ELSEVIER

Information Processing Letters 58 (1996) 129-133

Information Processing Letters

A unified linear-time algorithm for computing distance maps

Tomio Hirata [1]

Faculty of Engineering, Nagoya University, Chikusaku, Nagoya, 464-01 Japan

Received 25 July 1995
Communicated by K. Ikeda

Keywords: Algorithms; Computer vision; Euclidean distance transform; Image processing; Matrix searching

1. Introduction

Distance maps of binary images contain, for each pixel, the distance between that pixel and the pixel of value 0 closest to it. They have important uses in computer vision, pattern recognition, morphology and robotics [10]. Many algorithms have been proposed for computing distance maps for a variety of distances such as the L_1, L_∞, octagonal and Euclidean distances [5,11-13].

Recently, Kolounzakis and Kutsulakos [9] gave an $O(N^2 \log N)$ time algorithm for computing accurate Euclidean distance maps. It computes a distance map column by column, using a divide and conquer approach to each column. They also pointed out that their algorithm is available for other distances such as the L_1 distance. Dividing rows and columns alternately, Chen and Chuang [4] reduced the time complexity to $O(N^2)$ which is optimal. Breu et al. [2] also gave an $O(N^2)$ time algorithm which is based on the construction of the Voronoi diagram.

In this paper we give a simple unified algorithm for computing distance maps. The algorithm runs in $O(N^2)$ time for an input of $N \times N$ binary image. It was first developed for the Euclidean distance in [8]. We show that it is also available for a wide class of distances. The class contains most distances that appear in machine vision applications, such as the Euclidean, city block (or L_1), chessboard (or L_∞), octagonal and chamfer distances. A parallel version of our algorithm runs in $O(N^2/p)$ time with p ($1 \leqslant p \leqslant N$) processors, which is superior to the previous $O(N^2/p + N \log p)$ of [4] when p is of $O(N)$. Fujiwara et al. [6] also obtained a parallel version for the EREW PRAM model that runs in $O(\log N)$ time with $N^2/\log N$ processors.

Some definitions are given in Section 2. In Section 3 we review the algorithm of [8] to make our presentation self-contained. In Section 4 we show that the algorithm is available for a wide class of distances.

2. Preliminaries

Let $\mathbf{B} = \{b_{ij}\}$ be an $N \times N$ binary image. We denote by (i, j) the element in the ith row and the jth column. If $b_{ij} = 0$ we call it a 0-element and if $b_{ij} = 1$ a 1-element. The Euclidean distance map $\mathbf{D} = \{d_{ij}\}$ of a binary image $\mathbf{B} = \{b_{ij}\}$ is defined by

$$d_{ij} = \min_{1 \leqslant p,q \leqslant N} \{\sqrt{(i-p)^2 + (j-q)^2} \mid b_{pq} = 0\}.$$

[1] Email: hirata@nuee.nagoya-u.ac.jp.

PII S0020-0190(96)00049-X

In picture processing, the weighted 4-neighbor distance is used quite frequently. The distance $d(p_1,p_2)$ between $p_1 = (x_1,y_1)$ and $p_2 = (x_2,y_2)$ in the xy plane is defined by

$$d(p_1,p_2) = \begin{cases} w_0|x_1 - x_2| + w_1|y_1 - y_2| \\ \quad \text{if } |x_1 - x_2| > |y_1 - y_2|, \\ w_1|x_1 - x_2| + w_0|y_1 - y_2| \\ \quad \text{otherwise,} \end{cases}$$

where $w_0 \geqslant w_1 \geqslant 0$. When $w_0 = w_1 = 1$, the distance is called *city block* (or *4-neighbor*), and when $w_0 = 1$, $w_1 = 0$, *chessboard* (or *8-neighbor*). When $w_0 = 1$, $w_1 = \sqrt{2} - 1$, it is called *chamfer*, and when $w_0 = 1$, $w_1 = 1/\sqrt{2} - 1 + \sqrt{\sqrt{2} - 1} \approx 0.351$, *optimal chamfer* [3].

