Linear Time Euclidean Distance Transform Algorithms

Heinz Breu Joseph Gil David Kirkpatrick
Michael Werman

Abstract

Two linear time (and hence asymptotically optimal) algorithms for computing the Euclidean distance transform of a two-dimensional binary image are presented. The algorithms are based on the construction and regular sampling of the Voronoi diagram whose sites consist of the unit (feature) pixels in the image. The first algorithm, which is of primarily theoretical interest, constructs the complete Voronoi diagram. The second, more practical, algorithm constructs the Voronoi diagram where it intersects the horizontal lines passing through the image pixel centres. Extensions to higher dimensional images and to other distance functions are also discussed.

1 Introduction

A two-dimensional binary *image* is a function, $\mathcal{I}$, from the elements of an n by m array, referred to as *pixels*, to $\{0,1\}$. Pixels of unit (respectively, zero) value are referred to as *feature* (respectively, *background*) pixels of the image. We associate the pixel in row r and column c with the Cartesian point (c,r). Thus, any distance function defined on the Cartesian plane induces a distance function on the space of image pixels. For a given distance function, the *distance transform* of an image $\mathcal{I}$ is an assignment, to each pixel p, of the distance between p and the closest feature pixel in $\mathcal{I}$. The *nearest-neighbour transform* of an image is an assignment, to each pixel p, of the identity (or distance information sufficient to compute the identity in $O(1)$ time) of a feature pixel closest to p. Assuming that computing the distance between two pixels is a $O(1)$ time operation, it should be clear that the distance transform can be constructed from the nearest-neighbour transform in time linear in the total number of pixels. It should also be clear that, at least in their explicit forms, both transforms require at least linear time for their computation.

Distance and nearest-neighbour transforms have for a considerable time been recognized as important sources and encodings of metric information associated with images in digital picture processing, pattern recognition, robotics and related applications [17, 13, 2, 1, 10, 6, 11]. Such transforms were first introduced by Rosenfeld and Pfaltz [18, 19] and have been the focus of an extensive literature (cf. [13, 16]).

The most natural metric for computing distance in most applications is the Euclidean metric. However, the lack of efficient algorithms for computing the Euclidean distance transform has led many researchers to define and use other metrics (such as city-block, chessboard or chamfer [8, 22, 4]), or approximations to the exact Euclidean distance transform [7, 4, 5, 22]. While other metrics lend themselves to efficient algorithms, they do so at the expense of being dependent on the orientation of the coordinate system used for representing the image. To quote Borgefors [4], "If the Euclidean distance transform could be easily computed there would be no need for approximations".

By its definition, the Euclidean distance transform is a global operation; the most straightforward approach to its construction involves, for each pixel, an amount of computation that is proportional to the size of the entire image ($O(nm)$). In practice, one hopes to exploit the fact that the distance transform is a discretized version of the continuous distance transform associated with the feature pixels. Specifically, the transform values in a neighbourhood vary in a smooth and predictable fashion. Non-trivial approaches to the construction of (approximate) Euclidean distance transforms involve the propagation of distance information from pixels to neighbouring pixels. Methods vary on the size and shape of the neighbourhood, and the order in which pixels are processed.

In one family of algorithms (cf. [4, 7]), each pixel p consults, perhaps repeatedly, with other pixels in a typically small neighbourhood of p, and bases its updated transform value on those of its neighbours. A tradeoff exists between the quality of the resulting approximation, the size and shape of the neighbourhood, and the number of iterations performed. The total computational effort is proportional to the number of pixels, the neighbourhood size, and the number of iterations. Thus, to minimize cost, sequential algorithms typically use a constant (independent of the image size) sized neighbourhood and a constant number of iterations often structured to process pixels in some kind of scan order. Parallel implementations of this approach (cf. [21]) repeatedly update the transform value until it stabilizes (i.e., until a parallel round brings no change). In the worst case this may involve a number of iterations proportional to the image perimeter ($O(\max\{n, m\})$), but this is offset by the fact that individual pixels are processed in parallel.

