for each table entry for ϕ do

for each S and θ

$$x_c := x + r(\phi) S \cos[\alpha(\phi) + \theta]$$

$$y_c := y + r(\phi) S \sin[\alpha(\phi) + \theta]$$

Finally, step 2.2b is now

$$A(x_c, y_c, S, \theta) := A(x_c, y_c, S, \theta) + 1$$

4.4 EDGE FOLLOWING AS GRAPH SEARCHING

A graph is a general object that consists of a set of nodes $\{n_i\}$ and arcs between nodes $< n_i, n_j >$. In this section we consider graphs whose arcs may have numerical weights or *costs* associated with them. The search for the boundary of an object is cast as a search for the lowest-cost path between two nodes of a weighted graph.

Assume that a gradient operator is applied to the gray-level image, creating the magnitude image $s(\mathbf{x})$ and direction image $\phi(\mathbf{x})$. Now interpret the elements of the direction image $\phi(\mathbf{x})$ as nodes in a graph, each with a weighting factor $s(\mathbf{x})$. Nodes $\mathbf{x}_i$, $\mathbf{x}_j$ have arcs between them if the contour directions $\phi(\mathbf{x}_i)$, $\phi(\mathbf{x}_j)$ are appropriately aligned with the arc directed in the same sense as the contour direction. Figure 4.10 shows the interpretation. To generate Fig. 4.10b impose the following restrictions. For an arc to connect from $\mathbf{x}_i$ to $\mathbf{x}_j$, $\mathbf{x}_j$ must be one of the three possible eight-neighbors in front of the contour direction $\phi(\mathbf{x}_i)$ and, furthermore, $g(\mathbf{x}_i)$

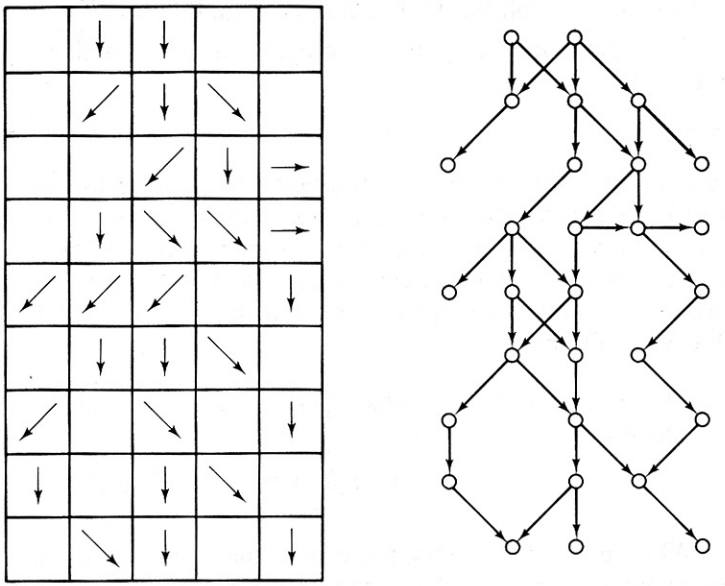


Fig. 4.10 Interpreting a gradient image as a graph (see text).

$> T$; $g(\mathbf{x}_j) > T$, where T is a chosen constant, and $|\{[\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)] \bmod 2\pi\}| < \pi/2$. (Any or all of these restrictions may be modified to suit the requirements of a particular problem.)

To generate a path in a graph from $\mathbf{x}_A$ to $\mathbf{x}_B$ one can apply the well-known technique of heuristic search [Nilsson 1971, 1980]. The specific use of heuristic search to follow edges in images was first proposed by [Martelli 1972]. Suppose:

1. That the path should follow contours that are directed from $\mathbf{x}_A$ to $\mathbf{x}_B$
2. That we have a method for generating the successor nodes of a given node (such as the heuristic described above)
3. That we have an evaluation function $f(\mathbf{x}_j)$ which is an estimate of the optimal cost path from $\mathbf{x}_A$ to $\mathbf{x}_B$ constrained to go through $\mathbf{x}_j$