The Euclidean, city block, and chessboard distances are coincide with the L_2, L_1 and L_∞ distances, respectively. Furthermore, the *octagonal* distance is defined by

$$d(p_1,p_2) = \max\{|x_1 - x_2|, |y_1 - y_2|, \\ 2(|x_1 - x_2| + |y_1 - y_2|)/3\}.$$

3. Distance transform algorithm

In this section we describe the algorithm for the Euclidean distance transformation originally presented in [8].

3.1. Basic algorithm

We first give a basic form of the algorithm which has the same framework as that of [4].

T1. For each column j of $\mathbf{B} = \{b_{ij}\}$, we compute a map $\mathbf{G} = \{g_{ij}\}$ such that

$$g_{ij} = \min_{1 \leqslant p \leqslant N} \{|i - p| \mid b_{pj} = 0\},$$

where $g_{ij} = \infty (1 \leqslant i \leqslant N)$ for a column j with no 0-element.

T2. Scanning each row i of $\mathbf{G} = \{g_{ij}\}$, we compute $\mathbf{D}' = \{d'_{ij}\}$ such that

$$d'_{ij} = \min_{1 \leqslant q \leqslant N} \{(j - q)^2 + g_{iq}^2\}.$$

It is clear that the value of d'_{ij} is the square of the distance value from an element (i,j) to the nearest 0-element in $\mathbf{B}$. It is also clear that T1 can be done in $O(N^2)$ time, and T2 in $O(N^3)$ time.

3.2. Algorithm for T2

Here we give an efficient algorithm for T2. Consider a scan of the ith row in T2. The square of the minimum distance from a 0-element of the kth column to (i,j) is expressed as $(j - k)^2 + g_{ik}^2$. We denote this value as a function of j, that is, $f_k^i(j) = (j - k)^2 + g_{ik}^2$. Then $d'_{ij} = \min_{1 \leqslant k \leqslant N} f_k^i(j)$. Therefore, what we have to do for each row is finding the lower envelope $\min_{1 \leqslant k \leqslant N} f_k^i(x)$ of the set of functions $F_N^i = \{f_k^i(x) \mid 1 \leqslant k \leqslant N\}$. From now on, we will use $f_k(x)$ and F_N instead of $f_k^i(x)$ and F_N^i when no confusion arise. Let $L_{\text{env}}(F_l)$ be the set of functions that give the lower envelope of F_l, that is, $L_{\text{env}}(F_l) = \{f_i(x) \mid \exists x_0 \; f_i(x_0) = \min_{1 \leqslant k \leqslant l} f_k(x_0)\}$. We denote by x_{ij} the abscissa of the intersection of $f_i(x)$ and $f_j(x)$. To compute $L_{\text{env}}(F_N)$, we initially set $L_{\text{env}}(F_2) = \{f_1, f_2\}$ (For simplicity we assume that $g_{i1} \neq \infty$ and $g_{i2} \neq \infty$.), and we compute $L_{\text{env}}(F_3), L_{\text{env}}(F_4), \ldots$, successively. To this end we use a stack. The following is the algorithm.

Efficient algorithm for T2

```
1:  Repeat 2–19 for each row $i$ of $\mathbf{G}$.
2:  begin
3:    Let $s$ be an empty stack.
4:    push($s$, 1);  {Push $f_1(x)$ into $s$.}
5:    push($s$, 2);  {Push $f_2(x)$ into $s$.}
6:    {Let $t' < t$ be the two top elements of $s$.}
7:    for $j$ := 3 to $N$ do
8:      if $g_{ij} \neq \infty$ then
9:        begin
10:         while $x_{tj} < x_{t't}$ do pop($s$);
11:           push($s$, $j$)  {Push $f_j(x)$ into $s$.}
12:       end;
13:     while $N < x_{t't}$ do pop($s$);
14:     for $j$ := $N$ downto 1 do
15:       begin
16:         if $j < x_{t't}$ then pop($s$);
17:         $d'_{ij}$ := $f_t(j)$
18:       end;
19: end
```

