Adaptive Boundary Detection Using “Live-Wire” Two-Dimensional Dynamic Programming

William A. Barrett
william_barrett@byu.edu

Bryan S. Morse
morse@byu.edu

Eric N. Mortensen

Jayaram Udupa

Adaptive Boundary Detection Using "Live-Wire" Two-Dimensional Dynamic Programming

Eric Mortensen, Bryan Morse, William Barrett, Jayaram Udupa*

Department of Computer Science, Brigham Young University, Provo, Utah
*Medical Image Processing Group, Department of Radiology, Univ. of Pensylvania

Abstract

An adaptive boundary detection algorithm that uses two-dimensional dynamic programming is presented. The algorithm is less constrained than previous one-dimensional dynamic programming algorithms and allows the user to interactively determine the mathematically optimal boundary between a user-selected seed point and any other dynamically selected "free" point in the image.

Interactive movement of the free point by the cursor causes the boundary to behave like a "live wire" as it adapts to the new minimum cost path between the seed point and the currently selected free point. The algorithm can also be adapted or customized to learn boundary-defining features for a particular class of images.

Adaptive 2D DP performs well on a variety of images (angiocardiograms, CT, MRI). In particular, it accurately detects the boundaries of low contrast objects, such as occur with intravenous injections, as well as those found in noisy, low SNR images.

1. Introduction

Defining an object's boundary is a general problem in medical imaging as well as many other image processing fields. The segmentation problem (ie. defining the boundaries/areas of the objects of interest in an image) has not been solved in a fully automated way. Many current edge following techniques exist which employ local edge gradient and/or orientation information combined, at times, with some idea of an object template. Such techniques are limited by the relative local strength of the edge "criteria" as compared to the criteria for neighboring edges and/or noise.

Dynamic programming (DP) attempts to overcome the problems associated with using only local information for edge following. It does this by employing both local gradient information with global "edge cost" information. In general, one-dimensional dynamic programming (1-D DP)[1-4,6] attempts to discover a globally optimal edge path but imposes directional sampling and searching constraints for two-dimensional (2-D) images: thus requiring 2-D boundary templates.

This paper presents a new technique in dynamic programming which allows freedom in two variables (2-D) as compared to freedom in one variable for 1-D DP. Two-dimensional dynamic programming (2-D DP) may discover a globally optimal boundary path that is allowed to vary freely between any given starting seed point and any other free point in the image. Further, 2-D DP can be used to dynamically and interactively define the desired boundary using an active contour ("live-wire") with minimal user interaction--typically two to four seed points per boundary.

2. 2-D dynamic programming

As with 1-D DP, 2-D dynamic programming can be formulated as a directed graph-searching problem. The goal is to find the globally "optimal" (least cost or greatest cost) path between start and end points or nodes in the directed graph.

Formulating dynamic programming as a graph-searching problem requires nodes and arcs between nodes. For 2-D images, pixels become nodes with (initial) local costs calculated for each pixel and its neighbors. Arcs are defined as connections between a pixel and its neighbors. We define the globally "optimal" path as the minimum cumulative cost path between the start and end points.

2.1 Connectivity

The basic difference between 1-D and 2-D dynamic programming lies in defining the connectivity between nodes/pixels. 1-D DP constrains node connections to be in the approximate direction of the end node. That is, node connections must be "in front" of each current node towards the end node. Thus, 1-D DP constrains the path to freedom in only one variable (y).

However, 2-D DP allows freedom in two variables (x and y). Thus, node connections exist for a node's entire neighborhood (ie. in front, to the side, and in back). Figure 1 illustrates the differences between 1-D and 2-D connectivity.

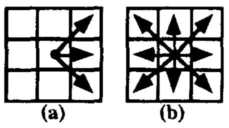


Figure 1.(a) 1-D DP and **(b)** 2-D DP connectivity.

0276-6547/92 $3.00 © 1992 IEEE

These connectivity differences can result in a different globally optimal 1-D DP path than for a 2-D DP path. Figure 2 shows how a globally optimal 1-D DP path can differ from a 2-D DP path. As can be seen, 1-D DP must cut across the peninsula since it can only search forward (in the x direction).