Danielsson [7] describes a sequential nearest-neighbour transform algorithm

of the above type, and shows that it is accurate to within one pixel, that is, the identified neighbour may be up to one pixel further away than the true nearest neighbour. Danielsson also argues that errors of this magnitude are inevitable with algorithms of this type (due essentially to discontinuities arising from the discretization of the continuous nearest-neighbour transform). Yamada [21] shows that parallel implementations are not subject to the same inherent approximation errors. Yamada proves that with a suitably large (in fact, eight element) neighbourhood, one can exploit the difference between propagation distance and Euclidean distance to overcome the problems arising from discretization. Thus, we have inexact sequential algorithms that take $O(nm)$ time in the worst case, or exact parallel algorithms that take $O(\max\{n, m\})$ parallel time with $O(nm)$ processors.

An alternative approach that does not lend itself as readily to parallel implementation, involves simulating a wavefront emanating from all feature pixels (cf. [16] and references therein). The idea here is to have certain pixels (those on the current wavefront) inform their neighbours so that the latter can update their distance transform values, and possibly join the wavefront. Thus, at any moment the propagation of distance information takes place only where there is potential for change. Algorithms of this type differ in their strategies for updating the wavefront. In fact, the scan-based algorithms can be viewed as a kind of wavefront approach as well. The wavefront approach suffers from the same inherent discretization error discussed above. Despite this, Ragnemalm [16] claims that it is possible to simulate Yamada's error-free algorithm using an $O(nm)$ time sequential wavefront algorithm. However, no proof of this assertion is provided (other than experimental "verification").

Yet another approach, again pioneered by Rosenfeld and Pfaltz [18], is based on the idea of dimensionality reduction. In short, the transform is first constructed using the distance function in one lower dimension (say, restricted to the pixels in a given row of the image) and then this intermediate result is used in a second phase to construct the full two-dimensional transform. Paglieroni [14, 13] abstracts this approach, showing that it is applicable to a broad class of distance functions, including Euclidean distance. While certain heuristics apply, even in the most abstract setting, to reduce the cost on typical images, the worst-case cost for an n by m image is $\Theta(nm \min\{n, m\})$. Recently an $O(nm \log(nm))$ time algorithm for computing the exact Euclidean distance transform was proposed [12]. This algorithm can be viewed as a special instance of Paglieroni's algorithm, in which properties of the Euclidean metric are exploited. The algorithm presented in Section 3 of this paper could be interpreted as a further (and, in an asymptotic sense, ultimate) refinement of Paglieroni's algorithm for Euclidean distance transforms.

In the next two sections of this paper two linear (in the size of the im-

age) time algorithms for constructing the nearest-neighbour transform are described. Both algorithms are based on the construction of the Voronoi diagram whose sites are the feature pixels. The first algorithm, which is of primarily theoretical interest, constructs the complete Voronoi diagram. The second, more practical, algorithm constructs the Voronoi diagram where it intersects the horizontal lines passing through the image pixel centres. The *Voronoi diagram* [20, 9] of a set of sites $S = \{h_1, \ldots, h_s\}$, denoted $\mathcal{V}_S$, consists of s disjoint *Voronoi cells* $\{C^S_{h_1}, \ldots, C^S_{h_s}\}$, where C^S_h (or simply C_h, if S is understood) denotes the set of all points in the plane whose closest site is h, together with the cell boundaries formed by points equidistant from two (*Voronoi edge* points) or more (*Voronoi vertices*) sites. We refer to h as the *Voronoi centre* of C^S_h.

The nearest-neighbour transform can be viewed as a discretized version of the Voronoi diagram, formed by sampling the diagram (i.e. recording the associated Voronoi centres) at points corresponding to all pixel locations. Despite this, our algorithms first view the problem in its continuous form ignoring (at least for the sake of the algorithm design) the inherent discreteness and regularity of the inputs and outputs. It would appear that the willingness to address the problem in its continuous form is the key to developing simple and asymptotically optimal algorithms for the Euclidean distance transform. We note, however, that the discreteness and regularity of the inputs and outputs do play an essential role in the detailed design of the algorithms, and in their analysis. In fact, it is well known (cf. [15]) that linear algorithms do not exist for determining, for each of an arbitrary set of query points, the closest of an arbitrary collection of sites.

