Information Processing Letters 17 (1983) 219–223
North-Holland 8 November 1983

# ON COMPUTING ALL NORTH–EAST NEAREST NEIGHBORS IN THE $L_1$ METRIC

Leo J. GUIBAS
*Xerox Parc, Palo Alto, CA 94304, U.S.A.*

Jorge STOLFI
*Stanford University, Stanford, CA 94305, U.S.A.*

Communicated by J. Nievergelt
Received 30 March 1983

Given a collection of n points in the plane, we exhibit an algorithm that computes the nearest neighbor in the north–east (first quadrant) of each point, in the $L_1$ metric. By applying a suitable transformation to the input points, the same procedure can be used to compute the $L_1$ nearest neighbor in any given octant of each point. This is the basis of an algorithm for computing the minimum spanning tree of the n points in the $L_1$ metric. All three algorithms run in $O(n \lg n)$ total time and $O(n)$ space.

*Keywords:* Minimum spanning tree, all nearest neighbors, computational geometry, analysis of algorithms

## 1. Introduction

*Closest-point problems* constitute an important area of computational geometry. A typical member of this class is the *all-nearest-neighbors problem*: For each point in a given set, determine which of the other points is closest to it. Another problem that can be included in this class is the computation of a *minimum spanning tree* (MST) for a set of points, that is, a tree with those points as vertices, straight line segments as edges, and total edge length as small as possible.

The answer to those problems depends in general on the particular metric used to define distances and edge lengths; the Euclidean or $L_2$ metric is perhaps the most common. In this paper we will be concerned almost exclusively with the *Manhattan* or $L_1$ metric, in which the distance between two points p and q on the plane is by definition

$$|p_x - q_x| + |p_y - q_y|.$$

Closest-point problems in the $L_1$ metric have several applications. For example, Lee and Wong [5] discuss various heuristics for scheduling head movements in handling I/O requests for secondary storage devices. These heuristics construct approximations to shortest traveling salesman tours in that metric. The minimum spanning tree problem in the $L_1$ metric [2] has significant applications in wire routing on integrated circuits and printed circuit boards, where wires are often constrained to follow the 'Manhattan' geometry.

Efficient algorithms for Euclidean closest-point problems have been known for some time [6], but their extension to $L_1$ and other metrics is generally not trivial. In the published literature all optimal algorithms for $L_1$ closest-point problems are based on a common tool, the *Voronoi diagram in the* $L_1$ *metric*. An $O(n \lg n)$ algorithm for constructing the $L_1$ Voronoi diagram of n points in the plane was given by Lee and Wong [5], and later generalized by Lee [4] to arbitrary $L_p$ metrics. The solution to several closest-point problems, including the MST and all nearest-neighbor pairs, can be obtained from the Voronoi diagram within the same time bound.

Yao [7] gave a different flavor of algorithm for computing minimum spanning trees in coordinate spaces, based on considering for each of the given points a sufficient number of closest neighbors in different directions. However, for the planar case Yao's algorithm runs in $O(n^{2-1/8}(\lg n)^{1-1/8})$ time, which is not as good as the bounds given above. In this paper we show that Yao's ideas admit of a very simple implementation for the plane.[1] The resulting MST algorithm avoids the overhead of constructing the Voronoi diagram, but still runs in time $O(n \lg n)$, and is therefore optimal.

In our method we first show how to compute in time $O(n \lg n)$ the nearest north–east (NE), or first quadrant, neighbor for each point in the given collection. We then modify this technique to give the nearest neighbor in each octant, and establish that the set of edges from each point to its eight nearest neighbors, one in each octant, encompasses an MST.

As a final remark, we note that our algorithm also solves the MST problem in the $L_\infty$ *metric*, defined by

$$d_\infty(p, q) = \max\{|p_x - q_x|, |p_y - q_y|\},$$

since a rotation by 45° and a scaling by a factor of $\frac{1}{2}\sqrt{2}$ transforms $L_\infty$ distances into $L_1$ distances.