Correctness comes from the observation that any two functions of F_N intersect exactly once. We can view the first task of the above algorithm (lines 2–12) as that of finding the convex hull of N points [6]. Let F'_N be the set $\{f'_k(x) = -2kx + k^2 + g^2_{ik} \mid f_k(x) \in F_N\}$. It is easy to see that the task of finding the lower envelope of F_N is reduced to that of finding the lower envelope of F'_N. Using the standard dual transformation from lines to points, we can view the latter task as that of finding the convex hull of the N points. Also we can view the task as a matrix searching problem: finding the minimum entry in each row of a matrix. Consider an $N \times N$ matrix $A = (a_{ij})$ such that $a_{jk} = f^i_k(j)$. Then $d'_{ij} = \min_{1 \leqslant k \leqslant N} a_{jk}$. Thus our task is reduced to a matrix searching for A. The matrix searching can be done in $O(N)$ time if A is totally monotone [1]. Total monotonicity of A follows from the one-intersection condition of $f^i_k(x)$. Note that the above algorithm is much simpler than a matrix-searching algorithm.

Once $L_{\text{env}}(F_N)$ is stored in the stack, we assign a value of a function in the stack to each element of the row (lines 13–18). It is clear that the above algorithm is of $O(N^2)$ time. Taking the time complexity of T1 into account, we can compute the Euclidean distance map in $O(N^2)$ time.

The above algorithm can be easily extended to the Euclidean distance transformation of a d-dimensional binary image ($d \geqslant 3$) and the time complexity is $O(N^d)$. Since the above algorithm works on one column (row) at a time, it can be easily parallelized. Since column scans are independent each other, with p ($1 \leqslant p \leqslant N$) processors the column scans can be done in parallel in $O(N^2/p)$ time. Similar observation holds for row scans. Therefore, with p ($1 \leqslant p \leqslant N$) processors, a parallel version of the above algorithm runs in $O(N^2/p)$ time, which is superior to the previous $O(N^2/p + N \log p)$ of [4] when p is of $O(N)$.

4. Class of distances

The algorithm of the previous section first scans all columns of an input binary image to compute a 1-dimensional distance map for each column and then scans all rows of this intermediate result to compute the Euclidean distance maps. This approach applies not only to the Euclidean distance. In fact, Paglieroni [10] proposed a unified transform architecture based on parallel row scanning followed by parallel column scanning and showed that it can be applied to any distance function $d(p_1, p_2)$ for which there exists a function $f(x, y)$ such that (i) $d(p_1, p_2) = f(|x_1 - x_2|, |y_1 - y_2|)$, and (ii) $\forall y\ |x_1| < |x_2| \Longrightarrow f(|x_1|, |y|) \leqslant f(|x_2|, |y|)$, and $\forall x\ |y_1| < |y_2| \Longrightarrow f(|x|, |y_1|) \leqslant f(|x|, |y_2|)$. Most distances used in machine vision applications such as the 4-neighbor and 8-neighbor distances belong to this class. Note that the distance in this class would not necessarily be a metric because the triangle inequality does not always hold.

The algorithm for T2 uses the fact that any two functions of F_N intersect exactly once. It does not work for all the distances in the Paglieroni's class. Consider the distance function $d(p_1, p_2) = |x_1 - x_2||y_1 - y_2|$, which, shown as an example in [10], satisfies the Paglieroni's conditions but not the triangle inequality. Then $f^i_k(x) = g_{ik}|x|$, and two functions intersect twice if the values g_{ik} are different. In the rest of this section we consider a class of distances to which the algorithm of the previous section applies.

We assume that the distance function satisfies the Paglieroni's condition and the triangle inequality. Furthermore we add the following condition: for successive three points p_1, p_2, p_3 on a line,

$$d(p_1, p_2) + d(p_2, p_3) = d(p_1, p_3).$$

This condition with the triangle inequality assures us that the quadrangle inequality holds, that is, the sum of length of diagonals of a quadrangle is not shorter than that of a pair of its opposite edges.

