COMPUTER GRAPHICS AND IMAGE PROCESSING 17, 281–289 (1981)

NOTE

A Fast Algorithm for Finding Lines in Pictures

D. HERBISON-EVANS

Basser Department of Computer Science, University of Sydney, N.S.W., 2006, Australia

Received August 13, 1979; revised January 7, 1981.

A fast algorithm is presented for finding the darkest line of given length, L, from an array of m by n gray values sampled from a picture. The number of operations required is $k \cdot m \cdot n \cdot \log_2 L$, where k depends on the type of neighborhood used. A generalization is described for lines whose length is not a power of 2. The restrictions imposed by the suppression of self backtracking are discussed. The application to contour, edge, and skeleton finding is described.

1. INTRODUCTION

This algorithm addresses itself to the problem of finding a set of L sequentially adjacent grid points from a rectangular array of such points such that the total (or average) darkness of these points is a maximum with respect to all such sets. The picture is assumed to have been sampled at the grid points, and a grayness value stored for each point. Then the darkest line will have the greatest sum of these gray levels over the points in the line.

2. LINEFINDING

Previous authors on this subject have pointed out the sufficiency of serial optimization in this process [1, 2].

Thus if the total darkness of the darkest line ending at point $\mathbf{x}_k$ is the sum of the gray levels of the points on the line,

$$f_k(\mathbf{x}_k) = \sum_{i=1}^{k} g(\mathbf{x}_i),$$

then the darkest line ending at x_{k+1} is

$$f_{k+1}(\mathbf{x}_{k+1}) = \max_{\text{legal } \mathbf{x}_{k+1}} (f_k(\mathbf{x}_k) + g(\mathbf{x}_{k+1})),$$

where the legal $\mathbf{x}_{k+1}$ are neighboring points of $\mathbf{x}_k$.

The neighboring points of a given point are normally taken to be either

(a) the four points directly above, below, to the left and to the right of the given point (a single step Rook's move in chess); or

(b) the four points diagonally adjacent to a given point (a single step Bishop's move); or

(c) the eight points surrounding the given point (a single step Queen's move).

It does not matter which definition is taken, providing it is applied consistently in the algorithm. The resulting lines will differ, of course, if only because the diagonal

0146-664X/81/110281-09$2.00/0

steps in Queen's and Bishop's moves cover twice as much ground as the Rook's move steps.

3. FAST LINEFINDING

Given an $m \times n$ point array of gray levels (e.g., Fig. 1), the first step is to take each point in turn and to scan its legal neighbors. The darkest pair which can be made using the given point as the head and the darkest neighbor as the tail is constructed. A table to show the total darkness of each of the $m \times n$ pairs is constructed (e.g., Fig. 2, where the two legal neighbors are to the right and below). Another table to show which neighbor is the darkest is constructed (e.g., Fig. 3).

These tables can now be used to construct the darkest quadruple that has a head at each of the $m \times n$ picture points. For each point in turn, the legal neighbors of its darkest neighbor are scanned. The darkest pair is sought with a head at a neighboring point to the tail of the darkest pair with a head at the given point. For example, in Fig. 2, consider the construction of the darkest quadruple with a head at point (2, 2). Its darkest neighbor is point (2, 3). The darkest pair that is a neighbor to point (2, 3) is at (3, 3) in Fig. 2. Figure 3 shows that it is composed of points (3, 3) and (4, 3). Then the darkest quadruple from point (2, 2) is (2, 2) (2, 3) (3, 3) (4, 3) with a total gray value of 6 (sum of elements (2, 2) and (3, 3) of Fig. 2). A table is made of

0	1	1	0	0
1	2	1	0	0
0	1	1	0	0
0	1	2	1	0

FIG. 1. Gray levels at points in a small picture.

1	3	2	0	0
3	3	2	0	0
1	2	3	1	0
1	3	3	1	0

FIG. 2. Gray Level sums of darkest pair that can be made using each point and either the point to the right or the one below it.

12	22	23	15	25
22	23	33	25	35
32	33	43	44	45
42	43	44	45	X

FIG. 3. Location of darkest neighbor to each point (neighborhood consists of elements to left and below).

