.	99	99	99	99	99	99	99	99	97	99

6 EXPERIMENTAL RESULTS

We conducted experiments in two image databases, both widely used in the literature.

The MPEG-7 data set consists of 1400 silhouette images grouped into 70 classes. Each class has 20 different shapes. The retrieval rate is measured by the number of shapes from the same class among the 40 most similar shapes. The following values were defined for the parameters used in the distance optimization method: $top_n = 40$, $th_{cohesion} = 70$, and $top_{nToAdd} = 10$. Table 1 shows the results of several descriptors using the MPEG-7 data set. The proposed methods are named CFD and $CFP + DistOpt$. $CFP + DistOpt$ considers both the feature description approach and the proposed distance optimization method. As it can be observed, $CFP + DistOpt$ has the best effectiveness performace when compared with several well-known shape descriptors.

We also present experimental results on the Kimia Data Set. This data set contains 99 shapes grouped into nine classes. In this case, the following parameters' values used in the distance optimization were used: $top_n = 15$, $th_{cohesion} = 7$, and $top_{nToAdd} = 4$. The retrieval results are summarized as the number of shapes from the same class among the first top 1 to 10 shapes (the best possible result for each of them is 99). Table 2 lists the number of correct matches of several methods. Again we observe that our approach yields a very high retrieval rate, being superior to several well-know shape descriptor.

7 CONCLUSIONS

In this paper, we have presented a shape description approach that combines features extracted from object contour. This paper has also presented a new optimization approach for reranking results in image retrieval systems. Several experiments were conducted using well-known data sets. Results demonstrate that the combination of both methods yield very high effectiveness performance when compared with important descriptors recently proposed in the literature.

Future work includes the investigation of new features to be incorporated into the proposed description framework. We also plan to use the distance optimization method with color and texture descriptors.

ACKNOWLEDGEMENTS

Authors thank CAPES, FAPESP and CNPq for financial support.

REFERENCES

Adamek, T. and OConnor, N. E. (2004). A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Trans. Circuits Syst. Video Techol.*, 14 i5:742–753.

Alajlan, N., El Rube, I., Kamel, M. S., and Freeman, G. (2007). Shape retrieval using triangle-area representation and dynamic space warping. *Pattern Recogn.*, 40(7):1911–1920.

Arica, N. and Vural, F. T. Y. (2003). Bas: a perceptual shape descriptor based on the beam angle statistics. *Pattern Recogn. Lett.*, 24(9-10):1627–1639.

Felzenszwalb, P. F. and Schwartz, J. D. (2007). Hierarchical matching of deformable shapes. *CVPR*, pages 1–8.

Ling, H. and Jacobs, D. W. (2007). Shape classification using the inner-distance. *PAMI*, 29(2):286–299.

Sebastian, T. B., Klein, P. N., and Kimia, B. B. (2004). Recognition of shapes by editing their shock graphs. *PAMI*, 26(5):550–571.

Yang, X., Bai, X., Latecki, L. J., and Tu, Z. (2008). Improving shape retrieval by learning graph transduction. In *ECCV*, pages 788–801.