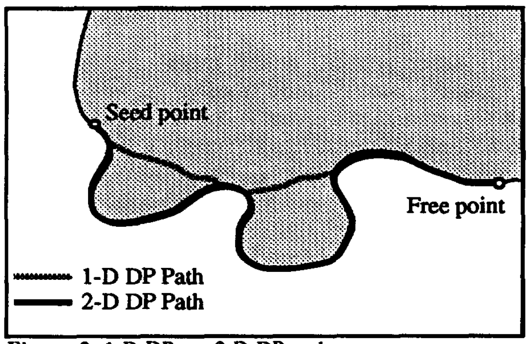


Figure 2. 1-D DP vs. 2-D DP path.

2.2 Cost matrix

Initially, we generate a two-dimensional "cost matrix" where every element in the matrix corresponds to an image pixel with the cost defined by the pixel's local boundary criteria. The cost matrix is generated as a function of the image's gradient magnitude and gradient orientation. Letting G_x and G_y represent the horizontal and vertical gradients of the image, then the cost matrix c(x,y) is generated as follows:

$$G(x,y) = \sqrt{G_x^2(x,y) + G_y^2(x,y)}$$
$$O(x,y) = \tan^{-1}\left(\frac{G_x(x,y)}{G_y(x,y)}\right)$$
$$c(x,y) = [\max_{x,y}(G(x,y)) - G(x,y)] + \alpha |O(x,y) \circledast f - O(x,y)|$$

where G(x,y) and O(x,y) are the gradient magnitude and orientation images respectively, f is an averaging or gaussian filter and α is a scaling factor. Note that the gradient magnitude image is subtracted from the maximum gradient magnitude value so that strong gradients represent low costs. Further, the absolute difference between the low-pass filtered orientation image and itself will also be low if a pixel's neighborhood contains gradient orientations that are similar to itself. Thus, low local edge costs correspond to pixels with strong gradient magnitude and gradient orientations similar to its neighbors. Figure 3a is an example cost matrix.

Associated with the cost matrix is a connection weighting matrix. Each connection is "weighted" such that diagonal connections have higher cost than horizontal or vertical connections. The horizontal and vertical connections have unity weights whereas the diagonal connections are weighted by $\sqrt{2}$, thus maintaining Euclidean distance weighting for the neighborhood.

2.3 Optimal path generation

Since the cost matrix and graph connections are defined in image space terms, image space terms will be used to describe the 2-D DP algorithm.

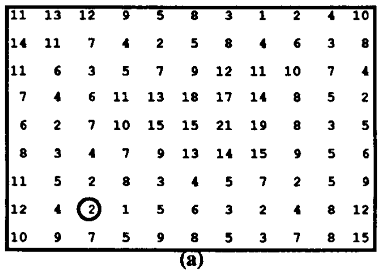
Unlike most other dynamic programming or graph-searching algorithms, we do not define both a start point and end point. Rather, we calculate the globally optimal path from a start or seed point to all other points/pixels in the image in an incremental fashion. The algorithm is similar to a well-known heuristic search algorithm[5] and is as follows:

c_x	-> local cost for point x.
p_x	-> optimal path pointer from point x.
w_{xy}	-> connection weight between points x and y.
T_x	-> total accumulated cost to point x.
L	-> list of "active" points sorted by total costs (initially empty).

```
add seed point to L;
while L not empty do begin
    x <- min(L)
    for each y in the neighborhood of x do
        if y not processed then begin
            $T_y$ <- $T_x + w_{xy}c_y$;
            add y to L;
            $p_y$ <- x;                    { set pointer from y to x }
            if y is a diagonal then mark $p_y$ as diagonal
        else if $p_y$ marked as diagonal and
              y not a diagonal then begin
            if $T_x + w_{xy}c_y < T_y$ then begin
                remove y from L;
                $T_y$ <- $T_x + w_{xy}c_y$;
                add y to L;
            end
        end
    end
end
```