2 Transform based on Complete Voronoi Diagram Construction

Determining the nearest among s arbitrary sites for each of q arbitrary query points requires, in the worst case, $\Theta(s \log s)$ time to construct the Voronoi diagram of the sites and $\Theta(q \log s)$ time to answer the queries (even if the diagram is given). In fact, these bounds are realized by familiar algorithms for constructing and querying Voronoi diagrams (cf. [15]). For our application this gives an immediate $O(nm \log(nm))$ time nearest-neighbour transform algorithm, since the number of pixels (and hence the number of both sites and query points) is $O(nm)$. Our objective, in the remainder of this section, is to demonstrate that this time complexity can be reduced to $O(nm)$ by exploiting the fact that both sites and query points are subsets of a two-dimensional pixel array.

2.1 Voronoi diagrams of pixel subsets

A familiar approach (cf. [15]) to the construction of Voronoi diagrams is divide and conquer: (linearly) separate the sites into two subsets, recursively construct the Voronoi diagrams of the subsets, and "merge" the resulting diagrams into the Voronoi diagram of the full set. The linear separation of the subsets ensures that much of the structure of the smaller Voronoi diagrams can be retained; they are in effect "sewn together" along one (monotone) chain of edges (called the *Voronoi merge boundary*) formed by points with two nearest sites, one from each subset. The cost of the merge step is proportional to the number of edges on this chain, which is just the number of cells that border on this chain. While the latter may be significantly larger than the number of cells that intersect the partition boundary (the line separating the set of sites), it is, of course, at worst the total number of sites, s. Thus if each separation is balanced the depth of the recursion is $O(\log s)$ and the total cost is $O(s \log s)$.

If we take note of the fact that (i) the sites consist of a subset of some n by m array of pixels and (ii) it suffices to construct the Voronoi diagram restricted to the rectangle $[1, n] \times [1, m]$, then we can ignore the structure of the subset and partition the array instead (by splitting it in half along its longest dimension). Although this may lead to an unbalanced division of the sites themselves, it is not difficult to show that the cost of merging the Voronoi diagrams of the resulting site subsets is sublinear in the size of the participating arrays, which gives rise to a linear total cost. (Intuitively, if a site is far removed from the partition boundary or if it is surrounded by other closer sites, its cell cannot border the Voronoi merge boundary. More precisely, if a site in any i by j subproblem has distance greater than $\min\{n, m\}$ from the partition boundary then it cannot influence the Voronoi merge boundary for that subproblem. Furthermore, if a site has distance greater than $(\min\{i, j\})^{1/2}$ from the boundary of its subarray, then it can influence the merge boundary only if at least one of its four rectilinear closest sites has distance at least $(\min\{i, j\})^{1/2}$.) Thus, if $T(i, j)$ denotes the worse-case cost of constructing the Voronoi diagram for any set of sites drawn from an i by j subarray of an n by m array, we have the recurrences $T(n, 2^k n) \leq 2T(n, 2^{k-1} n) + O(n^2)$, since there are $O(n^2)$ pixels of distance at most n from the partition boundary of two n by $2^{k-1} n$ subarrays, and $T(i, i) \leq 4T(i/2, i/2) + O(i^{3/2})$, since there are $O(i^{3/2})$ pixels of distance at most $i^{1/2}$ from the boundary of an i by i array, and at most $O(i^{3/2})$ of the remaining sites can influence the merge boundary, by the sparsity condition. If follows from these recurrences that $T(n, n) = O(n^2)$ and, more generally, $T(n, m) = O(nm)$.

2.2 Finding nearest pixels given the Voronoi diagram

Given the Voronoi diagram of the feature pixels, we can find the feature pixel nearest to pixel p by determining the Voronoi cell containing p. If we treat all such queries independently there is no way to avoid taking $\Omega(nm \log(nm))$ time in the worst case. However, it we batch the queries into rows then all $O(nm)$ queries can be answered in $O(nm)$ time in total. This follows from the facts that (i) a Voronoi diagram on s sites can be decomposed, in $O(s)$ time, into trapezoids whose non-horizontal edges are portions of Voronoi edges; (ii) the cost of visiting, in sequence, all of the Voronoi cells (equivalently, trapezoids) intersected by a horizontal line is proportional to the number of cells visited; and (iii) any horizontal line intersects the Voronoi diagram of a set of sites drawn from an n by m array in $O(m)$ cells (at most two cells are associated with sites in any fixed column).