## 2. The all NE nearest neighbors problem

In this section we discuss the following problem, which is of interest in its own right: Let n points in the plane be given. We wish to compute, for each of the given points p, its *nearest north–east neighbor*. By that we mean the point in our collection which is in the first quadrant with respect to p, and closest to p. (Two fine points: if more than one nearest neighbor to p exists, we choose one arbitrarily. We also adopt some suitable convention to indicate the case where no point exists at all in the first quadrant of p.)

Let $p_x$ and $p_y$ denote respectively the x and y coordinates of point p, and let

$$d(p, q) = |p_x - q_x| + |p_y - q_y|$$

denote the distance from point p to point q. Our algorithm depends crucially on the following lemma.

**Lemma 1.** *Let* p, q, r *and* s *be four points in the plane such that* p *and* q *are both smaller in* x *than* r *and* s, *and* r *and* s *are both greater in* y *than* p *and* q (*see Fig.* 1). *Then* $d(p, r) \leqslant d(p, s)$ *is equivalent to* $d(q, r) \leqslant d(q, s)$.

**Proof**

$$\begin{aligned} & d(p, r) \leqslant d(p, s) \\ & \Leftrightarrow (r_x - p_x) + (r_y - p_y) \leqslant (s_x - p_x) + (s_y - p_y) \\ & \Leftrightarrow r_x + r_y \leqslant s_x + s_y \\ & \Leftrightarrow (r_x - q_x) + (r_y - q_y) \leqslant (s_x - q_x) + (s_y - q_y) \\ & \Leftrightarrow d(q, r) \leqslant d(q, s). \qquad \square \end{aligned}$$

Using this lemma, we now devise a divide-and-conquer strategy for our problem: Start out by sorting all points in x. (This step is not strictly necessary, as we can use a linear median finding algorithm, but it would be the implementation of choice in practice.) We describe a recursive procedure that not only computes all NE nearest neighbors, but also returns the points sorted in y.

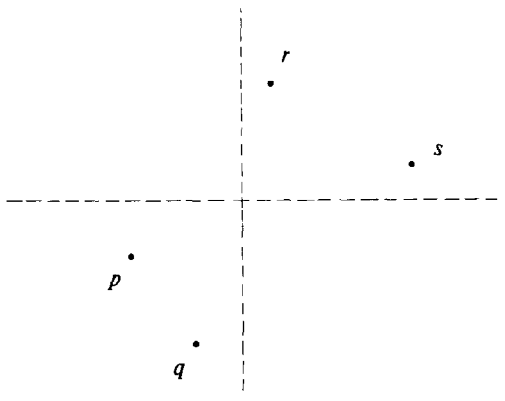

Fig. 1. Lemma 1.

[1] In the $L_1$ metric. Unless explicitly stated otherwise, this qualification applies also to all distances and lengths referred to or implied in the rest of this paper.

We first find an x value that divides our points into the left and right halves, and recursively call our procedure on the two halves. The computed NE neighbors for points on the right half are already the correct NE neighbors for the full problem. The key question is how fast we can update the NE neighbors of points on the left half.

We will show that in O(n) time we can compute for each point on the left half its nearest NE neighbor among the points on the right half. Thus the NE neighbors of the left half points can be updated and the computation completed in O(n) time. (This includes merging the two y-sorted lists of the left and right points into a common sorted list.) By standard analysis of algorithm techniques it follows that the overall computation time will then be O(n lg n).

To accomplish the computation described above we will keep three pointers. One pointer, *left*, will advance down the sorted list of left half points in decreasing y. And the other two, *right* and *min*, will advance down the list of the right hand points in decreasing y. No pointer will ever back up (see Fig. 2). In the normal case we are seeking the nearest NE neighbor of the left hand point indicated by *left*. Pointer *right* is advancing downwards on the list of right hand points, while always staying at points with y coordinates larger than that of *left*. Pointer *min* is keeping track of the nearest right hand element to *left* seen so far. As *right* advances to the next element, one of three things can happen.

