JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **49,** 204–217 (1998)
ARTICLE NO. PC971421

Time-Optimal Proximity Graph Computations on Enhanced Meshes[1]

Stephan Olariu[2]

Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529-0162

Ivan Stojmenović

Department of Computer Science, University of Ottawa, Ottawa, Ontario K1N 9B4, Canada

and

Albert Y. Zomaya

Parallel Computing Research Lab, Department of Electrical and Electronic Engineering, University of Western Australia, Perth, Western Australia 6907, Australia

In pattern recognition, morphology, and cellular system design one is interested in objects routinely represented by a polygon ***P***. Distance properties among the vertices of P are one of the fundamental descriptors useful in shape analysis and clustering. In order to capture perceptually relevant features of P one associates a variety of proximity graphs with the polygon P. Typically such a proximity graph $G(P)$ has the same vertices as P: two vertices of P are adjacent in $G(P)$ if they satisfy an application-specific predicate.

Our main contribution is to propose time-optimal algorithms for constructing the Euclidian minimum spanning tree, the all-nearest neighbor graph, the relative neighborhood graph, and the symmetric farthest neighbor graph of an n-vertex unimodal polygon. All our algorithms run on meshes enhanced with row and column buses. We begin by establishing a $\Omega(\log n)$ time lower bound for the task of computing the Euclidian minimum spanning tree of an n-vertex unimodal polygon. This lower bound holds for both the CREW-PRAM and for meshes with multiple broadcasting, regardless of the number of processors available.

Next, we show that this time lower bound is tight by exhibiting an algorithm for the Euclidian minimum spanning tree problem running in $\Theta(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. We also show that the all-nearest neighbor graph, the relative neighborhood graph, and the symmetric farthest neighbor graph of an n-vertex unimodal polygon can be computed in $O(1)$ time on a mesh with multiple broadcasting of size $n \times n$. © 1998 Academic Press

Key Words: Cellular system design; pattern recognition; morphology; perception; proximity graphs; unimodal polygons; parallel algorithms; mesh with multiple broadcasting; ANNG; EMST; RNG; SFNG.

[1]Work supported in part by NSF Grant CCR-9522093, ONR Grant N00014-97-1-0526, NATO Grant CRG-950744, NSERC Grant OGPIN007, and ARC Grant 04/15/412/194.

[2]Address for correspondence: Prof. Stephan Olariu, Department of Computer Science, Old Dominion, University, Norfolk, VA 23529-0162. E-mail: `olariu@cs.odu.edu`.

1. INTRODUCTION

Amongst the massively parallel architectures, the mesh has emerged as one of the platforms of choice for solving problems in image processing, computer vision, pattern recognition, handoff management, cellular system design, robotics, and computational morphology, with the number of application domains that benefit from this simple and intuitive architecture growing by the day. Its regular and intuitive topology makes the mesh eminently suitable for VLSI implementation, with several models built over the years. Examples include the ILLIAC IV, the STARAN, the MPP, and the MasPar, among many others [4, 5, 11, 32]. Yet, the mesh is not for everyone: its large computational diameter makes the mesh architecture less attractive in contexts where the computation involves data items spread over processing elements far apart.

To address this shortcoming, the mesh has been enhanced by the addition of various types of bus systems [3, 12, 13, 20, 24, 25, 32]. Early solutions involving the addition of one or more *global buses* shared by all the processors in the mesh, have been implemented on a number of massively parallel machines [4, 12, 18, 32]. Yet another popular way of enhancing the mesh architecture involves endowing every row with its own bus. The resulting architecture is referred to as *mesh with row buses* and has received a good deal of attention in the literature. Recently, a more powerful architecture has been obtained by adding one bus to every row and to every column in the mesh, as illustrated in Fig. 1. In [20] an abstraction of such a system is referred to as *mesh with multiple broadcasting*. The mesh with multiple broadcasting has been implemented in VLSI and is commercially available in the DAP family of multicomputers [21, 31, 34]. In turn, due to its commercial availability, the mesh with multiple broadcasting has attracted a great deal of attention. Applications ranging from image processing [21, 31, 34], to visibility and robotics [9, 30], to digital geometry and pattern recognition [7, 9, 20, 23, 30], to optimization [14], to other basic problems [3, 6, 8, 13] have been reported on this platform and some of its variants [23].

In essence, a mesh with multiple broadcasting of size $\sqrt{n} \times \sqrt{n}$, hereafter also referred to as a mesh if no confusion is possible, consists of n identical SIMD processors positioned on a square array, with the processors in the first column also serving as I/O ports.