2.3 Advantages and drawbacks

The nearest-neighbour transform algorithm implicit in the preceding subsections has several attractive features in addition to its asymptotically optimal worst case complexity. First, we note that it can be implemented using exact rational arithmetic, since it is concerned exclusively with bisectors of pixel pairs. Second, it seems to be more adaptable than most array-based algorithms to sparse images (assuming the feature pixels can be succinctly specified) and to the computation of "partial" nearest-neighbour transforms (i.e. computation of transform values at only a small subset of the domain). In both of these cases, the analysis above may be unduly pessimistic.

On the other hand, this explicit Voronoi diagram construction requires extra space of size proportional to the input image. In addition this space must be used to represent a data structure whose complexity far exceeds that of the input or output of the problem. The latter has an impact not only on the constants of proportionality inherent in any implementation, but also on the extent to which the algorithm can be effectively parallelized.

In the next section, we develop another asymptotically optimal nearest-neighbour transform algorithm that retains the advantages of the explicit Voronoi diagram approach along with the simpler data structures of the earlier array-based algorithms. This is achieved by constructing only a partial Voronoi diagram, that is, determining its structure only where it is necessary to know it.

3 Transform based on Partial Voronoi Diagram Construction

The nearest-neighbour transform is a discretized representation of the Voronoi diagram of the feature pixels of a given image. In the previous section we demonstrated that the transform can be computed efficiently by first constructing the (complete) Voronoi diagram and then sampling it at all pixel locations. In this section we show that we can achieve the same asymptotic time complexity, together with significant practical improvements, by constructing a Voronoi structure intermediate between the complete Voronoi diagram and the discretized version that is our ultimate objective. Specifically, we construct directly the intersection of the Voronoi diagram of the feature pixels with each row of the image. Since each such construct has complexity bounded by the width of the image, and each row can be processed independently, the total amount of auxiliary space is significantly reduced (compared to the previous algorithm) and the potential for exploiting parallelism is significantly increased.

3.1 High level description

As before, the algorithm takes as input an n by m binary image $\mathcal{I}$. Let $\mathcal{F}$ denote the set of feature pixels of $\mathcal{I}$, let $\mathcal{F}_r$ denote the set $\mathcal{F}$ restricted to rows 1 through r, and let $\mathcal{V}_r$ denote the Voronoi diagram of $\mathcal{F}_r$. For each row r, the algorithm computes the intersection of $\mathcal{V}_r$ with row r. More precisely, at its rth iteration the algorithm constructs the list $\mathcal{L}_r$ of Voronoi cells in $\mathcal{V}_r$ that intersect the line $y = r$ (which we denote by R_r). Note that $\mathcal{L}_r$ can be conveniently represented as a list of Voronoi centres (pixels) sorted by column order.

If p and q are distinct points in the plane, we denote by $\widehat{pq}$ the intersection of the perpendicular bisector between p and q with the line R_r. If p is any point in the plane, we denote its coordinates by $(p.x, p.y)$. (For the sake of consistency, imagine that the rows of an image are numbered from bottom to top, and the columns are numbered from left to right.)

Observation 1 *If u and v belong to $\mathcal{F}_r$, and $u.x = v.x$ and $u.y > v.y$, then the Voronoi cell centred at site v in $\mathcal{V}_r$ (i.e. $C_v^{\mathcal{F}_r}$) does not intersect R_r.*

Observation 2 *If u and v belong to $\mathcal{F}_r$, and $u.x < v.x$, then $C_u^{\mathcal{F}_r}$ (respectively, $C_v^{\mathcal{F}_r}$) lies to the left (respectively, right) of the perpendicular bisector between u and v.*

From Observation 1, we can conclude that no two points in $\mathcal{L}_r$ have the same column value, and hence $|\mathcal{L}_r| \leq m$. The algorithm scans row r, and, using $\mathcal{L}_r$, labels each pixel with the centre of the Voronoi cell of $\mathcal{V}_r$ in which it lies. By Observation 2, as R_r is scanned from left to right the intersection with cell C_u occurs before the intersection with cell C_v if and only if u precedes v in $\mathcal{L}_r$. That is, in scanning each row, the algorithm visits the Voronoi cells in column order, so that it labels each row in $O(m)$ time.