(1) The point that *right* points to is higher in y than *left*, but further away than *min*: in this case we do nothing.

(2) The point that *right* points to is higher in y than *left*, but closer than *min*: in this case we set *min* equal to *right*.

(3) The point that *right* points to drops below *left*: in this case we output *min* as the nearest NE neighbor of *left* (among the right points), and advance *left*.

Lemma 1 proves that this iteration correctly computes the nearest NE neighbor of each point on the left among points on the right. Since no pointer ever backs up, the linearity of this pass is immediate.

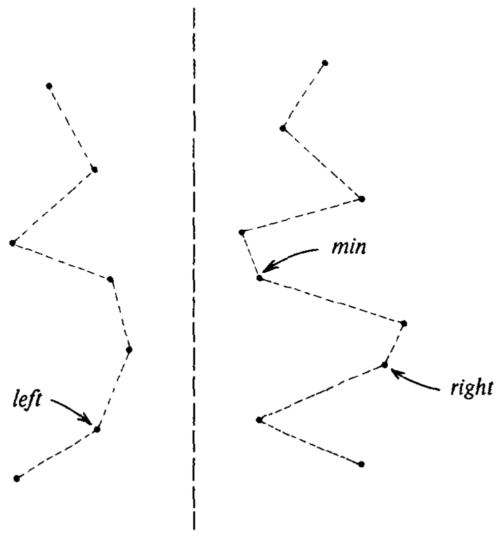

Fig. 2. The merge pass.

For the MST algorithm in Section 3 we will actually need to find the first octant nearest neighbor of each point in our set. This can be done by using the above algorithm after appropriately transforming our set of points. Consider the linear transformation T of the plane defined by

$$T:(x, y) \mapsto ((x-y)/2, y).$$

It is simple to check that point q is in the first octant of point p if and only if Tq is in the first quadrant of Tp. Intuitively, T is a shearing transform parallel to the x axis taking octants into quadrants. Furthermore, if $q_1$ and $q_2$ are two points in the first octant of p and $d(p, q_1) \leqslant d(p, q_2)$, then $d(Tp, Tq_1) \leqslant d(Tp, Tq_2)$, and conversely. This is because $p_x + p_y = 2(Tp_x + Tp_y)$, i.e., T maps lines of slope $-1$ to lines of slope $-1$. Thus nearest first octant neighbors map to nearest first quadrant neighbors under T. Slightly modified linear transformations can be used to map any other octant onto the first quadrant.

## 3. The minimum spanning tree

There exist several minimum spanning tree algorithms for graphs that require no more than O(n lg n) time when the given graph contains no more than O(n) edges. Kruskal's algorithm [3],

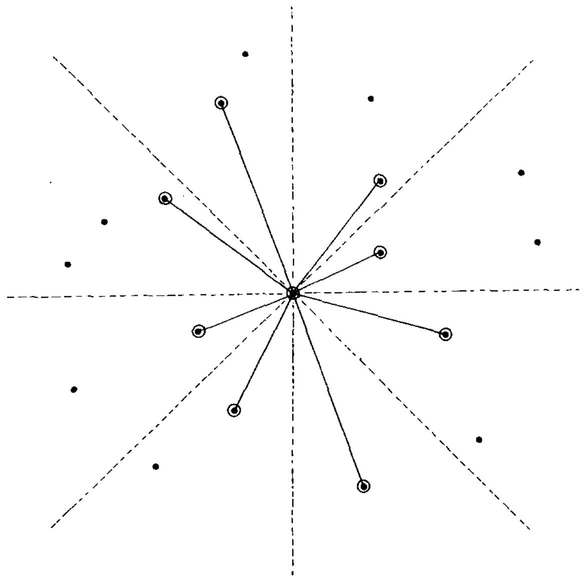

Fig. 3. The octant neighbors.

implemented with an appropriate union-find data structure, gives this bound (see also the extensive discussion in Cheriton and Tarjan [1]). The lemma below asserts that there exists an especially simple set of edges of our n points that is linear in size and contains some MST.