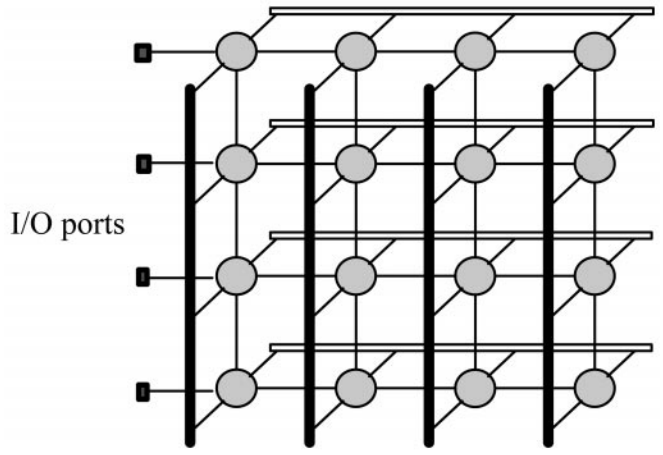

FIG. 1. A mesh with multiple broadcasting of size 4×4.

The processor located in row i and column j, $(1 \leq i, j \leq n)$, is referred to as $P(i, j)$. As customary, we assume that processors know their coordinates within the mesh and have a *constant* number of registers of size $O(\log n)$. In unit time, each processor performs an arithmetic or boolean operation, communicates with one of its neighbors using a local connection, broadcasts a value on a bus or reads a value from a specified bus. Each of these operations involves handling at most $O(\log n)$ bits of information. For practical reasons, only one processor is allowed to broadcast on a given bus at any one time. By contrast, all the processors on the bus can simultaneously read the value being broadcast. In accord with other researchers [3, 20, 25, 31], we assume that communications along buses take $O(1)$ time. Although inexact, recent experiments with the DAP, the PPA, and the YUPPIE multiprocessor array systems seem to indicate that this is a reasonable working hypothesis [21, 31, 34].

A PRAM [19] consists of synchronous processors, all having unit-time access to a shared memory. At each step, every processor performs the same instruction, with a number of processors masked out. In the CREW-PRAM, a memory location can be simultaneously accessed in reading but not in writing. A mesh with multiple broadcasting can be perceived as a restricted version of the CREW-PRAM: the buses are nothing more than *oblivious* concurrent read, exclusive write registers with the access restricted to certain sets of processors. Indeed, a square mesh with multiple broadcasting using p^2 processors can be viewed as a CREW-PRAM with p^2 processors where groups of p of these have concurrent read access to a register whose value is available for one time unit, after which it is lost. Given that the mesh with multiple broadcasting is, in this sense, *weaker* than the CREW-PRAM, it is very often quite a challenge to design algorithms in this model that match the performance of their CREW-PRAM counterparts. Typically, for the same running time, the mesh with multiple broadcasting uses more processors. This phenomenon will appear in our algorithms.

In pattern recognition and classification the shape of an object is routinely represented by a polygon obtained from an image processing device [2, 17, 38]. A class of fundamental features that contribute to a morphological description useful in shape analysis are the (Euclidian) distance properties among vertices of the polygon. Nearest- and furthest-neighbor computations are central to pattern recognition classification techniques, image processing, computer graphics, and computational geometry [16, 17, 33, 37]. In image processing, for example, proximity is a simple and important metric for potential similarities of objects in the image space. In pattern recognition, the same concept appears in clustering and computing similarities between sets [17]. In computational geometry, closeness is often a valuable tool in devising efficient algorithms for a number of seemingly unrelated problems [33].

A classic problem in this domain involves computing for every point in a given set S, a point that is closest to it: this problem is known as the all-nearest neighbor problem and has been well studied in the literature [16, 17, 30, 33, 37]. A class of related problems involves associating a certain graph with the set S. This graph is, of course, application specific. For example, in pattern recognition one is interested in the Euclidian minimum spanning tree (EMST) of S, the all-nearest neighbor graph (ANNG), the relative neighborhood graph (RNG) of S, the symmetric farthest neighbor graph (SFNG) of S, the Gabriel graph (GG) of S, and the Delaunay graph (DG) of S, to name a few [16, 33, 35–38].

The main contribution of this work is to provide time-optimal algorithms to construct the EMST, the ANNG, the RNG, and the SFNG of an n-vertex unimodal polygon. We begin by establishing a $\Omega(\log n)$ time lower bound for the task of computing the EMST of an n-vertex unimodal polygon. This time lower bound holds for both the CREW-PRAM and for the mesh with multiple broadcasting, even if an infinite number of processors are available. Next, we show that the bound is tight by exhibiting an EMST algorithm running in $\Theta(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. We also show that the ANNG, the RNG, and the SFNG of an n-vertex unimodal polygon can be computed in $O(1)$ time on a mesh with multiple broadcasting of size $n \times n$.

The remainder of the work is organized as follows: Section 2 discusses basic terminology and algorithms that are key ingredients of our subsequent sections; Section 3 presents our time lower bound arguments; Section 4 gives the details of the proposed time-optimal algorithms for the ANNG, RNG, EMST, and SFNG problems; finally, Section 5 offers concluding remarks and poses a number of open questions.