22	32	33	25	35
23	33	43	35	45
33	43	44	55	X
43	44	45	X	X

FIG. 4. Location of head of darkest pair that is a legal neighbor to the tail of the darkest pair from each point.

the gray values of the darkest quadruple that has a head at each point. Another table is constructed which shows which pairs were joined to make these quadruples. It must give the array element that is the head of the darkest pair that is attached to the tail of the darkest pair from each point, e.g., Fig. 4. The same procedure can then be used to link quadruples to give the set of darkest octuples that have a head at each point in the array.

Thus the darkest line of length $L = 2^p$ can be obtained by inspecting the array of total darkest values of the L-tuples after p iterations. This gives the head point of the darkest line.

The line itself can be obtained by backtracking appropriately down the position tables for the various iterations from this head. The position of the point that is "q" steps from the head of the line is obtained by expressing q as a binary number and initializing an array pointer to point to the head. The binary representation of q is scanned from most significant bit downward, and the presence of a 1 bit at some position "t" in this representation indicates that the array pointer must be replaced by the value at the position to which it points in the position array that was computed at iteration t. For example, the penultimate point on the line ($q = 1$) is the point referred to in the array of position of pair partners at the first iteration

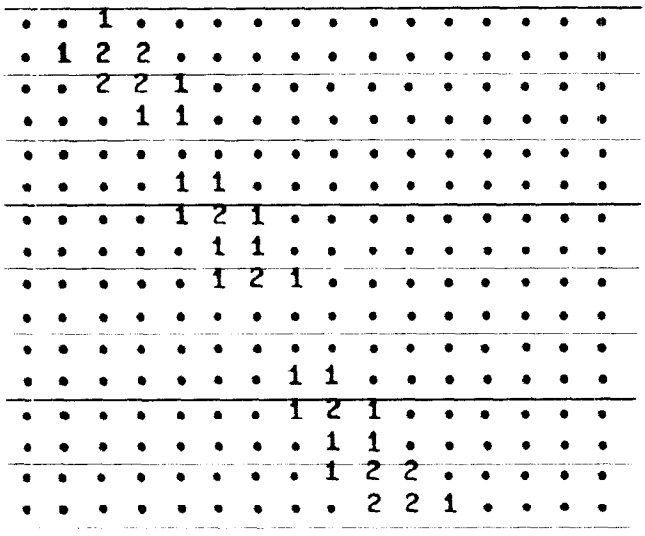

FIG. 5. Picture containing noisy line used in subsequent figures, showing gray values at square grid points (· has been printed for zero for greater clarity).

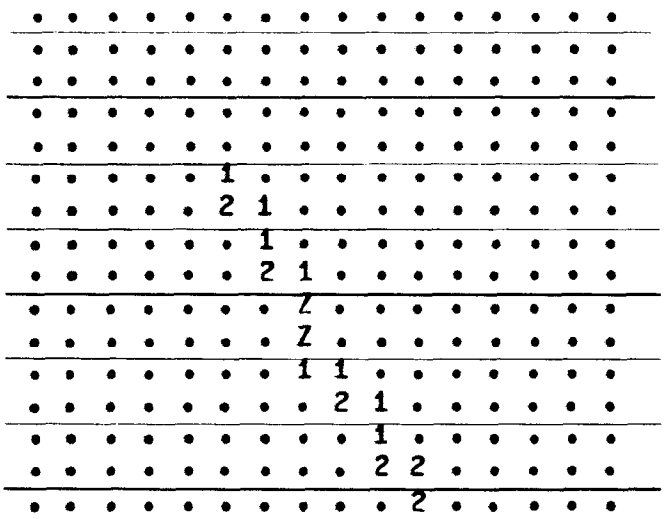

FIG. 6. Line of length 16 found using fast algorithm with Rook's move and legal moves in 4th quadrant (total Gray 20) (Z has been printed for zero on the found line for greater clarity).

($t = 1$), the first iteration corresponding to the units bit in the binary representation of q.

