Efficient Computation of the Euclidean Distance Transform [1]

Laurence Boxer [2] Russ Miller [3]

[1] *Computer Vision and Image Understanding* 80 (2000), 379-383

[2] Niagara University and State University of New York - Buffalo. Department of Computer and Information Sciences, Niagara University, Niagara University, NY 14109, USA. e-mail: boxer@niagara.edu. Paper was written during this author's sabbatical visit to the Center for Computational Research, State University of New York - Buffalo. Research partially supported by a grant from the Niagara University Research Council.

[3] Center for Computational Research and Department of Computer Science and Engineering, State University of New York - Buffalo, Buffalo, New York 14260, USA. e-mail: miller@cse.buffalo.edu. Research partially supported by NSF grant ACIR-9721373.

Abstract

We present a simple algorithm for the Euclidean distance transform of a binary image that runs more efficiently than other algorithms in the literature. We show our algorithm runs in optimal time for many architectures, and has optimal cost for the RAM and EREW PRAM.

Key words and phrases: Euclidean distance transform, binary image, parallel algorithm, RAM, EREW PRAM, mesh, hypercube, mesh-of-trees

1 Introduction

The Euclidean distance transform (DT) [16] computes for each pixel of an $n \times n$ binary image, the distance of the pixel to a nearest black pixel. This is an operation that has a variety of applications in image processing and computer vision, including the following.

- Expanding, shrinking, and skeletonizing [1, 3, 7, 13, 15];
- Computing Dirichlet tesselations [2, 17, 19];
- Solving binary pattern matching problems [4, 15];
- Reconstructing objects from portions of their boundaries [8, 10];
- Constructing shortest paths and computing shape factors [6, 16];
- Etc.

Recent papers have proposed algorithms for computing the Euclidean distance transform that are not cost-efficient.

In [5] and its references, sequential algorithms are discussed that are complex to describe and that have running times that are $\Omega(n^2)$. Observe that a running time of $\Theta(n^2)$ is optimal, since every pixel of the $n \times n$ binary image must be considered. In [9], algorithms are given to compute the Euclidean DT on the fine-grained mesh-of-trees and hypercube models of parallel computers. The mesh-of-trees algorithm runs in $O(\log n)$ time, using $\frac{n^3}{\log n}$ processors, for a total cost (time $\times$ processors) of $O(n^3)$. The hypercube algorithm runs in $O(\log n)$ time using $n^{2.5}$ processors for a total cost of $O(n^{2.5} \log n)$. In this paper, we give a simple algorithm for the Euclidean DT that can be implemented in optimal time and/or cost on a RAM and on a variety of fine-grained parallel architectures. Our algorithm achieves $\Theta(n^2 \log n)$ total cost for the models considered in [9]. Further, our algorithm works not only with the Euclidean metric, but with all L_p metrics.

The paper is organized as follows. In section 2, we explain models of computation, input assumptions, and parallel prefix and postfix operations. In section 3, we present and analyze our algorithm. In section 4, we give some concluding remarks.

2 Preliminaries

In this section, we define models of computation, as well as the assumptions about the input, used in this paper. We define parallel prefix and postfix operations.

2.1 Models of Computation

We discuss our algorithm on the following models of computation.

- The *random access machine (RAM)* is the traditional single-processor computer.
- A parallel random access machine (PRAM) is a computer with multiple processors that share memory. The exclusive read, exclusive write (EREW) PRAM, does not permit simultaneous reads from the same memory location, nor simultaneous writes to the same memory location.
- A *2-dimensional mesh-connected computer (mesh)* is a computer with multiple processors organized as a square lattice of processors. Each generic processor shares a bidirectional communication link with each adjacent processor in its row and in its column. A *mesh of size* n^2 has n^2 processors arranged as an $n \times n$ lattice.