2. BACKGROUND AND TERMINOLOGY

Specifying an n-vertex polygon P in the plane amounts to enumerating its vertices in *clockwise* order as $p_1, p_2, \ldots, p_n$, $(n \geq 3)$, in such a way that $p_i p_{i+1}$, $(1 \leq i \leq n-1)$, and $p_n p_1$ define the edges of P. This representation is known as *vertex* representation of P. The vertex representation of a polygon can be easily converted into an *edge* representation where P is represented by a sequence $e_1, e_2, \ldots, e_n$ of edges, specified in clockwise order, with e_i, $(1 \leq i \leq n-1)$, having p_i and p_{i+1} as its endpoints and e_n having p_n and p_1 as its endpoints. A polygon P is termed *convex* if all its diagonals lie entirely within P.

For definiteness, we now discuss a number of important representational issues for the objects we manipulate. Specifically, a polygon P is represented by a linked list stored in some order in an array $P[1 \ldots n]$. Occasionally, we shall find it convenient to perceive the polygon P as a graph, also denoted by P, whose vertices are the vertices of P and whose edges are precisely the edges of P. Since a number of different graphs will be considered in this paper, it is appropriate to clarify how these graphs are specified. As customary [19], a graph G will be specified by a linked list of edges i.e., *unordered* pairs of vertices, stored in some order in an array G. This representation facilitates a number of cardinality queries about the graph as well as the assignment of processors to the edges of G.

It is customary to call a real function $f\colon \{0, 1, \ldots, n-1\} \to R$ *unimodal* if there exists an integer i, $(0 \leq i \leq n-1)$, such that the function f is strictly increasing in the interval $[0, i]$ and strictly decreasing in the interval $[i, n-1]$. A vertex p_i of a polygon P is said to be *unimodal* if the Euclidian distance function $d(p_i, p_j)$ from vertex p_i to vertex p_j, $i \neq j$, of P is unimodal: in other words, the sequence $d(p_i, p_{i+1})$, $d(p_i, p_{i+2}), \ldots, d(p_i, p_{i-1})$ is unimodal. The polygon P itself is termed unimodal if all its vertices are unimodal. For an illustration, we refer the reader to Fig. 2.

Traditionally, convexity has played a central role in analyzing relevant features of the shape of a set of points [2, 17, 33]. Recently, Toussaint [37] pointed out that the notions of convexity and unimodality are quite different: convex polygons need not be unimodal, and unimodal polygons need not be convex. Furthermore, in [37] it is argued convincingly

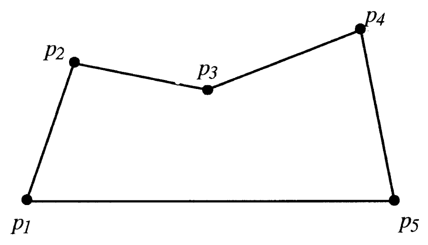

FIG. 2. A unimodal polygon.

that the key factor for obtaining very efficient algorithms for a large number of problems in pattern recognition and morphology is not convexity, but rather unimodality. It is not surprising, therefore, that unimodality and multimodality have received considerable attention in the literature [1, 26–28, 36–38].

The *Euclidian minimum spanning tree* of a set S of points in the plane is a minimum spanning tree of the complete graph whose vertices are the points in S and whose edges are weighted by the Euclidian distance between the corresponding points. The *all-nearest neighbor graph* of a set S of points has the points in S as vertices. Vertices p and q are joined by an edge in ANNG if p (resp., q) is the nearest neighbor of q (resp., p) in S. We note that the nearest neighbor relation is not symmetric. It is quite possible that q is the nearest neighbor of p but not conversely.

The *relative neighborhood graph* of a set S of points in the plane was introduced by Toussaint [37] in an effort to capture many perceptually relevant features of the set S. Specifically, given a set S of points in the plane, RNG(S) has for vertices the points of S together with an edge between p and q if $d(p, q) \leq \max_{s \in S}\{d(p, s), d(q, s)\}$. An equivalent definition states that two vertices p, q are joined by an edge in RNG(S) if no point s of S lies inside LUNE(p, q), the "lune of influence" of p, q defined as the intersection of two open disks each of radius $d(p, q)$ centered at p, q, respectively, as illustrated in Fig. 3.

Given a set S a points in the plane, the points p and q of S are a *symmetric farthest neighbor pair* if both p and q are furthest from each other among the points of S, i.e., all other points lie inside LUNE(p, q). Put differently,

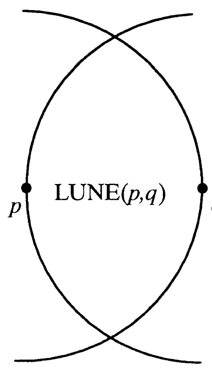

FIG. 3. Illustrating LUNE(p, q).

$$d(p, q) = \max_{s \in S}\{d(p, s)\} \text{ and } d(q, p) = \max_{s \in S}\{d(q, s)\}.$$

The *symmetric farthest neighbor graph* associated with S has the points of S as vertices. Two vertices p and q are connected by an edge if and only if p and q are a symmetric furthest pair [38]. For an illustration of the EMST, ANNG, RNG, and SFNG of a set S of points in the plane the reader is referred to Fig. 4a–e.