The process described above assigns to each pixel in row r, for $1 \leq r \leq n$, the closest element of $\mathcal{F}_r$ (i.e. the closest feature pixel belonging to a row of index no greater than r). To complete the transform computation, the process must be repeated, the second time iterating from row n down to 1 (i.e., the image is turned upside down). Since the total cost, given the lists $\mathcal{L}_r$, $1 \leq r \leq n$, is $O(nm)$, it remains to determine the cost of constructing the lists.

3.2 Computing $\mathcal{L}_r$

Let $\mathsf{Candidates}_r$ be the column-ordered list of pixels in $\mathcal{F}_r$ that attain the maximal row values in their respective columns. By Observation 1, every pixel in $\mathcal{L}_r$ is in $\mathsf{Candidates}_r$. Thus, it suffices to construct $\mathsf{Candidates}_r$ and then determine which centres in $\mathsf{Candidates}_r$ are *not* in $\mathcal{L}_r$. Note that $\mathsf{Candidates}_1$ is just $\mathcal{F}_1$ presented in column order. Furthermore, $\mathsf{Candidates}_{i+1}$ is easily constructed from $\mathsf{Candidates}_i$ in $O(m)$ time. Hence, constructing the sequence $\mathsf{Candidates}_1, \ldots, \mathsf{Candidates}_n$ takes $O(nm)$ time in total. Hereafter, we will assume that $\mathsf{Candidates}_r$ is available and focus on the process of determing those elements of $\mathsf{Candidates}_r$ that are not in $\mathcal{L}_r$. Since the subscript r simply describes the current iteration, we will drop this subscript from $\mathcal{F}_r$, $\mathsf{Candidates}_r$, $\mathcal{L}_r$ and R_r, whenever it is understood from the context.

Let u, v, and w be three pixels in $\mathsf{Candidates}$ such that $u.x < v.x < w.x$. Let $\widehat{uv}$ denote the intersection of the bisector between u and v with R, and let $\widehat{vw}$ denote the intersection of the bisector between v and w with R, see Figure 1. By Observation 2, a point on R right of $\widehat{vw}.x$ is not in C_v, and a point left of $\widehat{uv}.x$ is not in C_v. Therefore, C_v does not intersect R if $\widehat{uv}.x \geq \widehat{vw}.x$.

We can now construct $\mathcal{L}$ from $\mathsf{Candidates}$ with the predicate **Remove**(u, v, w, r). **Remove** has four arguments: pixels u, v, and w, where $u.x < v.x < w.x$, and row index r. Predicate **Remove** is *TRUE* if and only if $\widehat{uv}.x \geq \widehat{vw}.x$, that is, the Voronoi cell C_v does not intersect row R.

It turns out, that for the Euclidean metric, this predicate **Remove** can be implemented using only integer arithmetic. To see this, first note that the intersection of the perpendicular bisector of u and v (respectively, v and w) is the unique point $\widehat{uv}$ (respectively, $\widehat{vw}$) on R equidistant from u and

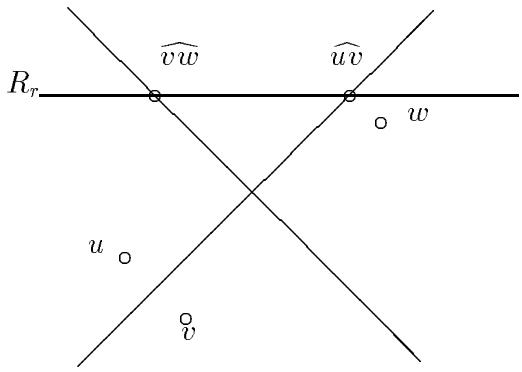

Figure 1: Voronoi cell C_v does not intersect line R_r if $\widehat{uv}.x \geq \widehat{vw}.x$.

v (respectively, u and v). More formally, if $||p - q||$ denotes the Euclidean distance between points p and q, then

$$\begin{aligned} ||u - \widehat{uv}||^2 &= (u.x - \widehat{uv}.x)^2 + (u.y - r)^2 \text{ and} \\ ||v - \widehat{uv}||^2 &= (v.x - \widehat{uv}.x)^2 + (v.y - r)^2. \end{aligned}$$

Thus, $||u - \widehat{uv}|| = ||v - \widehat{uv}||$ implies that

$$\widehat{uv}.x = \frac{(v.x)^2 - (u.x)^2 - 2r(v.y - u.y) + (v.y)^2 - (u.y)^2}{2(v.x - u.x)}.$$

Hence, $\widehat{uv}.x \geq \widehat{vw}.x$ if and only if

$$\begin{aligned} &(w.x - v.x)((v.x)^2 - (u.x)^2 - 2r(v.y - u.y) + (v.y)^2 - (u.y)^2) \\ &\quad \geq (v.x - u.x)((w.x)^2 - (v.x)^2 - 2r(w.y - v.y) + (w.y)^2 - (v.y)^2). \end{aligned}$$

This gives us an easy implementation of the predicate **Remove** as a function.