  A mesh of size n^2 has communication diameter $\Theta(n)$, meaning that the maximum number of communication links separating any pair of processors in the mesh is $\Theta(n)$. Therefore, if a problem on a mesh of size n^2 requires the possibility of two processors that are $\Theta(n)$ communication links apart to communicate, and the running time of an algorithm to solve the problem is $O(n)$, then the algorithm is optimal.
- A *mesh-of-trees* computer has a mesh with a full binary tree of processors over every row and every column of the mesh. The mesh processors are leaf processors for these trees.

  If the base mesh is $n \times n$, then any pair of processors in the base can communicate a constant amount of data in $O(\log n)$ time as follows. Suppose the data must get from processor $P_{a,b}$ to processor $P_{c,d}$, where the subscripts indicate the row and column, respectively, of the processor in the base mesh. Processor $P_{a,b}$ can send the data through its column tree to processor $P_{c,b}$ in $O(\log n)$ time; then processor $P_{c,b}$ can send the data through its row tree to processor $P_{c,d}$ in $O(\log n)$ time. Since there are processors in the base mesh (*e.g.*, opposite corners) that are

no closer to each other than $\Theta(\log n)$ communications links, it follows that $O(\log n)$ time is optimal for a mesh-of-trees algorithm that requires data communication between maximally separated processors.

- A hypercube computer has n processors, where $n = 2^d$ for some nonnegative integer d. The processors are labeled from 0 to $n-1$. Two processors are connected if and only if their labels, written as binary numbers, differ in exactly one bit. It follows [11] that maximally separated processors are $d = \log_2 n$ communications links apart. Therefore, an algorithm that requires communications between maximally separated processors will require $\Omega(\log n)$ time.

2.2 Input Assumptions

Input to our problem consists of an $n \times n$ binary image. For parallel models other than the PRAM, we will assume there is a natural mapping of the image to the set of processors so that each pixel of the image is initially stored in the corresponding processor, as follows.

- For the $n \times n$ mesh, we assume the pixel in the i^{th} row and j^{th} column of the image is stored in the processor in the i^{th} row and j^{th} column.

- For the mesh-of-trees, we assume the pixel in the i^{th} row and j^{th} column is stored in the processor in the i^{th} row and j^{th} column of the base mesh.

- For the hypercube of $n^2 = 2^{2k}$ processors, we consider the binary label of a processor in two groups of k bits apiece, the *most significant bits* and the *least significant bits*; the processor with label having most significant bits representing the value i and least significant bits representing the value j stores the pixel in row i and column j. It follows that each row and each column of the image is stored in a subhypercube of n processors.

2.3 Parallel Prefix

Let A be a nonempty set and let $\circ$ be a binary operator for A that is closed and associative. Let $X = \{x_i\}_{i=1}^n \subset A$. A *parallel prefix computation* on X produces the set of values

$$\{x_1, x_1 \circ x_2, \ldots, x_1 \circ x_2 \circ \cdots \circ x_n\}.$$

A *parallel postfix computation* on X produces the set of values

$$\{x_1 \circ x_2 \circ \cdots \circ x_n, x_2 \circ \cdots \circ x_n, \ldots, x_{n-1} \circ x_n, x_n\}.$$

These operations are often thought of as *scans* or *sweeps*, used to accumulate values (including partial computations) that result from applying an operator to a list. For example, a parallel prefix (postfix) operation based on the maximum (respectively, minimum) operator may be used to inform each pixel p in a digital image of the nearest black pixel in its row that is not to the right (respectively, left) of p. If a single computation of the $\circ$ operator takes $\Theta(1)$ time, then we have the following running times for parallel prefix and postfix operations (see [11]).