**Lemma 2.** *Let* n *points in the plane be given. Consider the graph* G *obtained by joining every point to each of its eight nearest neighbors, one in each octant, by an edge whose weight is the* $L_1$ *distance between the two* (*see Fig.* 3). *Then an* MST *for* G *is also an* MST *for our point set in the* $L_1$ *metric.*

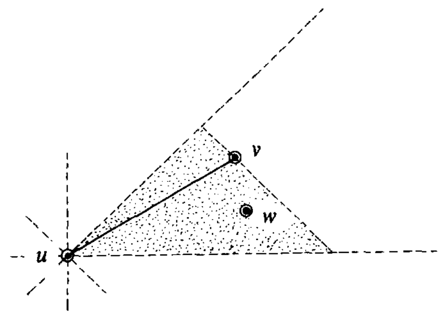

Fig. 4. Lemma 2.

**Proof.** We will show that the edges of some MST for our set of points are contained among the edges of G. We adopt the convention that each octant contains the half-axis bounding it and shared with the clockwise following octant. The value of choosing octants is that if v and w are two nearest neighbors of point u in the same octant, then

$$d(v, w) < d(u, v) = d(u, w).$$

Consider an edge e from point u to point v in some MST of the set of points. We can assume, without loss of generality, that v is in the first octant of u. We now claim that v is a NE nearest neighbor of u (see Fig. 4). For suppose there could be some point w in the first octant of u, strictly closer to it than v. If we remove the edge e from the MST, we will disconnect it into two fragments, one of which must contain w. Now note that $d(u, v) > d(u, w)$ and also $d(u, v) > d(v, w)$, since u is farther away from v than any other point in the shaded area. By joining w to either u or v, according to which of them lies in the fragment not containing w, we obtain a new and smaller MST, which is a contradiction.

If the nearest neighbor of u we chose to include in G was not v, but (say) x, where $x \neq v$, then consider again deleting e from our MST. The fragment in which x falls must contain v, else adding the edge (v, x) would result in a smaller MST, by the observation of the previous paragraph. Thus edge (u, x) can be used instead of e to obtain an equivalent MST. □

As a result of Lemma 2 we have a new MST algorithm that works in O(n lg n) time. We invoke the method of Section 2 eight times to compute the eight nearest octant neighbors of each point. (Actually these computations can be merged into four pairs of two diametrically opposite octants each, since the transformation T that maps the first octant to the first quadrant also maps the fifth octant to the third quadrant, etc.) We then throw all the edges thus produced into Kruskal's algorithm, and obtain our MST.

## 4. Conclusions

We have shown how to compute all NE nearest neighbors of n points in the plane in the $L_1$ metric in O(n lg n) time. This has also given us an algorithm for computing the MST of the n points within the same time bound. It would be of interest to explore whether these ideas extend to higher dimensions.

## References

[1] D. Cheriton and R.E. Tarjan, Finding minimum spanning trees, SIAM J. Comput. 5 (1976) 724-742.

[2] F.K. Hwang, An O($n \log n$) algorithm for rectilinear minimal spanning trees, J. ACM 26 (1979) 177-182.

[3] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Amer. Math. Soc. 7 (1956) 48-50.

[4] D.T. Lee, Two-dimensional Voronoi diagrams in the $L_p$ metric, J. ACM 27 (1980) 604-618.

[5] D.T. Lee and C.K. Wong, Voronoi diagrams in $L_1$ ($L_\infty$) metrics with 2-dimensional storage applications, SIAM J. Comput. 9 (1980) 200-211.

[6] M.I. Shamos, Computational geometry, Ph. D. Thesis, Yale University, 1977.

[7] A.C. Yao, On constructing minimum spanning trees in $k$-dimensional spaces and related problems, SIAM J. Comput. 11 (1982) 721-736.