The *prefix computation* problem has turned out to be one of the basic techniques in parallel processing, being a key ingredient in many algorithms. The problem is stated as follows: given an associative binary operation $\circ$ and a sequence $a_1, a_2, \ldots, a_n$ of items, compute all the sums of the form $a_1, a_1 \circ a_2, a_1 \circ a_2 \circ a_3, \ldots, a_1 \circ a_2 \circ \ldots \circ a_n$.

PROPOSITION 2.1 [20, 29]. *The prefix sums* (*also maxima or minima*) *of a sequence of n real numbers stored in one row of a mesh with multiple broadcasting of size $n \times n$ can be computed in $\Theta(\log n)$ time. Furthermore, this is time optimal.*

The convex hull of a set of points in the plane is defined as the smallest convex set that contains the original set [33]. Our arguments rely, in part, on the following result proved in [29].

PROPOSITION 2.2 [29]. *The convex hull of planar set of n points stored in the first row of a mesh with multiple broadcasting of size $n \times n$ can be computed in $\Theta(\log n)$ time. Furthermore, this is time optimal.*

Vertices p_i and p_j of a convex polygon P are termed *antipodal* if P admits parallel supporting lines through p_i and p_j. The *diameter* of a convex polygon [33] is the largest Euclidian distance between any pair of its vertices. A classic result [33] asserts that the diameter of a convex polygon is the largest distance between antipodal pairs.

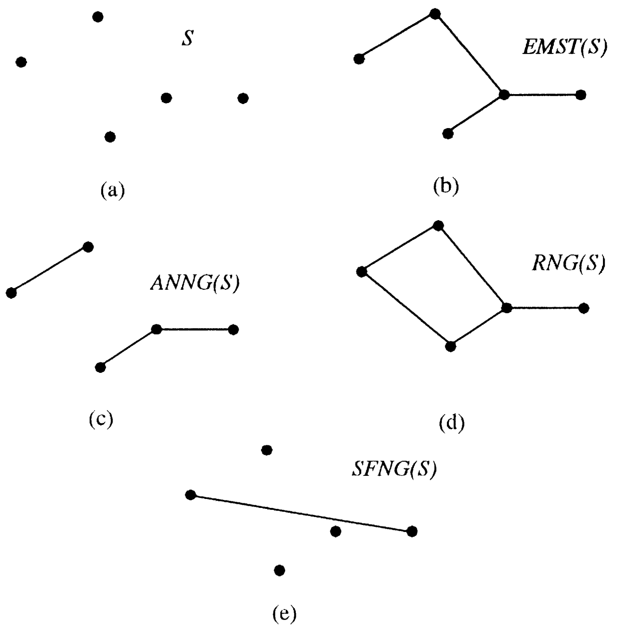

FIG. 4. Illustrating the EMST, ANNG, RNG, and SFNG graphs of a set of points.

Recently, [8] showed that all the antipodal pairs of a convex polygon as well as any semigroup computations involving these antipodal pairs can be determined efficiently. More precisely, the following result was proved in [8].

PROPOSITION 2.3 [8]. *Let P be an n-vertex convex polygon stored one vertex per processor in one row of a mesh with multiple broadcasting of size $n \times n$. Any semigroup computation involving the antipodal pairs of P can be performed in $\Theta(\log n)$ time. Furthermore, this is time optimal.*

3. THE LOWER BOUND

The purpose of this section is to establish a time lower bound for the following proximity graph problems.

EMST: given an n-vertex unimodal polygon compute its Euclidian minimum spanning tree.

Our lower bound arguments will be stated first in the CREW-PRAM. This approach is motivated by a recent result of Lin *et al.* [22] that allows us to extend many lower bound results from the CREW-PRAM to meshes with multiple broadcasting.

For further reference we now state a fundamental result of Cook *et al.* [15].

PROPOSITION 3.1 [15]. *The time lower bound for computing the logical OR of n bits on the* CREW-PRAM *is* $\Omega(\log n)$, *regardless of the number of processors and memory cells available.*

In addition, we shall rely on the following recent result of Lin *et al.* [22].

PROPOSITION 3.2. *Any computation that takes $O(t(n))$ steps on an n-processor mesh with multiple broadcasting can be performed in $O(t(n))$ steps on an n-processor* CREW-PRAM.

It is important to note that Proposition 3.2 guarantees that if $T_M(n)$ is the execution time of an algorithm for solving a given problem on an n-processor mesh with multiple broadcasting, then there exists a CREW-PRAM algorithm to solve the same problem in $T_P(n) = T_M(n)$ time using n processors. In other words, "too fast" an algorithm on the mesh with multiple broadcasting implies "too fast" an algorithm for the CREW-PRAM. This observation is exploited in [22] to transfer known computational lower bounds for the PRAM to the mesh with multiple broadcasting.