Nilsson expresses $f(\mathbf{x}_i)$ as the sum of two components: $g(\mathbf{x}_i)$, the estimated cost of journeying from the *start node* $\mathbf{x}_A$ to $\mathbf{x}_i$, and $h(\mathbf{x}_i)$, the estimated cost of the path from $\mathbf{x}_i$ to $\mathbf{x}_B$, the *goal node.*

With the foregoing preliminaries, the heuristic search algorithm (called the A algorithm by Nilsson) can be stated as:

Algorithm 4.4: Heuristic Search (the A Algorithm)

1. "Expand" the start node (put the successors on a list called OPEN with pointers back to the start node).
2. Remove the node $\mathbf{x}_i$ of minimum f from OPEN. If $\mathbf{x}_i = \mathbf{x}_B$, then stop. Trace back through pointers to find optimal path. If OPEN is empty, fail.
3. Else expand node $\mathbf{x}_i$, putting successors on OPEN with pointers back to $\mathbf{x}_i$. Go to step 2.

The component $h(\mathbf{x}_i)$ plays an important role in the performance of the algorithm; if $h(\mathbf{x}_i) = 0$ for all i, the algorithm is a *minimum-cost search* as opposed to a *heuristic search.* If $h(\mathbf{x}_i) > h^*(\mathbf{x}_i)$ (the actual optimal cost), the algorithm may run faster, but may miss the minimum-cost path. If $h(\mathbf{x}_i) < h^*(\mathbf{x}_i)$, the search will always produce a minimum-cost path, provided that h also satisfies the following consistency condition:

> If for any two nodes $\mathbf{x}_i$ and $\mathbf{x}_j$, $k(\mathbf{x}_i, \mathbf{x}_j)$ is the minimum cost of getting from $\mathbf{x}_i$ to $\mathbf{x}_j$ (if possible), then
>
> $$k(\mathbf{x}_i, \mathbf{x}_j) \geqslant h^*(\mathbf{x}_i) - h^*(\mathbf{x}_j)$$

With our edge elements, there is no guarantee that a path can be found since there may be insurmountable gaps between $\mathbf{x}_A$ and $\mathbf{x}_B$. If finding the edge is crucial, steps should be taken to interpolate edge elements prior to the search, or gaps may be crossed by using the edge element definition of [Martelli 1972]. He defines

edges on the image grid structure so that an edge can have a direction even though there is no local gray-level change. This definition is depicted in Fig. 4.11a.

4.4.1 Good Evaluation Functions

A good evaluation function has components specific to the particular task as well as components that are relatively task-independent. The latter components are discussed here.

1. *Edge strength.* If edge strength is a factor, the cost of adding a particular edge element at $\mathbf{x}$ can be included as

$$M - s(\mathbf{x}) \qquad \text{where } M = \max_{\mathbf{x}} s(\mathbf{x})$$

2. *Curvature.* If low-curvature boundaries are desirable, curvature can be measured as some monotonically increasing function of

$$diff[\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)]$$

where diff measures the angle between the edge elements at $\mathbf{x}_j$ and $\mathbf{x}_i$.

3. *Proximity to an approximation.* If an approximate boundary is known, boundaries near this approximation can be favored by adding:

$$d = dist\ (\mathbf{x}_i, B)$$

to the cost measure. The dist operator measures the minimum distance of the new point $\mathbf{x}_i$ to the approximate boundary B.

4. *Estimates of the distance to the goal.* If the curve is reasonably linear, points near the goal may be favored by estimating h as $d(\mathbf{x}_i, \mathbf{x}_{goal})$, where d is a distance measure.

Specific implementations of these measures appear in [Ashkar and Modestino 1978; Lester et al. 1978].

