Lecture 17: Segmentation (Edge Based, cont'd)

Last modified on February 26, 2000 at 6:00 PM

Contents

Reading

SH&B, 5.2.4–5.2.5

17.1 Global Processing: Graph Searching

Suppose that instead of finding clusters of linked edge pixels we want to find single-pixel thick curves for object boundaries.

One way to do this is to consider the image as being a graph where the vertices of the graph are the pixel corners and the edges of the graph are the pixel cracks. We can then use gradient or other edge-detecting operators to assign costs to each edge. Edge-like cracks would have low costs; cracks that do not look like edges would have high cost. The problem of finding an optimal boundary through as set of possible pixels thus becomes a simple minimal-cost path, graph-searching problem. (And of course, you should all know how to find minimum-cost paths through graphs.)

There are four basic subparts to this problem:

1. How to contruct the graph,
2. How to assign costs,
3. How to choose starting and ending points, and
4. How to find the minimum-cost path.

Let's take a look at an example before we discuss each of these four in more detail.

17.1.1 Example: Top-to-Bottom Scan

Graph Formulation

As an example, let's consider the case where we know that the boundary runs from the top to the bottom of some subset of the image without looping backwards. We can thus build a starting node representing the top of the image. The edges proceeding out from this node are all vertical cracks on the top row of the image. For each of these edges, successors include the corresponding vertical crack on next row, the horizontal crack to the left, and the horizontal crack to the right. Each of these has successors that go down vertically or sideways horizontally. Eventually, we get to the bottom row of the image where all paths lead to the end node.

Cost Function

In this simple case, we can let the cost of the edge along the crack between pixels p and q be

$$c(p, q) = I_{\text{max}} - |I(p) - I(q)| \tag{17.1}$$

where f_{max} is the maximum pixel value. Remember that we're trying to trying to trace between the inside (one intensity) and the outside (presumably another intensity) of an object.

Starting/Ending Points

In this case, selecting the starting and ending nodes is easy—they're simply the top and bottom of the image.

Graph Searching

The following is an algorithm for finding the minimum cost path.

Let $g(n)$ be the cost from the start point to point n.
Let $h(n)$ be an estimate of the cost from n to the end point.
Let $r(n) = g(n) + h(n)$

1. Mark the start point s as open and set $g(s) = 0$.
2. In no node is open, exit with a failure; otherwise continue.
3. Mark closed the open node n whose estimate $r(n)$ is the smallest.
4. If n is the end point, the least-cost path has been found—the least-cost path can be found by following the pointers back from n to s.
5. Expand node n by generating all of its successors.
6. If a successor n_i is not marked, set
$$g(n_i) = g(n) + c(n, n_i)$$
mark it as open, and direct pointers from it back to n.
7. If a successor is already marked, update the cost:
$$g(n_i) = \min(g(n_i, g(n) + c(n, n_i)$$
and set the pointer for n_i to n. Mark as open those successors that were so updated.
8. Go back to step 2.

Algorithm 17.1: Simple graph-searching algorithm

(We'll work through this in class to show how it works).

17.2 Graph Formulation

Suppose that we know roughly where the boundary lies or some other information that constrains the location of the boundary. One such case is the example we just used.

The example is obviously a simple case, but we can do similar things for more complicated cases. If we know that the boundary goes around the object with a monotomically increasing radial angle (i.e., it may move towards or away from the center as it goes around, but it never doubles back as one sweeps around), we can construct a constraining

band around the potential boundary. We can also construct our graph so that it at each stage it sweeps radially around the image, but can vary in radial distance.

Both of these formulations give us discrete multistage graphs, either as we move from row to row or radially around the boundary. Sometimes, though, we may want to allow the graph to curve and wind, passing through possibly *any* pixel. In such a formulation, we can allow unoriented edges along any pixel crack in the image. In other words, an $N \times N$ image would have $N-1 \times N-1$ nodes and $N-1 \times N-1$ edges. Our search could be between any two vertices and could trace through any of the edges. This is obviously a huge graph and a compute-intensity search.

17.3 Cost Functions

The cost function in the example uses only a simple 2-element mask. Better cost functions can be constructed by using

- better gradient operators,
- gradient orientation,
- other edge-determining operators such as Laplacian zero-crossings. (Remember that this gives you the position, not the magnitude of the edge—we can use it as a cost function by giving lower costs to pixels closer to the zero crossings.)

All of the above possible cost functions are based on the image data, not properties of the boundary curve itself. If we want smoother contours, we can add more cost for paths that would cause greater curvature.

Cost properties based on image data are called *external cost* terms; cost properties based on the boundary curve are called *internal cost* terms. Notice that internal cost for an edge is not fixed—it depends on the curve preceding the edge.

17.4 Selecting Start/End Points

There are three common ways to select starting and ending points:

- Use some known special points.
- Find ”landmark” points—ones that have certain special geometric properties.
- Ask the user.

One important property of the starting and ending points is that thier selection shouldn’t dramatically change the result. While this sometimes happens, it is not a good thing.

17.5 Graph Searching

Finding minimum-cost paths is a very old and well-studied topic in Computer Science. Common ways include:

- Exhausting searching. (Yuck!)
- Greedy algorithms
- Expansion-style algorithms
- Expansion with heuristic algorithms for pruning
- Dynamic programming

Notice that all of these involve optimization. This is a recurring throughout segmentation and other parts of Computer Vision: set up the solution as the optimization of some criteria-evaluation function.

17.5.1 Greedy Algorithms

Greedy algorithms attempt to get the most at each step. Simple pixel-by-pixel linking by taking the best successive edge would be one such greedy algorithm.

17.5.2 Expansion Algorithms

The algorithm discussed earlier in the example (and in your book) is an expansion-based one. The idea is to grow the search outwards looking for optimal additions to each path. When we finally get a path that leads to the end point, it is guaranteed to be optimal. One such method is *Dijsktra's algorithm*.

17.5.3 Expansion with Heuristics for Pruning

If we can estimate using some type of heuristics (guesses) what the remainder of a past might cost, we can prune unproductive paths. These might miss optimal paths through indiscriminate pruning, but it has been shown that if the heuristic is truly a lower bound on the remaining cost (though not necessarily exact), this finds the optimal path. An example of this is the one shown in Algorithm 17.1. The classic paper on using heuristic graph-searching methods for boundary finding is Martelli, 1976.

17.5.4 Dynamic Programming

Suppose that the graph has several discrete stages through which the paths must go. It can be shown that we can find the optimal path by finding the optimal path from the start to each node in the first stage, from each node in the first stage to the second, and so forth. The optimal path from the start to the second stage is the one whose sum from the start to the first stage plus the first to the second stage is minimal. This type of algorithm is called *dynamic programming*.

17.6 Global vs. Local Methods

This lecture highlights one recurring theme in Computer Vision: local processing vs. global processing. Sometimes we can use strictly-local processing, but we may miss more general properties in the image. We may also use strictly-global methods, but while they might optimize some global criteria they may not seem like ideal solutions when viewed locally. This balance between both local and global optimization is one we'll see frequently in this course.

Vocabulary

- Graph-based contour following
- Cost function
- Greedy algorithm
- Heuristic graph searching
- Dynamic programming