We show that the time lower bound for EMST is $\Omega(\log n)$ on the CREW-PRAM, by reducing the OR problem to EMST. For this purpose, let $c_1, c_2, \ldots, c_n$ be an arbitrary input to the OR problem. Consider a circle C and let $b_1, b_2, \ldots, b_{2n}$ be equally spaced points on the boundary of C. It is easy to construct C such that the points are one unit apart. Associate with each input bit c_i, $(1 \leq i \leq n)$, the pair of points b_i and b_{n+i} on C. If $c_i = 1$ then perturb the points b_i and b_{n+i} by a small amount ϵ clockwise as illustrated in Fig. 5. Let P be the resulting polygon with its vertices denoted in clockwise order by $b_1, b_2, \ldots, b_{2n}$. It is easy to see that P is unimodal. Fig. 5 illustrates this construction for $n = 6$ and the input sequence 0, 0, 1, 0, 0, 0.

If $c_1 = 1$ then the answer to OR is 1 and we are done. Otherwise, consider any algorithm that correctly returns the EMST of the polygon P. Clearly, EMST has $2n - 1$ edges. In $O(1)$ time n processors can identify the *unique* edge of P that does not belong

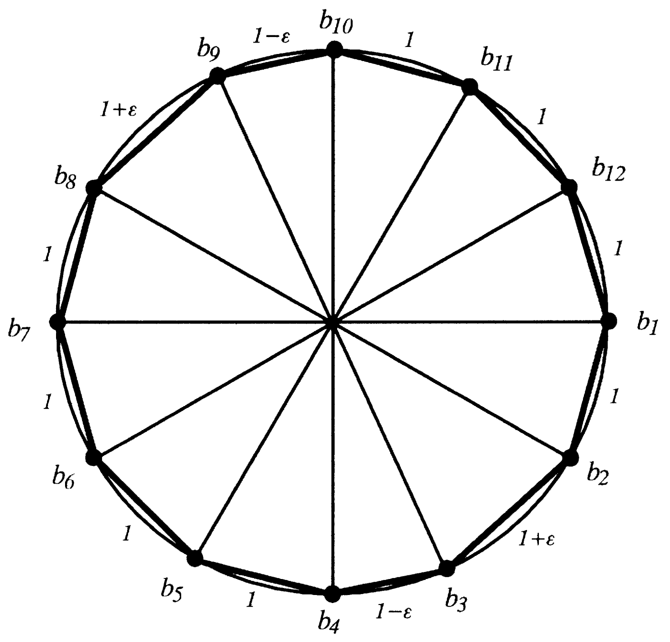

FIG. 5. Illustrating the lower bound argument for EMST.

to the EMST. Now the answer to OR is 1 if and only if the length of this edge is larger than 1. Since the construction of P takes $O(1)$ time using n processors on the CREW-PRAM, Proposition 3.1 implies the following result.

THEOREM 3.3. *The task of computing the Euclidian minimum spanning tree of an n-vertex unimodal polygon has a time lower bound of* $\Omega(\log n)$ *on the* CREW-PRAM, *regardless of the number of processors and memory cells available.*

It is important to note that the polygon P used in the reduction above is always convex, in addition to being unimodal. Therefore, we have the following result which is of an independent interest.

THEOREM 3.4. *The task of computing the Euclidian minimum spanning tree of an n-vertex convex polygon has a time lower bound of* $\Omega(\log n)$ *on the* CREW-PRAM, *regardless of the number of processors and memory cells available.*

By virtue of Proposition 3.2 and of Theorem 3.3, combined, we have the following important result.

THEOREM 3.5. *The task of constructing the* EMST *of an n-vertex unimodal polygon has a time lower bound of* $\Omega(\log n)$ *on a mesh with multiple broadcasting of size* $n \times n$.

4. THE ALGORITHMS

The purpose of this section is to design time-optimal algorithms to compute the ANNG, the RNG, the EMST, and the SFNG of an n-vertex unimodal polygon $P = p_1, p_2, \ldots, p_n$ stored in the first row of a mesh with multiple broadcasting of size $n \times n$. We let ANNG(P), RNG(P), EMST(P), and SFNG(P) stand for the proximity graphs we wish to compute.

Quite recently, a rather complicated algorithm for computing the ANNG of an arbitrary set of points in the plane was proposed in [30]. To solve the ANNG problem for unimodal polygons we rely on the following simple fact whose proof follows directly from definition.

LEMMA 4.0. *Let P be a unimodal polygon. Every edge is adjacent to its nearest neighbor in P.*

Note that Lemma 4.0 suggests an obvious $O(1)$ time algorithm for constructing the ANNG(P) of P. This algorithm only involves communications between adjacent processors. As a result, if the unimodal polygon is available in a linear array of n processors, the corresponding instance of the ANNG problem can be solved in $O(1)$ time.