4.4.2 Finding All the Boundaries

What if the objective is to find *all* boundaries in the image using heuristic search? In one system [Ramer 1975] Hueckel's operator (Chapter 3) is used to obtain

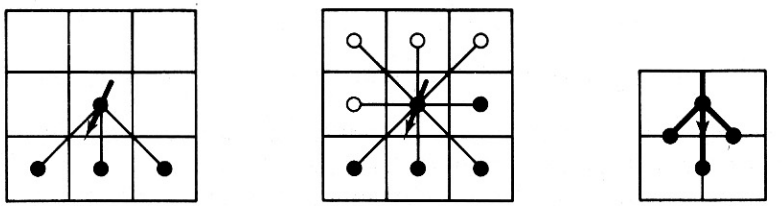


Fig. 4.11 Successor conventions in heuristic search (see text).

strokes, another name for the magnitude and direction of the local gray-level changes. Then these strokes are combined by heuristic search to form sequences of edge elements called *streaks*. Streaks are an intermediate organization which are used to assure a slightly broader coherence than is provided by the individual Hueckel edges. A bidirectional search is used with four eight-neighbors defined in front of the edge and four eight-neighbors behind the edge, as shown in Fig. 4.11b. The search algorithm is as follows:

1. Scan the stroke (edge) array for the most prominent edge.
2. Search in front of the edge until no more successors exist (i.e., a gap is encountered).
3. Search behind the edge until no more predecessors exist.
4. If the bidirectional search generates a path of 3 or more strokes, the path is a streak. Store it in a streak list and go to step 1.

Strokes that are part of a streak cannot be reused; they are marked when used and subsequently skipped.

There are other heuristic procedures for pruning the streaks to retain only *prime streaks*. These are shown in Fig. 4.12. They are essentially similar to the re-

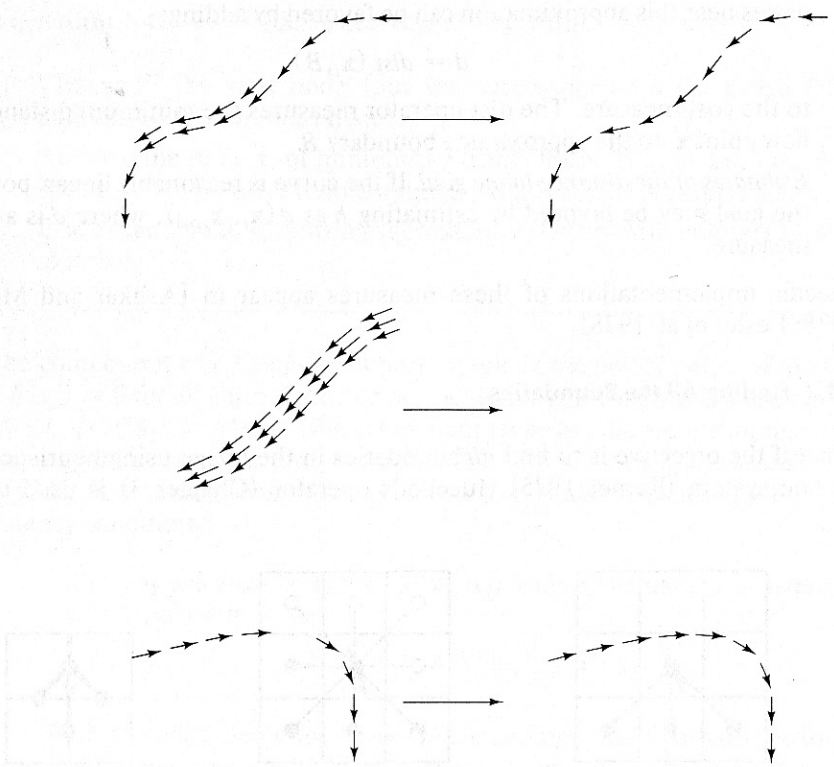


Fig. 4.12 Operations in the creation of prime streaks.