Function **Remove**(u, v, w, r)
return $((w.x - v.x)((v.x)^2 - (u.x)^2 - 2r(v.y - u.y) + (v.y)^2 - (u.y)^2)$
$\geq (v.x - u.x)((w.x)^2 - (v.x)^2 - 2r(w.y - v.y) + (w.y)^2 - (v.y)^2))$

In fact, an even simpler implementation [1] follows if we normalize the y-coordinates by first subtracting out r. We can now use **Remove** in the following algorithm. This algorithm runs in $O(n)$ time since there are never more

[1] The normalized y-coordinate $p.\bar{y}$ of a point p is just its distance from the line R. Any terms that have r as a factor vanish. This gives the following equivalent function.

than n points in Candidates, and each point is added and removed from $\mathcal{L}$ at most once. Note that $\mathcal{L}$ can be implemented as a stack.

Algorithm: Create $\mathcal{L}$ from Candidates.

$\mathcal{L}[1] \leftarrow$ Candidates$[1]$
$\mathcal{L}[2] \leftarrow$ Candidates$[2]$
$k \leftarrow 2,\ l \leftarrow 3,\ c \leftarrow |$Candidates$|$
while $l \leq c$ **do**
    $w \leftarrow$ Candidates$[l]$
    while $k \geq 2$ **and Remove**$(\mathcal{L}[k-1], \mathcal{L}[k], w, r)$ **do**
        $k \leftarrow k - 1$
    od
    $k \leftarrow k + 1,\ l \leftarrow l + 1$
    $\mathcal{L}[k] \leftarrow w$
od

Note that the inner loop removes points from the top of $\mathcal{L}$ as long as their elimination is certified by the predicate **Remove**. Once this is complete, the outer loop simply adds the next element of Candidates to $\mathcal{L}$. Given this, it is straightforward to confirm that the following invariants are preserved by both of the loops of the above algorithm. Recall that $\mathcal{V}_S$ denotes the Voronoi diagram associated with the set of sites S.

Invariant 1 $\mathcal{V}_{\mathcal{F}} \cap R = \mathcal{V}_{\mathcal{L} \cup \text{Candidates}[l,c]} \cap R$.

Invariant 2 *If we define* $(\mathcal{L}[\widehat{0}\mathcal{L}[1]).x = -\infty$ *and* $(\mathcal{L}[k]\widehat{\mathcal{L}[k+1]}).x = \infty$, *then for all* i, $1 \leq i \leq k$, $C^{\mathcal{L}}_{\mathcal{L}[i]} \cap R$ *is the nonempty interval* $((\mathcal{L}[\widehat{i-1}]\mathcal{L}[i]).x, (\mathcal{L}[i]\widehat{\mathcal{L}[i+1]}).x)$.

Theorem 1 *Algorithm "Create* $\mathcal{L}$ *from* Candidates*" is correct.*

Proof: Following initialization, Invariant 1 (specifically, $\mathcal{V}_{\mathcal{F}} \cap R = \mathcal{V}_{\text{Candidates}} \cap R$) is an immediate consequence of Observation 1. Invariant 2 (specifically, $C^{\mathcal{L}}_{\mathcal{L}[1]} \cap R = (-\infty, (\mathcal{L}[\widehat{1}]\mathcal{L}[2]).x)$ and $C^{\mathcal{L}}_{\mathcal{L}[2]} \cap R = ((\mathcal{L}[\widehat{1}]\mathcal{L}[2]).x, \infty))$ is an immediate consequence of Observation 2. At the conclusion of the algorithm $l = c + 1$, and hence $\mathcal{V}_{\mathcal{F}} \cap R = \mathcal{V}_{\mathcal{L}} \cap R$, by Invariant 1. Together with Invariant 2 this establishes the correctness of the algorithm. □

Function **Remove**(u, v, w)
    return $(w.x - v.x)((v.x)^2 - (u.x)^2 + (v.\bar{y})^2 - (u.\bar{y})^2) \geq (v.x - u.x)((w.x)^2 - (v.x)^2 + (w.\bar{y})^2 - (v.\bar{y})^2)$

This method may be preferred in an implementation if the squared distance values are readily available, for example, from the algorithm's output array.