Although the method uses pairs at the first stage, the method is global in the sense that noise and gaps in the line do not unduly impede finding a line (e.g., see Figs. 5 and 6).

Furthermore, the method simultaneously finds the darkest lines of given length that end at every point in the picture, and if all those darker than some threshold are required, they can be obtained by inspection of the final tables (e.g., see Fig. 13).

4. COMPUTATION DEMANDS

The number of operations used in finding the darkest line of length $L = 2^p$ in an array of $m \times n$ gray values, where each point is allowed "a" neighbors, can be obtained by observing that at each of the p iterations, for each of the $m \times n$ points, "a" gray values are compared, each being obtained by an indirect reference from the position table in the total tuple table of gray values of the previous iteration. Thus $a \cdot p \cdot m \cdot n$ gray totals are indirectly referenced and then compared, and $p \cdot m \cdot n$ pairs of values are summed. For storage economy, the total gray levels of the k-tuples before the r^{th} iteration ($k = 2^{r-1}$) can be overwritten by the gray levels of the $2k$-tuples at the end of that iteration, so that only the table and a buffer for it are required, giving a storage requirement of $2 \cdot m \cdot n$ gray levels. The position table of every iteration must be saved for final backtracking to find the whole line, so that storage for $p \cdot m \cdot n$ two-dimensional addresses is also required.

5. GENERALIZATION

The method is not restricted to finding darkest line lengths that are a power of 2. If the length L is expressed in binary, then initially the iterations are performed as appropriate for a line whose length is the highest power of two, say "s," which is smaller than L (i.e., $2^s \leq L < 2^{s+1}$). Then the bits representing the value of L are

scanned from most significant bit downward. For a nonzero bit that is at position "t" in the representation, the lengths of the $m \cdot n$ darkest lines are lengthened by notionally adding a line of length 2^t by seeking the darkest neighboring 2^t-tuple from the arrays of the tth initial iteration, and creating a table of the resulting total gray values and the positions of the neighbors chosen. Note that for this process, all the "s" gray level total tuple tables need to be retained, and only the last one can be overwritten. In the worst case for a line of length $L = 2^{s+1} - 1$, $2 \cdot a \cdot s \cdot m \cdot n$ gray levels are referenced, $2 \cdot s \cdot m \cdot n$ values are summed, $s \cdot m \cdot n$ gray level values must be stored, but only $s \cdot m \cdot n$ two-dimensional addresses need be retained.

The order in which the pieces are summed is not unique. Only $2 \cdot m \cdot n$ gray level values need be stored if the values for the line are summed as created, the opposite order to that above.

For some values of L, this breakdown into the sum of powers of 2 is not optimum, e.g., $L = 31$. The problem is analogous to that of finding the best way to compute x^L.

A further problem with either method is that by adding small pieces of line to large pieces, the found line will be more susceptible to diversion by gaps and noise. The composition of a line from two equal sublines at each stage is optimum in this respect, and is only fully applicable when L is a power of two.

6. PROBLEMS

There is a price for using this fast algorithm rather than the sequential scheme of Montanari [2] or Martelli [1]. This price is that curvature or other shape information cannot easily be included in the choice of neighbor at each stage.

The inclusion of such information is the normal method of avoiding the generation of lines that backtrack along themselves. This backtracking can be avoided in the fast algorithm by restricting the neighboring points that are searched at each

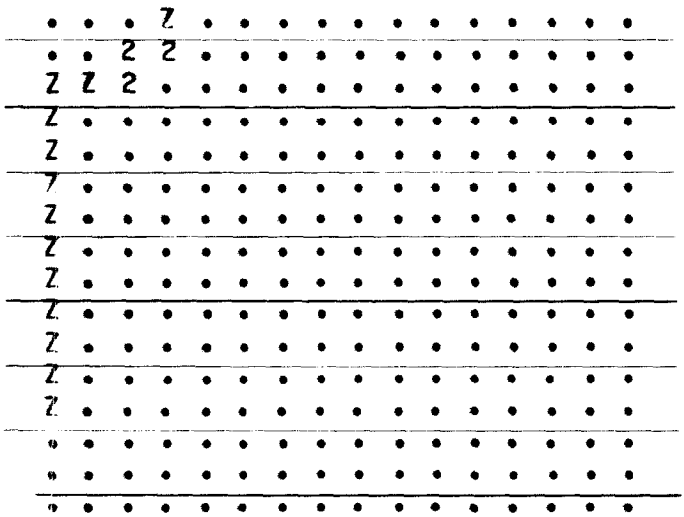

FIG. 7. Rook's move and legal moves in 3rd quadrant. (Total gray 6.)