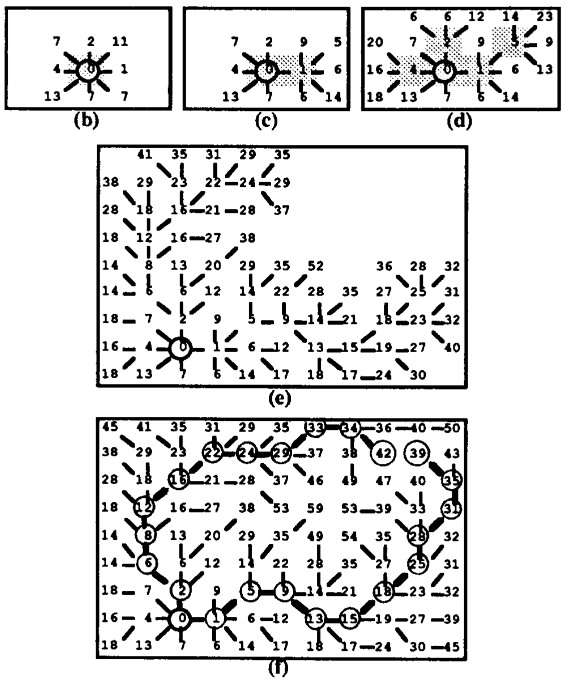
Starting with a user-defined seed point, the algorithm places that point on an initially empty sorted list, L. The point, x, with minimum total cost, T_x (initially on the seed point), is then removed from the list and checked for unprocessed neighbors. A total cost, T_y, is then computed for each unprocessed point y in the neighborhood of x. T_y is computed as the sum of the total cost to x, T_x, and the weighted local cost from x to y, $w_{xy}c_y$. Each neighboring point, y, along with its associated total cost, is added to the sorted list for later processing and the optimal path pointer for the neighbor is set to the point being processed. If the neighboring point is a diagonal then it is marked as such since it may be necessary to recompute the total cost to that point.

Figure 3 gives an example of how the total cost and the least cost path is computed. Figure 3a is the initial local cost matrix with the seed point circled. Figure 3b shows the total cost/path matrix after the seed point has been processed. Figure 3c shows the matrix after processing 2 points--the seed point and the next lowest total cost point on the sorted list. Notice how the points diagonal to the seed point have changed total cost and direction pointers. The Euclidean weighting between the seed and diagonal points make them more expensive than non-diagonal paths. Figures 3d, 3e, and 3f show the matrix at various stages of completion. Note how the algorithm produces a "wave-front" of active points and that the wave-front grows out faster were there are lower costs. Thus the wave-front grows out more quickly along edges in the image.



11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15

(a)



(b) (c) (d) (e) (f)

Figure 3.(a) Initial local cost matrix. (b) Seed point (shaded) processed. (c) 2 points (shaded) processed. (d) 5 points (shaded) processed. (e) 47 points processed. (f) Finished total cost and path matrix with two of many paths indicated.

2.4 Live-wire 2-D DP

Once the path matrix is finished, a boundary path can be chosen dynamically via a "free" point. Interactive movement of the free point by cursor position causes the boundary to behave like a live-wire as it adapts to the new minimum cost path. Thus, by constraining the seed point and the free point to lie near a given edge, the user is able to interactively "wrap" the live-wire boundary around the object of interest. When movement of the free point causes the boundary to digress from the desired object edge, input of a new seed point prior to the point of departure reinitiates the 2-D DP boundary detection. This causes potential paths to be recomputed from the new seed point, while effectively "tieing off" the boundary computed up to the new seed point. Figures 4 and 5 show two example images (an MRI scan and a left ventriculogram) and indicate how many seed points each outlined object required.

3. Results

Adaptive 2-D DP performs well on a variety of images (angiocardiograms, CT, MRI). In particular, it accurately detects the boundaries of low contrast objects, such as occur with intravenous injections, as well as those found in noisy, low SNR images. Boundaries are typically detected with two to four seed points.

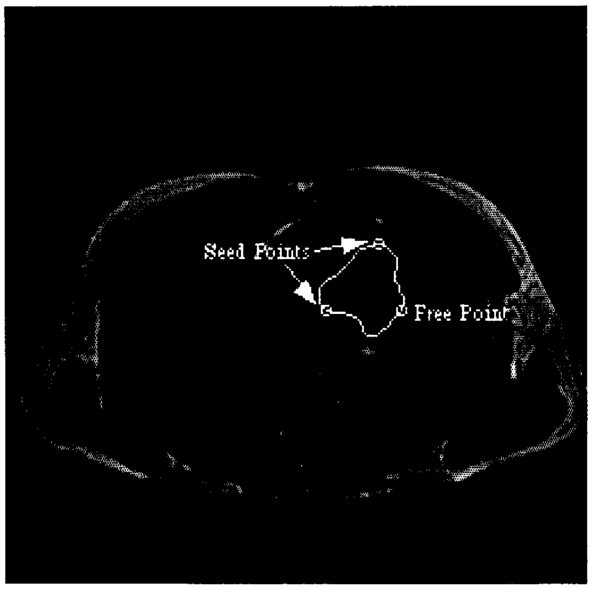


Figure 4. MRI scan with left ventricle outlined and seed points and free point circled.