4 Conclusions

We have presented two linear-time algorithms for constructing the nearest-neighbour (or distance) transform of a two-dimensional binary image under the Euclidean metric. Apparently, these are the first such algorithms whose worst case optimality has been formally verified. Interestingly, the algorithms are conceptually simple enough that they are of practical value as well. We are in the process of conducting emperical evaluations of these algorithms; these results will be reported elsewhere.

Our results raise several related questions the most interesting of which seems to be whether or not the algorithms can be generalized to work on three and higher dimensional images (cf. [3] for motivation and related work). In fact, it turns out that a natural (but nontrivial) generalization provides a linear time algorithm for Euclidean distance transforms of rectilinear images in *all* fixed dimensions. Similar results (including algorithms that are optimal to within logarithmic factors) hold for a broad class of distance functions satisfying axioms similar to those set out by Paglieroni [13]. Like the algorithm of Section 3, these more general algorithms lend themselves to parallel implementation and can be implemented with sublinear auxiliary storage. A more extensive paper detailing these results is in preparation.

References

[1] N. Ahuja. Dot pattern processing using Voronoi polygons as neighborhoods. In *Proc. 5th Internat. Conf. Pattern Recogn.*, pages 1122–1127, Miami Beach, FL, 1980.

[2] C. Arcelli and G. Sanniti de Baja. Computing Voronoi diagrams in digital pictures. *Pattern Recognition Letters*, pages 383–389, 1986.

[3] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27:321–345, 1984.

[4] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[5] G. Borgefors. A new distance transformation approximating the Euclidean distance. *Proc. International Joint Conf. on Pattern Recognition*, pages 336–338, 1986.

[6] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[7] P.E. Danielson. Euclidean distance mapping. *Comp. Graph. and Image Proc.*, 14:227–248, 1980.

[8] P.P. Das and P.P. Chakrabarti. Distance functions in digital geometry. *Information Sciences*, 42:113–136, 1987.

[9] G. L. Dirichlet. Über die Reduktion der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen. *J. Reine Angew. Math.*, 40:209–227, 1850.

[10] J. Fairfield. Segmenting dot patterns by Voronoi diagram concavity. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-32:104–110, 1983.

[11] F. Klein and O. Kübler. Euclidean distance transformations and model-guided image interpretation. *Pattern Recognition Letters*, 5:19–29, 1987.

[12] M.N. Kolountzakis and K.N. Kutulakos. Fast computation of the Euclidean distance map for binary images. *Information Processing Letters*, 43:181–184, 1992.

[13] D.W. Paglieroni. Distance transforms. *Computer Vision, Graphics and Image Processing:Graphical Models and Image Processing*, 54:56–74, 1992.

[14] D.W. Paglieroni. A unified distance transform algorithm and architecture. *Machine Vision and Applications*, 5:47–55, 1992.

[15] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[16] I. Ragnemalm. Neighborhoods for distance transformations using ordered propagation. *Computer Vision, Graphics and Image Processing: Image Understanding*, 56:399–409, 1992.

[17] A. Rosenfeld and A.C. Kak. *Digital Picture Processing.* Academic Press, New York, second edition, 1978.

[18] A. Rosenfeld and J. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM*, 13:471–494, 1966.

[19] A. Rosenfeld and J. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[20] M. G. Voronoi. Nouvelles applications des parametres continus à la theorie des formes quadratiques. *J. Reine Angew. Math.*, 134:198–287, 1908.

[21] H. Yamada. Complete Euclidean distance transformation by parallel operation. In *Proc. 7th Int. Conf. on Pattern Recognit.*, pages 69–71, Montreal, Canada, 1984.

[22] M. Yamashita and T. Ibaraki. Distances defined by neighborhood sequences. *Pattern Recognition*, 19:237–246, 1986.