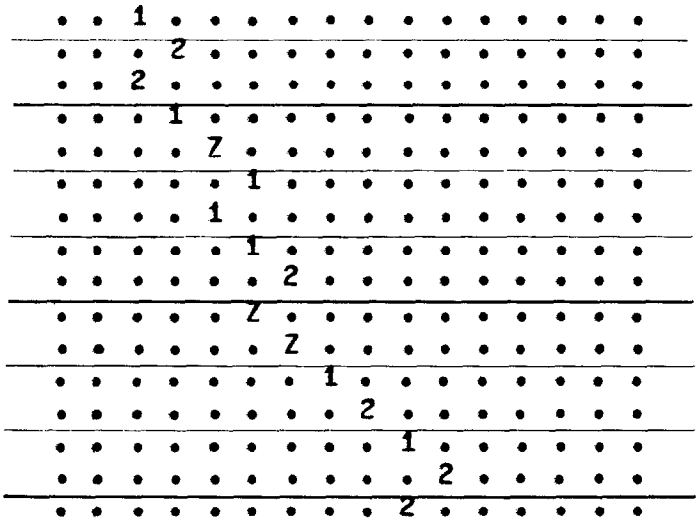

FIG. 8. Bishop's move, legal moves above current position. (Total gray 19.)

stage to an adjacent set of only half the actual neighbors, so that the line is only allowed to grow monotonically in a restricted range of directions.

For Rook's move neighbors, for instance, this can be achieved by looking at values only below and to the right of a given point. This restricts the lines that are found to have all their direction transitions in the fourth quadrant. Then the extremes of possible lines that can be found are horizontally to the right of, and vertically below, the head.

Thus this fast algorithm cannot discover curved lines where the curvature takes the slope of the line through more than 90°. If the orientation of the required line is not known initially, then the procedure must be applied with the quadrant of search

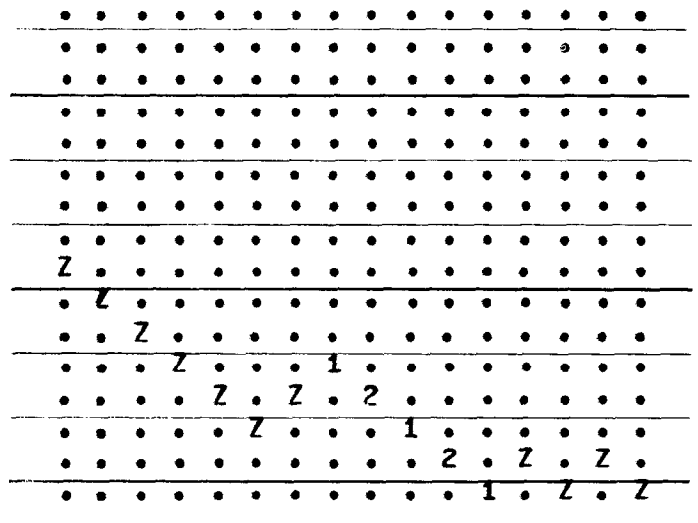

FIG. 9. Bishop's move, legal moves to left of current position. (Total gray 7.)