Recently, Olariu [26] developed simple linear-time sequential algorithms to compute the EMST and RNG of a unimodal polygon. To make this work self-contained we now recall a number of relevant properties of unimodal polygons established in [26].

PROPOSITION 4.1. *If a diagonal of P is an edge of* RNG(P) *then its endpoints belong to the two longest edges of P.*

Call an edge e of RNG(P) *P-critical* if $e \notin P$. It is worth noting that Proposition 4.1 asserts that if $p_i p_j$ is P-critical, then p_i and p_j must be endpoints of the two longest edges in P. A natural question to ask is: "How many P-critical edges can RNG(P) contain?" The answer to this question is given by the following result.

PROPOSITION 4.2. *If P is a unimodal polygon, then* RNG(P) *contains at most one P-critical edge.*

An important consequence of Proposition 4.2 is the following.

COROLLARY 4.3. *If P is an n-vertex unimodal polygon, then* EMST(P) *shares at least $n-2$ edges with P.*

The following result makes Corollary 4.3 more precise, by characterizing the diagonals of P that can belong to EMST(P).

PROPOSITION 4.4 [26]. *A diagonal of P is an edge in* EMST(P) *if and only if it is a P-critical edge in* RNG(P).

Figure 6 features a unimodal polygon $P = x, u, w, v$. As easily seen, LUNE(u, v) is empty and so uv is an edge in RNG(P). By Proposition 4.4, uv is also an edge in EMST(P). We begin by showing that given an n-vertex unimodal polygon $P = p_1, p_2, \ldots, p_n$, stored one vertex per processor in the first row of a mesh with multiple broadcasting of size $n \times n$, the task of detecting the edges of P that do not belong to RNG(P) can be performed in $O(1)$ time. For definiteness, assume that for every i, $1 \leq i \leq n$, processor $P(1, i)$ stores the vertex p_i. Using the vertical buses we replicate the contents of the first row in all the rows of the platform. Next, for every i, $1 \leq i \leq n-1$, processor $P(i, i)$ broadcasts the edge $p_i p_{i+1}$ to all the processors in row i. Similarly, processor $P(n, n)$ broadcasts the edge $p_n p_1$ throughout row n. Every processor in row i that detects that the point it stores in inside LUNE(p_i, p_{i+1}) sets a local register to 1. All other processors set the register to 0. The following technical result is key for our $O(1)$ time algorithm for computing the RNG(P).

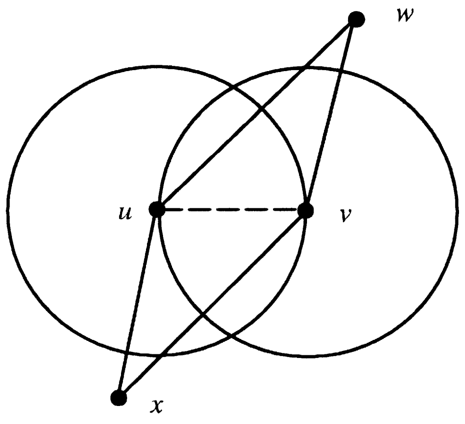

FIG. 6. Illustrating the concept of a *P*-critical edge.

LEMMA 4.5. *For every row i, the set of processors that store a* 1 *form an interval.*

Proof. Suppose not and refer to Fig. 7. We find an edge $p_i p_{i+1}$ of the polygon and two vertices p_j and p_k such that $j < k$, p_j and p_k belong to LUNE(p_i, p_{i+1}) while p_{j+1} does not belong to LUNE(p_i, p_{i+1}). However, now we contradict that P is unimodal, since $d(p_{i+1}, p_j) < d(p_{i+1}, p_{j+1})$, $d(p_{i+1}, p_{j+1}) > d(p_{i+1}, p_k)$, and $d(p_{i+1}, p_k) < d(p_{i+1}, p_i)$. This completes the proof of the lemma. ■

By virtue of Lemma 4.5, the *leftmost* processor in row i that stores a vertex that lies inside LUNE(p_i, p_{i+1}) can be detected in $O(1)$ time. In turn, this processor informs $P(i, i)$ that the edge $p_i p_{i+1}$ cannot belong to RNG(P). To summarize our findings we state the following result.

LEMMA 4.6. *The task of detecting the edges of P that do not belong to* RNG(P) *can be performed in $O(1)$ time.*

We are now in a position to spell out the details of the algorithm to compute the RNG of a unimodal polygon P.