The algorithm's computational complexity for a N image pixels is O(N). This can been seen by examining the algorithm in a worst case situation. Suppose first that all the weights were unity, then once a total cost to a point is computed, it is not computed again since that total cost already represents the minimum cost to that pixel. Thus the total cost is computed only once for each of the N pixels. There is also some computation required to add the point to a sorted list. But the unique conditions of this algorithm allow us to use a sort algorithm that requires only an array indexing operation (indexed by total cost) and changing two pointers. Thus, the computation complexity for sorting N points is N . Now, since the diagonal weights are not unity, then it may become necessary to recompute the cost to those points processed initially as diagonal neighbors. In the worst case, this will have to be done for half the points in the matrix since half of the points will be initially processed as diagonal neighbors. This means in the worst case that total costs for N/2 points will have to be recomputed and those points will have to be re-added to the sorted list again. The algorithm's computational complexity is therefore N (to calculate the total costs) + N (to add N points to the sorted list) + N (to recompute the costs for diagonals and add them again to the list) = 3N, or O(N). This is comparable to the complexity for the more restricted 1-D DP algorithm.

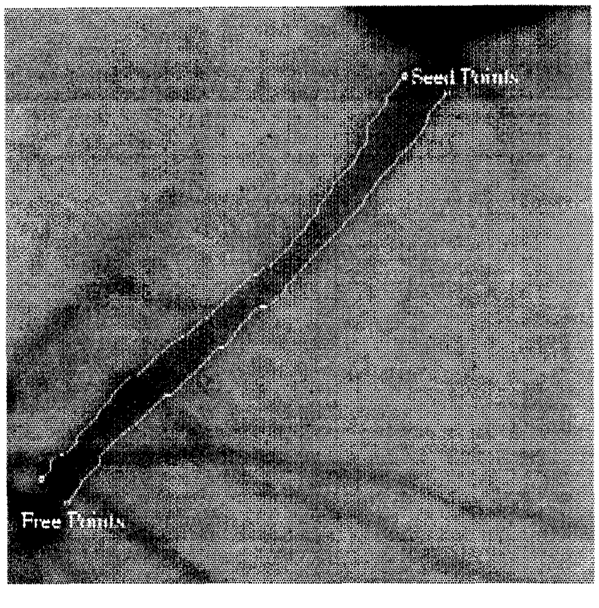


Figure 5. Coronary angiogram with vessel outlined and seed and free points circled.

The algorithm was implemented on a IBM compatible 33 MHz '386 with '387 co-processor and a hardware imaging board. Generating the cost matrix for a 512x512 image requires approximately 45 seconds but only needs to be done once per image. The user then selects a seed point interactively with the mouse. "Growing" the optimal path map requires up to one and a half minutes per seed point for a 512x512 image but this process in usually interrupted after 15 to 20 seconds when the DP wave-front encloses the desired object or point. The user can then use the path matrix to interactively wrap a boundary around the desired object. Though this process requires a small delay for each seed point, we are currently porting this algorithm to HP workstations where we expect to generate the optimal path matrix at interactive speeds.

4. Conclusions

An algorithm has been presented for iterative determination of globally optimal paths derived from local gradient magnitude and orientation information. The algorithm uses two-dimensional dynamic programming and can be applied to a variety of image types and anatomy. 2-D DP performs well based on visual comparison and is the same order of complexity as the more restrictive 1-D DP.

By calculating the optimal path from all points to a seed point, 2-D DP accommodates interactive selection of the desired optimal path via a "live-wire", making it a valuable interactive tool for defining an object's boundaries.

References

[1] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[2] Y. P. Chien and K. S. Fu, "A Decision Function Method for Boundary Detection," *Computer Graphics and Image Processing*, Vol. 3, No. 2, pp. 125-140, June 1974.

[3] A. Martelli, "An Application of Heuristic Search Methods to Edge and Contour Detection," *Communications of the ACM* Vol.19, No. 2, pp. 73-83, February 1976.

[4] U. Montanari, "On the Optimal Detection of Curves in Noisy Pictures," *Communications of the ACM*, Vol. 14, No. 5, pp. 335-345, May 1971.

[5] N. J. Nilsson, *Principles of Artificial Intelligence*, Palo Alto, CA:Tioga, 1980.

[6] D. L. Pope, D. L. Parker, D. E. Gustafson, and P. D. Clayton, "Dynamic Search Algorithms in Left Ventricular Border Recognition and Analysis of Coronary Arteries," *IEEE Proceedings of Computers in Cardiology*, pp. 71-75, September 1984.