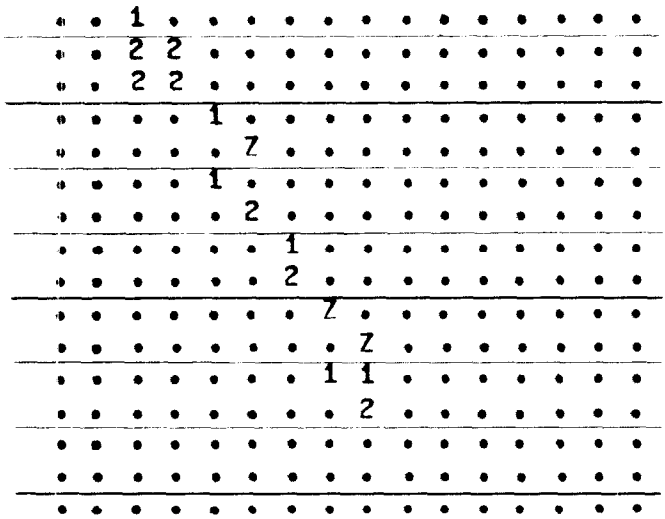

FIG. 10. Queen's move and legal moves in range 45° to 180°. (Total gray 20.)

in all possible positions, and darkest line taken from the results. This is illustrated in Figs. 5, 6, and 7.

Simlarly, Bishop's move neighbors must be restricted to be either above, below, to the left or to the right of the given position, and this is illustrated in Figs. 8 and 9.

For Queen's move neighbors, a set of four adjacent neighbors occupies 135°, so that possible lines must lie in a quadrant and a half. This is illustrated in Figs. 10, 11, and 12.

The effect of noise on the linefinder is difficult to visualize. For instance, a critical gap in a line can make the algorithm fail to find a line when processing in one direction, but it may succeed when processing in the opposite direction. For this

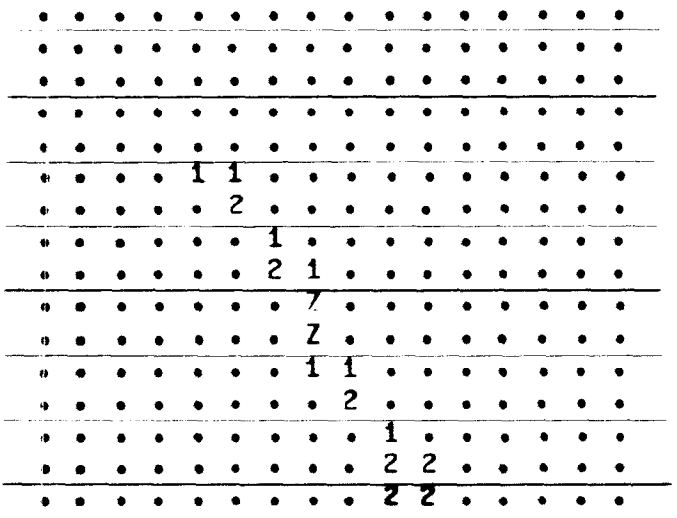

FIG. 11. Queen's move and legal moves in range 90° to 225°. (Total gray 21.)

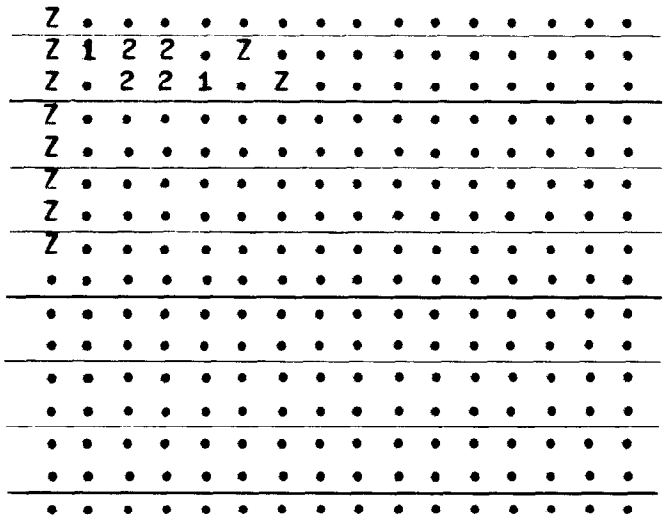

FIG. 12. Queen's move and legal moves in range 135° to 270°. (Total gray 10.)

reason all four or eight directions must be searched if the noise amplitude is comparable with the mean grayness of a pixel on the sought line. If the noise is bigger than this, the darkest line cannot necessarily be found.