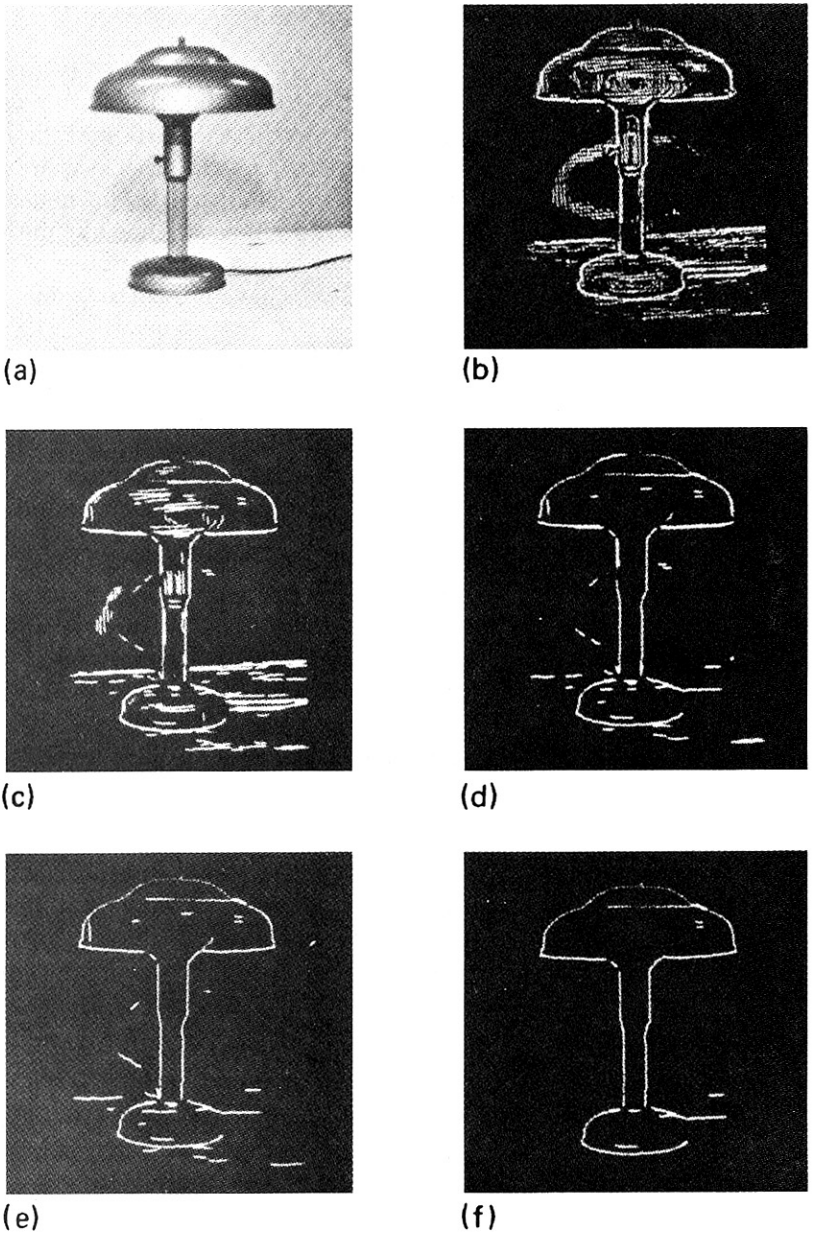


Fig. 4.13 Ramer's results.

laxation operations described in Section 3.3.5. The resultant streaks must still be analyzed to determine the objects they represent. Nevertheless, this method represents a cogent attempt to organize bottom-up edge following in an image. Fig. 4.13 shows an example of Ramer's technique.

4.4.3 Alternatives to the A Algorithm

The primary disadvantage with the heuristic search method is that the algorithm must keep track of a set of current best paths (nodes), and this set may become very large. These nodes represent tip nodes for the portion of the tree of possible paths that has been already examined. Also, since all the costs are nonnegative, a good path may eventually look expensive compared to tip nodes near the start node. Thus, paths from these newer nodes will be extended by the algorithm even though, from a practical standpoint, they are unlikely. Because of these disadvantages, other less rigorous search procedures have proven to be more practical, five of which are described below.

Pruning the Tree of Alternatives

At various points in the algorithm the tip nodes on the OPEN list can be pruned in some way. For example, paths that are short or have a high cost per unit length can be discriminated against. This pruning operation can be carried out whenever the number of alternative tip nodes exceeds some bound.

