In Infor. Proc. Letters 43 (1992), 181-184

Fast Computation of the Euclidean Distance Map for Binary Images

Mihail N. Kolountzakis [1] [2], Kiriakos N. Kutulakos[2] [3]

Abstract
A simple algorithm is given for the computation of the Euclidean distance from the set of black points in a $N \times N$ black and white image, for all points in the image. The running time is $O(N^2 \log N)$ and $O(N)$ extra space is required. The algorithm is suitable for implementation on a parallel machine.

Keywords
Euclidean Distance Map - Parallel Algorithms - Computer Graphics - Pattern Recognition - Robotics - Image Processing.

1. Introduction

Consider a black and white $N \times N$ binary image: i.e., a two dimensional array where $a_{ij} = 0$ or 1, for $i, j = 0, \ldots, N-1$. The index i stands for the row, the index j for the column and $(0,0)$ is the upper left point in the image. We solve the problem of finding for each point (i,j) its Euclidean distance from the set of all black points $B = \{(i,j) : a_{ij} = 1\}$. In other words, we compute the array

$$d_{ij}^2 = \min_{(x,y) \in B} \left((i-x)^2 + (j-y)^2 \right), \quad \text{for all } i, j.$$

An algorithm is given which requires $O(N^2 \log N)$ time and $O(N)$ space (plus the space needed to store the result). The algorithm is suitable for implementation on a parallel machine.

The Euclidean distance map d_{ij} has important uses in computer vision, pattern recognition and robotics [2]. For instance, if the black points represent obstacles then d_{ij} tells us how far the point (i,j) is from these obstacles. This information is useful when one tries to move a robot in the free space (white points of the image) keeping it away from the obstacles.

It is much easier to compute the distance map for the L^1 distance. This can be done in optimal time $O(N^2)$, if one uses the fact that the L^1 distance between two points in the $N \times N$ grid is the length of the shortest path that joins them. However, there is no such interpretation for the Euclidean distance. One way around this problem is to approximate the Euclidean distance with distances that can be computed in the same way as the L^1 distance (Danielsson [2]). The only non-trivial algorithm for the computation of the exact Euclidean distance map that we are aware of is that of Yamada [3]. Yamada's algorithm performs N iterations of local transformations of the entire image. It is not clear whether Yamada's algorithm can be converted to an efficient (i.e. running in time close to $O(N^2)$) algorithm on a sequential computer.

2. The Algorithm

First note that it is sufficient to give an algorithm for the computation of

$$r_{ij}^2 = \min_{(x,y) \in B, y \geq j} \left((i-x)^2 + (j-y)^2 \right), \quad \text{for all } i, j.$$

Here r_{ij}^2 is the square of the distance of the point (i,j) from the part of B that is to the right of (i,j). Computing the distance from the right and from the left and taking the minimum of the two gives d_{ij}^2. So only the computation of r_{ij}^2 is described here.

[1] Department of Mathematics, Stanford University.

[2] Both authors were at the Image Analysis Laboratory, Department of Computer Science, University of Crete, Greece, when this work was carried out (1988).

[3] Computer Sciences Department, University of Wisconsin – Madison.

Let $B_j = B \cap \{(x,y) \ : \ y \geq j\}$ and let N_{ij} denote any nearest point to (i,j) in B_j. The following lemma is exploited by our algorithm.

Lemma 1: Let $P = (a,j)$, $Q = (b,j)$, $b < a$, be two points in the same column with Q above P. Let $N_P = (x,y)$, and $N_Q = (z,w)$. Then $z \leq x$, that is N_Q is above N_P. This holds for any choice of the nearest black points N_P and N_Q.

Proof: We have

$$(x-a)^2 + (y-j)^2 \leq (z-a)^2 + (w-j)^2$$

and

$$(z-b)^2 + (w-j)^2 \leq (x-b)^2 + (y-j)^2.$$

Adding the above inequalities gives $z \leq x$.

The array N_{ij} is computed column by column, moving from left to right. The values of r_{ij}^2 can be computed from N_{ij}. While computing N_{ij} for fixed j and for all i, we use the vector

$$V_i^{(j)} = \min\{k \ : \ k \geq j, \ (i,k) \in B\}.$$

If the above set is empty, we set $V_i^{(j)} = \infty$. Each vector $V_i^{(j+1)}$ is computed from $V_i^{(j)}$ as follows. For each i such that $a_{ij} = 1$, scan the i-th row to the right, starting from column $j+1$, until a black point is encountered, which will be $V_i^{(j+1)}$. If no black point is encountered up to the column $N-1$, set $V_i^{(j+1)} = \infty$. If $a_{ij} = 0$ then set $V_i^{(j+1)} = V_i^{(j)}$.

Only the current $V^{(j)}$ vector is stored. The computation for all i and j of $V_i^{(j)}$ takes $O(N^2)$ time, since each row is traversed to the right only once. So, while computing N_{ij} for any fixed column j, we may assume the vector $V^{(j)}$ is given.

It remains to give an $O(N \log N)$ time algorithm for computing N_{ij} for any fixed j. We have to find for each i a nearest point of (i,j) among the points in the vector $V^{(j)}$. (Notice that we identify $V_k^{(j)}$ with the point $(k, V_k^{(j)})$.) Doing this by exhaustive search results in an $O(N^2)$ algorithm. Instead, the following recursive algorithm is used.

Denote by $A(x_1, x_2, z_1, z_2)$ the algorithm below. This algorithm finds nearest black points for all points (i,j), $x_1 \leq i \leq x_2$ (j is fixed), considering only the points $V_k^{(j)}$, $z_1 \leq k \leq z_2$. The problem is solved by the call $A(0, N-1, 0, N-1)$.