ALGORITHM. Compute-RNG(P);

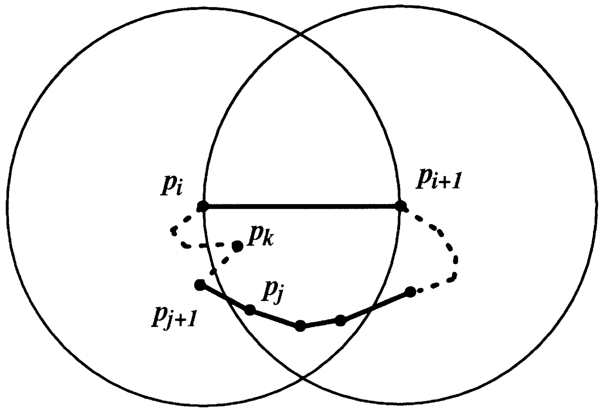

FIG. 7. Illustrating the proof of Lemma 4.5.

Step 1. Determine the edges of P that do not belong to RNG(P). If no such edge exists then return(P);

Step 2. Let $p_i p_{i+1}$ and $p_j p_{j+1}$ be the two edges of P that do not belong to RNG(P); let $p_r p_s$ be the critical edge of the polygon determined by $p_i,\ p_{i+1},\ p_j,\ p_{j+1}$. Return$(P \cup \{p_r p_s\} \setminus \{p_i p_{i+1},\ p_j p_{j+1}\})$.

THEOREM 4.7. *Algorithm Compute*-RNG *returns* RNG(P) *of an n-vertex unimodal polygon in* $O(1)$ *time on a mesh with multiple broadcasting of size* $n \times n$.

Proof. By Lemma 4.6, Step 1 can be executed in $O(1)$ time. If all the edges of P belong to RNG(P) then, at the end of Step 1 we exit with the correct answer.

On the other hand, if Step 2 is executed then by Proposition 4.2 we know that RNG(P) has exactly one critical edge. Moreover, this edge is determined by the endpoints of the two edges of P that do not belong to RNG(P). This is precisely what Step 2 is computing. Finally, the task of compacting the edges of RNG(P) into the leftmost positions in the first row of the mesh is done in the obvious way: $p_i p_{i+1}$ is replaced with $p_r p_s$. All the edges to the left of $p_j p_{j+1}$ are to move one position to the left. Clearly, the compaction phase of Step 2 can be performed in $O(1)$ time. This completes the proof of the theorem.

Next, we present an algorithm to compute the EMST(P) of an n-vertex unimodal polygon $P = p_1,\ p_2,\ \ldots,\ p_n$.

ALGORITHM. Compute-EMST(P);

Step 1. Determine the longest and second-longest edges of P and let them be $p_i p_{i+1}$ and $p_j p_{j+1}$;

Step 2. Let $p_r p_s$ be any critical edge of the polygon determined by $p_i,\ p_{i+1},\ p_j,\ p_{j+1}$. If such a critical edge exists then return$(P \cup \{p_r p_s\} \setminus \{p_i p_{i+1},\ p_j p_{j+1}\})$.

Step 3. On the other hand, if such a critical edge does not exist, then EMST(P) is obtained from P by removing the edge $p_i p_{i+1}$. Return$(P \setminus \{p_i p_{i+1}\})$.

THEOREM 4.8. *Algorithm Compute*-EMST *returns* EMST(P) *of an n-vertex unimodal polygon in* $\Theta(\log n)$ *time on a mesh with multiple broadcasting of size* $n \times n$. *Furthermore, this is time optimal on this architecture.*

Proof. The correctness of this simple algorithm is guaranteed by a result in [26]. By Proposition 2.1, computing the longest and the second-longest edges of P takes $O(\log n)$ time. The task of testing whether or not LUNE$(p_i,\ p_{i+1})$ is empty takes $O(1)$ time. Thus, the running time is bounded by $O(\log n)$ which is time-optimal by Theorem 3.5. ■

Let us outline the idea of our algorithm to compute the SFNG of an n-vertex unimodal polygon P pretiled one vertex per processor in the first row of a mesh with multiple broadcasting of size $n \times n$ such that for every i, $(1 \leq i \leq n)$, processor $P(1,\ i)$ stores vertex p_i. By using the vertical buses, the polygon P is replicated in all the rows of the platform. In every row i, $(1 \leq i \leq n)$, processor $P(i,\ i)$ broadcasts p_i horizontally. Every processor $P(i,\ j)$ in row i computes the Euclidian distance between p_i and p_j. Since the polygon is unimodal, exactly one processor, say $P(i,\ k)$, detects that it stores the vertex p_k which is farthest away from p_i. Next, $P(i,\ k)$ broadcasts k to all the processors in row i. This is tantamount to informing every processor in this row that p_k is the farthest neighbor of p_i. Further, $P(i,\ i)$ broadcasts k to $P(k,\ i)$. Clearly, p_i and

p_k are a symmetric farthest neighbor pair only if $P(k, i)$ stores both p_i and p_k as a result of the previous broadcasts. The details follow.