- $\Theta(n)$ time for a RAM, and for a linear array of n processors (*e.g.*, a row or column of an $n \times n$ mesh).
- $\Theta(\log n)$ time for an EREW PRAM of $\Theta(n/\log n)$ processors. In the algorithm of section 3, several steps require simultaneous parallel prefix/postfix operations on every row or column of an $n \times n$ binary image. During such a step, we will assume that processors

  $$\frac{(i-1)n}{\log n} + 1, \frac{(i-1)n}{\log n} + 2, \ldots, \frac{in}{\log n}$$

  are assigned to row i (respectively, column i), $i \in \{1, 2, \ldots, n\}$.
- $\Theta(\log n)$ time for a tree with n leaf processors, operating on $\Theta(1)$ data per leaf processor (*e.g.*, a row or column tree in a mesh-of-trees with $n \times n$ base mesh).
- $\Theta(\log n)$ time for a hypercube of n processors.

3 Algorithm

A key to our algorithm is the following property, observed in [18].

Proposition 3.1 *Let x and y be two pixels in the same column of a digital image. Let z be a black pixel that is nearest to x among the black pixels of the row of x. Then z is a nearest black pixel to y among the black pixels of the row of x, with respect to any L_p metric.* ■

We now present our algorithm for the Euclidean DT.

1. For each row of the image, perform a parallel prefix operation so that each pixel $x_{i,j}$ learns the nearest black pixel (if it exists) not to the right of $x_{i,j}$ in row i.

2. For each row of the image, perform a parallel postfix operation so that each pixel $x_{i,j}$ learns the nearest black pixel (if it exists) not to the left of $x_{i,j}$ in row i.

3. Each pixel compares its nearest not-right black pixel and its nearest not-left black pixel to determine a nearest black pixel (if it exists) in its row.

   It follows from Proposition 3.1 that for every pixel $x_{i,j}$ in the image, a nearest black pixel in the image can be found by comparing in-row nearest black pixels to $x_{k,j}$ (a pixel in the same column as $x_{i,j}$) over all rows k. This is done via steps similar to the above, as follows.

4. For each column of the image, perform a parallel prefix operation so each pixel $x_{i,j}$ learns the nearest black pixel (if it exists) not below $x_{i,j}$.

5. For each column of the image, perform a parallel postfix operation so each pixel $x_{i,j}$ learns the nearest black pixel (if it exists) not above $x_{i,j}$.

6. Each pixel compares its nearest not-below black pixel and its nearest not-above black pixel to determine a nearest black pixel in the image.

Observe that for a mesh, the algorithm could be described in two steps: the first 3 steps above could be replaced by a parallel row rotation step, and the last 3 steps above could be replaced by a parallel column rotation step.

The algorithm consists of four parallel prefix or postfix steps and two steps that call for each pixel to be associated with a $\Theta(1)$-time decision. Thus, we have the following.

Theorem 3.2 *Let X be an $n \times n$ binary image. Then the Euclidean DT for X can be computed using the following resources.*

- *Optimal $\Theta(n^2)$ time on a RAM.*
- *$\Theta(\log n)$ time on an EREW PRAM of $\Theta(n^2/\log n)$ processors (optimal cost).*

- $\Theta(n)$ *time on an* $n \times n$ *mesh (optimal time for this architecture).*
- $\Theta(\log n)$ *time on a mesh-of-trees with* $n \times n$ *base mesh (optimal time for this architecture).*
- $\Theta(\log n)$ *time on a hypercube with* n^2 *processors (optimal time for this architecture).*

Proof: For the RAM, the asserted running time of $\Theta(n^2)$ follows from the fact that each of n rows and each of n columns has a prefix and a postfix operation that require $\Theta(n)$ time apiece, and there are $\Theta(n^2)$ decisions that require $\Theta(1)$ time apiece. That this is optimal follows from the fact that every pixel must be considered. For the parallel models, the asserted running times follow from the running times required for parallel prefix and postfix operations discussed in section 2.3. The claim of optimal cost for the EREW PRAM follows from the running time of the RAM implementation. The claims that the running times are optimal for the mesh, mesh-of-trees, and hypercube follow from the fact that for each of these models, the running time coincides with the communications diameter. ■

4 Further Remarks

We have given a simple algorithm for computing the Euclidean distance transform of an $n \times n$ binary image. The algorithm is optimal for the RAM and simpler than those of [5] and its references. Our algorithm is also cost-optimal for the EREW PRAM, has optimal running times on a variety of models of fine-grained parallel computation, and greatly improves on the cost of the algorithm of [9] for the mesh-of-trees and the hypercube.