Lemma 1. *Let $a = (x_a, y_a)$ and $b = (x_b, y_b)$ be points not on the x-axis and $p' = (x_{p'}, 0)$ be a point on the x-axis such that $x_a < x_b$ and $d(a, p') = d(b, p')$. Then, for a point $p = (x_p, 0)$ on the x-axis, if $x_{p'} < x_p$, $d(a, p) \geqslant d(b, p)$, and if $x_{p'} > x_p$, $d(a, p) \leqslant d(b, p)$.*

Proof. We first consider the case $y_a > 0$ and $y_b > 0$ (or $y_a < 0$ and $y_b < 0$). Assume that $x_{p'} < x_p$. Let c be the intersection of the line segments $\overline{ap}$ and $\overline{bp'}$. (If such an intersection does not exist, then $|y_a| > |y_b|$ and $|x_p - x_a| > |x_p - x_b|$. By Paglieroni's condition (ii), $d(a, p) \geqslant d(b, p)$.) By the quadrangle inequality, we have $d(a, p) + d(b, p') \geqslant d(a, p') + d(b, p)$. But $d(a, p') = d(b, p')$ implies $d(a, p) \geqslant d(b, p)$. The case $x_{p'} > x_p$ is similar.

When $y_a > 0$ and $y_b < 0$, consider the point $b' = (x_b, -y_b)$. Paglieroni's condition (i) implies that $d(b', p') = d(b, p')$ and $d(b', p) = d(b, p)$. Therefore the above discussion applies. □

Lemma 1 guarantees that any two of the functions $f_i(x)$ in F_N have at most one intersecting point/part with each other. That is, assume that there are 0-elements at the points a and b and consider a scan of the ith row in T2, then the abscissa of the intersection of $f_{x_a}(x)$ and $f_{x_b}(x)$ corresponds p'. (If $f_{x_a}(x)$ and $f_{x_b}(x)$ overlap partially, we slightly translate one of them to find an intersecting point.) It is reasonable to assume that the intersection can be found in O(1) time. From the above discussion we have

Theorem 2. *If a distance satisfies the Paglieroni's conditions and the quadrangle inequality the distance map can be computed in* $O(N^2)$ *time.*

Paglieroni's condition (i) suggests that we concentrate ourselves to distances that come from a vector space norm.

A distance *is based on* (or *comes from*) *a vector space norm* if the distance is defined as $d(p_1, p_2) = \|p_1 - p_2\|$, where $\|\cdot\|$ denotes a norm in a vector space Note that a distance based on a vector space norm satisfies the axioms of metric. Since the condition on successive three points on a line naturally holds, the quadrangle inequality also holds. To satisfy Paglieroni's conditions we need a further condition.

A vector space norm is *x-axis* (*y-axis*) *symmetric* if its unit sphere is a symmetric figure with respect to the vertical (horizontal) line through its center. Since the unit sphere of a vector space norm is symmetric with respect to its center, an x-axis symmetric norm is also y-axis symmetric. Therefore, we call an x-axis symmetric norm an *axial symmetric* norm. If a distance is based on an axial symmetric norm, it depends on absolute values of differences of x-coordinates and y-coordinates, that is, $d(p_1, p_2) = \|(|x_1 - x_2|, |y_1 - y_2|)\|$.

Lemma 3. *Let* $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $p_2' = (x_2, y_2')$, $p_2'' = (x_2'', y_2)$ *be four points on a plane such that* $|y_2 - y_1| < |y_2' - y_1|$ *and* $|x_2 - x_1| < |x_2'' - x_1|$. *Then* $d(p_1, p_2) \leqslant d(p_1, p_2')$ *and* $d(p_1, p_2) \leqslant d(p_1, p_2'')$.

Proof. We show that $d(p_1, p_2) \leqslant d(p_1, p_2')$. The proof for $d(p_1, p_2) \leqslant d(p_1, p_2'')$ is similar. Consider a sphere S with radius $d(p_1, p_2')$ centered at p_1, that is, $S = \{p \mid \|p - p_1\| \leqslant d(p_1, p_2')\}$. We claim that S is a convex region. To observe this, we show that $ta + (1-t)b$ is in S for any $a, b \in S$ and t $(0 \leqslant t \leqslant 1)$. Since $a, b \in S$, $\|a - p_1\| \leqslant d(p_1, p_2')$ and $\|b - p_1\| \leqslant d(p_1, p_2')$. On the ground of the property of a norm, we have

$$\begin{aligned}
&\|ta + (1-t)b - p_1\| \\
&\quad = \|ta + (1-t)b - tp_1 - (1-t)p_1\| \\
&\quad = \|t(a - p_1) + (1-t)(b - p_1)\| \\
&\quad \leqslant \|t(a - p_1)\| + \|(1-t)(b - p_1)\| \\
&\quad = |t|\|(a - p_1)\| + |1-t|\|(b - p_1)\| \\
&\quad \leqslant |t|d(p1, p_2') + |1-t|d(p1, p_2') \\
&\quad = d(p_1, p_2').
\end{aligned}$$