The edges of the picture also need some attention. The simplest way of coping with them is notionally to embed the picture in a further ring of pixels of zero grayness. Then the program can just skip the access and addition of gray values when a point outside the picture is called for.

There is a problem if, at any stage, more than one neighbor is found with the same value. Then ideally a program should investigate all these choices, but this can lead

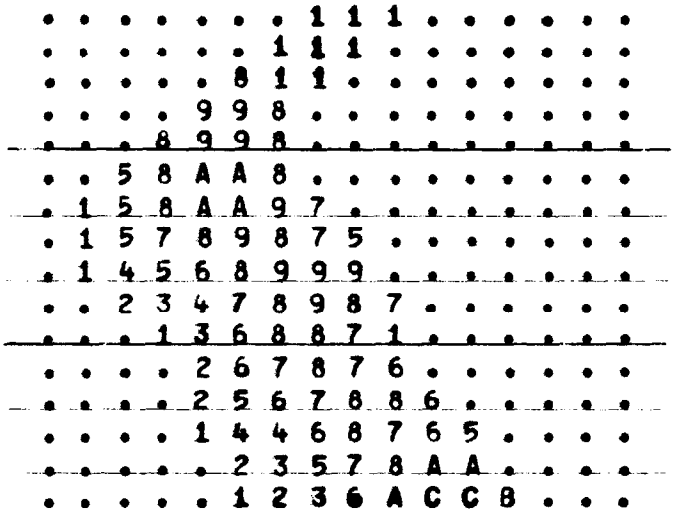

FIG. 13. Total gray values of darkest lines of length 8 with tails at each point in the picture using Rook's move and legal moves in 2nd quadrant, using the hexadecimal system.

to an unacceptable time expansion. The other extreme action is arbitarily to take any one of the equal alternatives. The rationale for this is that if some arbitarily small amount of noise in the picture were distributed differently on different attempts to find the darkest line, then different elements of the equal set would be the largest, and so the program should work with any of them.

In practice, it is not very important. If the line passes through an extended blob, the formal routing of the line through the blob is of limited interest.

7. PREPROCESSING

A number of other problems can be transformed into the darkest line problem by suitable preprocessing of the gray values, notably: contouring, boundary finding, and skeleton finding [1].

To perform contouring, a copy of the picture array is generated with all elements zero except those whose corresponding picture element is within a certain tolerance of the desired contour level, or else opposite neighboring picture elements span the desired level. The nonzero elements in the copy are set either to an arbitrary positive value or to the values of the corresponding picture elements (assumed to be positive).

Boundaries of areas of nearly constant brightness can be found by using the linefinder on a processed copy of the picture. The copy of the picture is generated with each value replaced by the maximum modulus of the difference in values between opposite neighbors. Effectively, the copy contains an approximation to the modulus of the gradient of the original picture.

Skeletons of areas of nearly constant brightness can also be found by using the linefinder on a processed copy of the picture. The processing required for this case is to let each point in the copy be the negative modulus of the minimum difference between its opposite picture element neighbors. Then the linefinder will find lines of least slope, i.e., skeletons.

If no noise is present in the picture and the sampling interval is sufficiently small, special algorithms can be employed for each of contour, boundary, and skeleton finding that work iteratively from each element to its neighbors [3]. If the data is noisy and/or badly sampled, these algorithms can be paralyzed by occasional unfortunate values. In order to cope with such problems, a more global averaging type of approach is required. Thus the local operations described above for preprocessing for contour, edge, and skeleton finding should be altered to use some sort of averaging of the neighborhood values on each side of the point being processed, and then a global type of linefinder should be used (such as the one herein described).

REFERENCES

1. A. Martelli, An application of heuristic search methods to edge and contour detection, *Comm. ACM* **19**, 1976, 73–83.
2. U. Montanari, On the optimal detection of curves in noisy pictures, *Comm. ACM* **14**, 1971, 355–345.
3. A. Rosenfeld, *Picture Processing by Computer*, Academic Press, New York, 1969.