Modified Depth-First Search

Depth-first search is a meaningful concept if the search space is structured as a tree. Depth-first search means always evaluating the most recent expanded son. This type of search is performed if the OPEN list is structured as a stack in the A algorithm and the top node is always evaluated next. Modifications to this method use an evaluation function f to rate the successor nodes and expand the best of these. Practical examples can be seen in [Ballard and Sklansky 1976; Wechsler and Sklansky 1977; Persoon 1976].

Least Maximum Cost

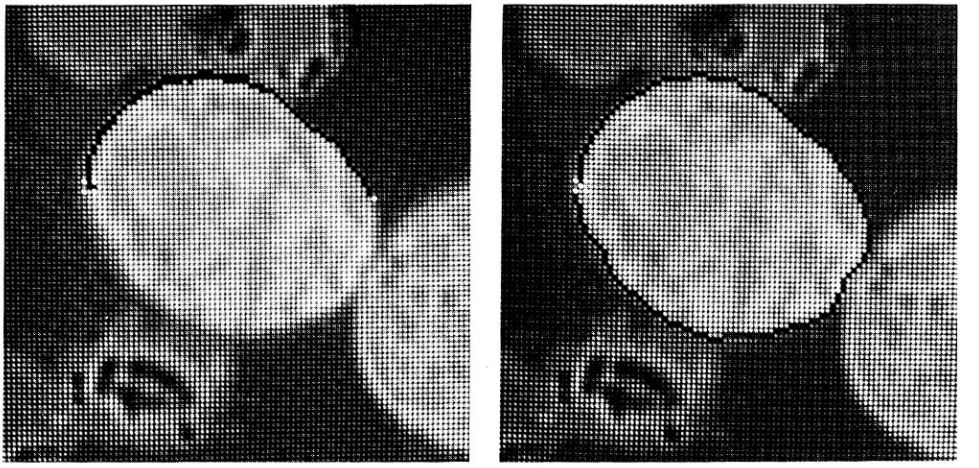
In this elegant idea [Lester 1978], only the maximum-cost arc of each path is kept as an estimate of g. This is like finding a mountain pass at minimum altitude. The advantage is that g does not build up continuously with depth in the search tree, so that good paths may be followed for a long time. This technique has been applied to finding the boundaries of blood cells in optical microscope images. Some results are shown in Fig. 4.14.

Branch and Bound

The crux of this method is to have some upper bound on the cost of the path [Chien and Fu 1974]. This may be known beforehand or may be computed by actually generating a path between the desired end points. Also, the evaluation function must be monotonically increasing with the length of the path. With these conditions we start generating paths, excluding partial paths when they exceed the current bound.

Modified Heuristic Search

Sometimes an evaluation function that assigns negative costs leads to good results. Thus good paths keep getting better with respect to the evaluation function, avoiding the problem of having to look at all paths near the starting point.



(a) (b)

Fig. 4.14 Using least maximum cost in heuristic search to find cell boundaries in microscope images. (a) A stage in the search process. (b) The completed boundary.

However, the price paid is the sacrifice of the mathematical guarantee of finding the least-cost path. This could be reflected in unsatisfactory boundaries. This method has been used in cineangiograms with satisfactory results [Ashkar and Modestino 1978].

4.5 EDGE FOLLOWING AS DYNAMIC PROGRAMMING

4.5.1 Dynamic Programming

Dynamic programming [Bellman and Dreyfus 1962] is a technique for solving optimization problems when not all variables in the evaluation function are interrelated simultaneously. Consider the problem

$$\max_{x_i} h(x_1, x_2, x_3, x_4) \tag{4.8}$$

If nothing is known about h, the only technique that guarantees a global maximum is exhaustive enumeration of all combinations of discrete values of $x_1, \ldots, x_4$. Suppose that

$$h(\cdot) = h_1(x_1, x_2) + h_2(x_2, x_3) + h_3(x_3, x_4) \tag{4.9}$$

x_1 only depends on x_2 in h_1. Maximize over x_1 in h_1 and tabulate the best value of $h_1(x_1, x_2)$ for each x_2:

$$f_1(x_2) = \max_{x_1} h_1(x_1, x_2) \tag{4.10}$$

Since the values of h_2 and h_3 do not depend on x_1, they need not be considered at