Algorithm $A(x_1, x_2, z_1, z_2)$

1. Set $x_0 = \lfloor \frac{1}{2}(x_1 + x_2) \rfloor$.
2. Find $N_{x_0 j} = (z_0, w_0)$ by searching one by one the points $V_k^{(j)}$, $z_1 \leq k \leq z_2$.
3. If $x_1 \leq x_0 - 1$ then recursively call $A(x_1, x_0 - 1, z_1, z_0)$.
4. If $x_0 + 1 \leq x_2$ then recursively call $A(x_0 + 1, x_2, z_0, z_2)$.

When a nearest black point of the midpoint (x_0, j) is found in step 2 of the algorithm, we know that all nearest points of the upper half of the part of the current column are above $N_{x_0 j}$ (similarly for the lower half). This justifies the recursive calls 3 and 4 and the correctness of the algorithm.

Remark on other distances

Our algorithm can be applied to the computation of the distance map for distances other than the Euclidean distance. Such a distance must satisfy Lemma 1 and must also have the following property: if $b \leq y \leq z$ and a, x are arbitrary then the distance of (x,y) from (a,b) is less than or equal to the distance of (x,z) from (a,b). The latter property was used in the algorithm above when we restricted the search for nearest black points to the points in the vector $V^{(j)}$.

Neither of the above two properties need hold for an arbitrary distance, not even when the distance comes from a vector space norm. Indeed, the norm of the plane for which the unit sphere is the parallelogram defined by the points $(0,-0.1), (1,1), (0,0.1)$ and $(-1,-1)$, does not have any of the above properties. A distance which has both properties is the L^1 distance. Thus our algorithm is applicable to the L^1 distance.

3. Time

Let $T(m,n)$ be the maximum running time of algorithm A when $x_2 - x_1 \le m$ and $z_2 - z_1 \le n$. Obviously, $T(1,n) = n$ and in general we have the recurrence inequality

$$T(m,n) \le \max\left(n + T(\frac{m}{2}, n_1) + T(\frac{m}{2}, n_2)\right),$$

where $n_1 + n_2 = n + 1$, and the maximum is taken over all such n_1, n_2. The first two steps of the algorithm account for the n term and the two recursive calls account for the two T terms in the right hand side.

Assume for simplicity that $m = 2^L$ is a power of 2. Iterating the above inequality L times we have

$$\begin{aligned}
T(m,n) &\le n + T(\frac{m}{2}, n_1^{(1)}) + T(\frac{m}{2}, n_2^{(1)}), \quad \text{where } n_1^{(1)} + n_2^{(1)} = n+1 \\
&\le n + n_1^{(1)} + n_2^{(1)} + \sum_{i=1}^{4} T(\frac{m}{4}, n_i^{(2)}), \quad \text{where also } \sum_{i=1}^{4} n_i^{(2)} = n+1+2 \\
&\cdots \\
&\le n + \sum_{k=1}^{L-1}\sum_{i=1}^{2^k} n_i^{(k)} + \sum_{i=1}^{2^L} T(\frac{m}{2^L}, n_i^{(L)}), \quad \text{where for all } k,\ \sum_{i=1}^{2^k} n_i^{(k)} = n+1+2+\ldots+2^{k-1} \le n+m \\
&\le n + (L-1)(m+n) + \sum_{i=1}^{2^L} T(1, n_i^{(L)}).
\end{aligned}$$

Hence there is a constant $C > 0$ such that

$$T(m,n) \le C(n + m + (m+n)\log m).$$

This gives the desired $T(N-1, N-1) = O(N \log N)$.

4. Parallelizing the Algorithm

Since the above algorithm works on one column at a time, it can be easily parallelized. Consider for example an Exclusive Read, Exclusive Write Parallel Random Access Machine (EREW PRAM) with p processors [1]. This consists of random access memory to which all p processors have simultaneous access provided they read or write to different locations. Our algorithm can be parallelized on the EREW PRAM with optimal speedup as follows.

We use one more $N \times N$ array to store the values of $V_i^{(j)}$ for all i and j. To compute all $V_i^{(j)}$ we assign N/p conscecutive rows to each processor. Each processor is responsible for computing the values of $V_i^{(j)}$ for all i and j that belong to the region that has been assigned to the processor. This does not give rise to any read/write conflicts since all processors work on disjoint regions. This computation takes $O(N^2/p)$ time, since each row of the V array takes $O(N)$ time, and each processor computes N/p rows.

Having computed and stored the values of $V_i^{(j)}$, we now assign N/p consecutive columns to each processor. Each processor can perform the algorithm $A(0, N-1, 0, N-1)$ on all the columns that have been assigned to it, since, for each column, all the information that the processor needs is stored in the corresponding column of the V array. Again no read/write conflicts arise, the running time is $O(N^2 \log N/p)$ and the speedup is p.

As mentioned in section 2 our algorithm applies without any changes to the case of the L^1 distance. Of course, for the L^1 distance there is a well known and optimal $O(N^2)$ algorithm. This algorithm starts propagating a wave from the black points until it has reached the whole image. This algorithm is superior to ours in the case of the sequential implementation, but a parallel implementation of it is far from obvious, since the access to the image is not uniform. In this case our algorithm could be preferable.

Acknowledgement

We would like to thank the referees for their helpful comments.

References

[1] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Introduction to Algorithms, McGraw-Hill, New York, 1990.

[2] Danielsson, P.E., Euclidean Distance Mapping, Computer Graphics Image Processing, 14, 1980, 227-248.

[3] Yamada, H., Complete Euclidean Distance Transformation by Parallel Operation, Proc. 7th Int. Conference on Pattern Recognition, 1, 1984, 69-71.