Other problems in image analysis, such as computing the Hausdorff distance [14] between two $n \times n$ digital images and finding the distance from each black pixel or component of an $n \times n$ digital image to the nearest distinctly labeled pixel or component of the image, have efficient solutions that make use of similar computations. See [18, 12, 11].

5 Acknowledgement

We thank the anonymous referees for their suggestions concerning the presentation of our results.

References

[1] C. Arcelli and G. Sanniti di Baja, A width-independent fast thinning algorithm, *IEEE Trans. Pattern Analysis and Machine Intelligence* **C-7** No. 4 (1985), 463-474.

[2] C. Arcelli and G. Sanniti di Baja, Computing Voronoi diagrams in digital pictures, *Pattern Recognition Letters* 4, (1986), 383-389.

[3] G. Borgefors, Distance transformations in arbitrary dimensions, *Computer Vision Graphics Image Processing* 27 (1984), 321-345.

[4] G. Borgefors, Hierarchical chamfer matching: a parametric edge matching algorithm, *IEEE Trans. Pattern Analysis and Machine Intelligence* **C-10** No. 6 (1988), 849-865.

[5] O. Cuisenaire and B. Macq, Fast Euclidean distance transformation by propagation using multiple neighborhoods, *Computer Vision and Image Understanding* 76 (1999), 163-172.

[6] P.E. Danielsson, A new shape factor, *Computer Graphics Image Processing* 7 (1978), 292-299.

[7] P.E. Danielsson, Euclidean distance mapping, *Computer Graphics Image Processing* 14 (1980), 227-248.

[8] M.A. Fishler and P. Barrett, An iconic transform for sketch completion and shape abstraction, *Computer Vision Graphics and Image Understanding* 13, (1980), 334-360.

[9] Y.-H. Lee, S.-J. Horng, T.-W. Kao, and Y.-J. Chen, Parallel computation of the Euclidean distance transform on the mesh of trees and the hypercube computer, *Computer Vision and Image Understanding* 68 (1997), 109-119.

[10] T. Matsuyama and T.Y. Phillips, Digital realization of the labeled Voronoi diagram and its application to closed boundary detection, *Proc. 7th Intl. Conf. on Pattern Recognition*, 1984, 478-480.

[11] R. Miller and L. Boxer, *Algorithms Sequential and Parallel*, Prentice Hall, 2000.

[12] R. Miller and Q.F. Stout, *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*, The MIT Press, Cambridge, MA, 1996.

[13] U. Montanari, A method for obtaining skeletons using a quasi-Euclidean distance, *J. Association for Computing Machinery* 15 (1968), 600-624.

[14] S.B. Nadler, Jr., *Hyperspaces of Sets*, Marcel Dekker, Inc., New York, 1978.

[15] D.W. Paglieroni, A unified distance transform algorithm and architecture, *Mach. Vision Appl.* 5 (1992), 47-55.

[16] A. Rosenfeld and J.L. Pfalz, Distance function on digital pictures, *Pattern Recognition* 1 (1968), 33-61.

[17] Q. Schwarzkopf, Parallel computation of discrete Voronoi diagrams, *Proc. 6th Annual Symp. Theoretical Aspects of Computer Science* (1989).

[18] R. Shonkwiler, An image algorithm for computing the Hausdorff distance efficiently in linear time, *Information Processing Letters* 30 (1989), 87-89.

[19] O.Z. Ye, The signed Euclidean distance transform and its applications, Proc. 9th Intl. Conf. on Pattern Recognition (1988), 495-499.