Consequently $ta + (1-t)b \in S$, and thus S is convex.

Now consider the point $\bar{p}_2' = (x_2, 2y_1 - y_2')$. By the axial symmetry, $d(p_1, \bar{p}_2') = d(p_1, p_2')$, and thus $\bar{p}_2' \in S$. But $p_2 = tp_2' + (1-t)\bar{p}_2'$ with $t = (y_2 + y_2' - 2y_1)/(2y_2' - 2y_1)$. This implies $p_2 \in S$ and $d(p_1, p_2) \leqslant d(p_1, p_2')$. □

Lemma 3 implies that the distance $d(p_1, p_2)$ is non-decreasing in $|x_1 - x_2|$ and in $|y_1 - y_2|$, and thus a distance $d(p_1, p_2)$ based on an axial symmetric norm satisfies the Paglieroni's conditions.

Corollary 4. *If a distance is based on an axial symmetric norm the distance map can be computed in* $O(N^2)$ *time.*

The class of distances satisfying the condition of the above corollary is wide enough to contain the Euclidean, 4-neighbor, 8-neighbor, chamfer and octagonal distances.

Acknowledgements

We would like to thank Te. Asano and T. Tokuyama for their helpful comments. We also thank T. Masuzawa and A. Fujiwara for pointing out the work of [4]. Thanks are also due to the referees whose comments improve the exposition of the paper.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor and R. Wilber, Geometric applications of a matrix-searching algorithm, *Algorithmica* **2** (1987) 195–208.

[2] H. Breu, J. Gil, D. Kirkpatrick and M. Werman, Linear time Euclidean distance transform algorithms, *IEEE Trans. Pattern Analysis and Machine Intelligence* **17** (1995) 529–533.

[3] G. Borgefors, Distance transformations in digital images, *Comput. Graphics and Image Process.* **34** (1986) 344–371.

[4] L. Chen and H.Y.H. Chuang, A fast algorithm for Euclidean distance maps of a 2-D binary image, *Inform. Process. Lett.* **51** (1994) 25–29.

[5] P.E. Danielsson, Euclidean distance mapping, *Comput. Graphics and Image Process.* **14** (1980) 227–248.

[6] A. Fujiwara, T. Masuzawa and H. Fujiwara, A parallel algorithm for the Euclidean distance maps, *SIGAL Workshop of IPS* **AL-43** (1995).

[7] T. Hirata and T. Kato, An algorithm for Euclidean distance transformation, *SIGAL workshop of IPS* **AL-41** (1994), in Japanese.

[8] T. Kato, T. Hirata, T. Saito and K. Kise, An efficient algorithm for the Euclidean distance transformation, Draft, in Japanese.

[9] M.N. Kolountzakis and K.N. Kutulakos, Fast computation of Euclidean distance maps for binary images, *Inform. Process. Lett.* **43** (1992) 181–184.

[10] D.W. Paglieroni, A unified distance transform algorithm and architecture, *Machine Vision Appl.* **5** (1992) 47–55.

[11] A. Rosenfeld and J. Pfalts, Sequential operations in digital picture processing, *J. ACM* **13** (1966) 471–494.

[12] T. Saito and J. Toriwaki, Fast algorithms for *n*-dimensional Euclidean distance transformation, in: *Proc. 8th Scandinavian Conf. on Image Analysis*, Vol. II (1993) 747–754.

[13] H. Yamada, Complete Euclidean distance transformation by parallel operation, in: *Proc. 7th Internat. Conf. on Pattern Recognition*, Vol. 1, Montreal (1984) 69–71.