ALGORITHM. Compute-SFNG(P);

Step 1. Using vertical buses, every processor $P(1, i)$, $(1 \leq i \leq n)$, broadcasts the vertex p_i to the entire column i;

Step 2. Every processor $P(i, i)$, $(1 \leq i \leq n)$, broadcasts p_i to the entire row i;

Step 3. Every processor $P(i, j)$, $(1 \leq i \neq j \leq n)$, computes the Euclidian distance $d(p_i, p_j)$; in every row i, $(1 \leq i \leq n)$, exactly one processor, say, $P(i, k)$ detects that p_k is the farthest neighbor of p_i;

Step 4. For every i, $(1 \leq i \leq n)$, the processor $P(i, k)$ that stores the farthest neighbor of p_i broadcasts k to the entire row i. Every processor in row i stores k in a local register R_1;

Step 5. Every processor $P(i, i)$, $(1 \leq i \leq n)$, broadcasts k, the index of the farthest neighbor of p_i, vertically to processor $P(k, i)$ who stores the value received in a local register R_2;

Step 6. Every processor $P(k, i)$, $(1 \leq i \neq k \leq n)$, for which the contents of its registers R_1 and R_2 are i and k, respectively, identifies (p_i, p_k) as a symmetric farthest neighbor pair and broadcasts this information to processor $P(k, 1)$ in the first row of the mesh.

THEOREM 4.9. *The task of computing the* SFNG *of an n-vertex unimodal polygon can be performed in $O(1)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is optimal in the sense that $\Omega(n^2)$ processors are necessary to guarantee a constant-time performance.*

Proof. The correctness of the algorithm being easy to see, we turn to the complexity. Clearly, all the steps in the algorithm involve simple broadcasting and arithmetic/logic operations and run in $O(1)$ time.

To argue for the processor optimality, let n be even and consider a regular n-gon stored in the first row of the platform such that for every i, $(1 \leq i \leq n)$, processor $P(1, i)$ stores vertex p_i. Clearly, the symmetric farthest neighbor of every vertex p_i, $(1 \leq i \leq n)$, is the vertex $p_{n/2+i}$. However, information about this vertex must cross from the right half of the mesh into the left half. Since $\Omega(n)$ pieces of information must cross this “boundary,” an $O(1)$ time performance requires a platform with $\Omega(n^2)$ processors, as claimed. ■

5. CONCLUSIONS AND OPEN PROBLEMS

In this work we have addressed the problems of computing the ANNG, the RNG, the EMST, and the SFNG of an n-vertex unimodal polygon. We have shown a time-lower bound of $\Omega(\log n)$ for the task of computing the EMST of an n-vertex unimodal polygon. This lower bound holds for the CREW-PRAM as well as the mesh with multiple broadcasting. We have shown that this time lower bound is tight by exhibiting a matching EMST algorithm. In addition, the task of computing the ANNG, the RNG, and the SFNG of a unimodal polygon can be computed in $O(1)$ time.

It is well known that EMST $\subseteq$ RNG $\subseteq$ GG $\subseteq$ DT [33]. We can prove $\Omega(\log n)$ time lower bounds for the problem of constructing the GG and DT of an n-vertex unimodal polygon on a mesh with multiple broadcasting of size $n \times n$. Are these bounds tight?

Our time lower bound result shows that the above problems cannot be solved faster in the computational model that we adopted. We have shown that the $O(1)$ time resolution of the SFNG problem requires $\Omega(n^2)$ processors and that the ANNC can be solved in $O(1)$ time with $\Omega(n)$ processors. It would be interesting to know whether the remaining problems can be solved with fewer than n^2 processors. We pose this as an open problem.

Recently, several authors have shown that rectangular meshes sometimes yield faster algorithms than square meshes. It would be interesting to see whether this is also the case for nearest and farthest neighbor computations.

ACKNOWLEDGMENT

The authors thank four anonymous referees for their constructive comments and suggestions that have greatly contributed to enhancing the quality of this paper.

REFERENCES

1. A. Aggarwal and R. C. Melville, Fast computation of the modality of polygons, *J. Algorithms* **7** (1986), 369–381.
2. D. H. Ballard and C. M. Brown, "Computer Vision," Prentice-Hall, Englewood Cliffs, NJ, 1982.
3. A. Bar-Noy and D. Peleg, Square meshes are not always optimal, *IEEE Trans. Comput. C* **40** (1991), 196–204.
4. K. E. Batcher, Design of massively parallel processor, *IEEE Trans. Comput. C* **29** (1980), 836–840.
5. K. E. Batcher, STARAN parallel processor system hardware, *in* "Proceedings of the 1974 NCC," pp. 405–410, AFIPS, Montvale, NJ, 1974.
6. D. Bhagavathi, P. J. Looges, S. Olariu, J. L. Schwing, and J. Zhang, A fast selection algorithm on meshes with multiple broadcasting, *IEEE Trans. Parallel Distrib. Systems* **5** (1994), 772–778.
7. D. Bhagavathi, S. Olariu, W. Shen, and L. Wilson, A time-optimal multiple search algorithm on enhanced meshes, with applications, *J. Parallel Distrib. Comput.* **22** (1994), 113–120.
8. D. