INPUT OUTPUT

17.13 Shape Similarity

Input description: Two polygonal shapes, P_1 and P_2.

Problem description: How similar are P_1 and P_2?

Discussion: Shape similarity is a problem that underlies much of pattern recognition. Consider a system for optical character recognition (OCR). We are given a library of shape models representing letters, and unknown shapes obtained by scanning a page. We seek to identify the unknown shapes by matching them to the most similar shape models.

Shape similarity is an inherently ill-defined problem, because what "similar" means is application dependent. Thus, no single algorithmic approach can solve all shape-matching problems. Whichever method you select, expect to spend a large chunk of time tweaking it to achieve maximum performance.

Among your possible approaches are

- *Hamming distance* – Assume that your two polygons have been overlaid one on top of the other. *Hamming distance* measures the area of symmetric difference between the two polygons—in other words, the area lying within one of the two polygons but not both of them. When two polygons are identical and properly aligned, the Hamming distance is zero. If the polygons differ only in a little noise at the boundary, then the Hamming distance of properly aligned polygons will be small.

  Computing the area of the symmetric difference reduces to finding the intersection or union of two polygons (discussed in Section 17.8 (page 591)) and then computing areas (discussed in Section 17.1). But the difficult part of

computing Hamming distance is finding the right alignment of the two polygons. This overlay problem is simplified in applications such as OCR because the characters are inherently aligned within lines on the page and are not free to rotate. Efficient algorithms for optimizing the overlap of convex polygons without rotation are cited below. Simple but reasonably effective heuristics are based on identifying reference landmarks on each polygon (such as the centroid, bounding box, or extremal vertices) and then matching a subset of these landmarks to define the alignment.

Hamming distance is particularly simple and efficient to compute on bit-mapped images, since after alignment all we do is sum the differences of the corresponding pixels. Although Hamming distance makes sense conceptually and can be simple to implement, it captures only a crude notion of shape and is likely to be ineffective in most applications.

- *Hausdorff distance* – An alternative distance metric is *Hausdorff distance*, which identifies the point on P_1 that is the maximum distance from P_2 and returns this distance. The Hausdorff distance is not symmetrical, for the tip of a long but thin protrusion from P_1 can imply a large Hausdorff distance P_1 to P_2, even though every point on P_2 is close to some point on P_1. A fattening of the entire boundary of one of the models (as is liable to happen with boundary noise) by a small amount may substantially increase the Hamming distance yet have little effect on the Hausdorff distance.

  Which is better, Hamming or Hausdorff? It depends upon your application. As with Hamming distance, computing the right alignment between the polygons can be difficult and time-consuming.

- *Comparing Skeletons* – A more powerful approach to shape similarity uses thinning (see Section 17.10 (page 598)) to extract a tree-like skeleton for each object. This skeleton captures many aspects of the original shape. The problem now reduces to comparing the shape of two such skeletons, using such features as the topology of the tree and the lengths/slopes of the edges. This comparison can be modeled as a form of subgraph isomorphism (see Section 16.9 (page 550)), with edges allowed to match whenever their lengths and slopes are sufficiently similar.

- *Support Vector Machines* – A final approach for pattern recognition/matching problems uses a learning-based technique such as neural networks or the more powerful *support vector machines*. These prove a reasonable approach to recognition problems when you have a lot of data to experiment with and no particular ideas of what to do with it. First, you must identify a set of easily computed features of the shape, such as area, number of sides, and number of holes. Based on these features, a black-box program (the support vector machine training algorithm) takes your training data and produces a classification function. This classification function accepts as input the values

of these features and returns a measure of what the shape is, or how close it is to a particular shape.

How good are the resulting classifiers? It depends upon the application. Like any ad hoc method, SVMs usually take a fair amount of tweaking and tuning to realize their full potential.

There is one caveat. If you don't know how/why black-box classifiers are making their decisions, you can't know when they will fail. An interesting case was a system built for the military to distinguish between images of cars and tanks. It performed very well on test images but disastrously in the field. Eventually, someone realized that the car images had been filmed on a sunnier day than the tanks, and the program was classifying solely on the presence of clouds in the background of the image!

Implementations: A Hausdorff-based image comparison implementation in C is available at *http://www.cs.cornell.edu/vision/hausdorff/hausmatch.html*. An alternate distance metric between polygons can be based on its angle-turning function [ACH+91]. An implementation in C of this turning function metric by Eugene K. Ressler is provided at *http://www.cs.sunysb.edu/∼algorith*.

Several excellent support vector machine classifiers are available. These include the kernal-machine library (*http://www.terborg.net/research/kml/*), SVM^{light} (*http://svmlight.joachims.org/*) and the widely used and well-supported LIBSVM (*http://www.csie.ntu.edu.tw/∼cjlin/libsvm/*).

Notes: General books on pattern classification algorithms include [DHS00, JD88]. A wide variety of computational geometry approaches to shape similarity testing have been proposed, including [AMWW88, ACH+91, Ata84, AE83, BM89, OW85]. See the survey by Alt and Guibas [AG00].

The optimal alignment of n and m-vertex convex polygons subject to translation (but not rotation) can be computed in $O((n+m)\log(n+m))$ time [dBDK+98]. An approximation of the optimal overlap under translation and rotation is due to Ahn, et al. [ACP+07].

A linear-time algorithm for computing the Hausdorff distance between two convex polygons is given in [Ata83], with algorithms for the general case reported in [HK90].

Related Problems: Graph isomorphism (see page 550), thinning (see page 598).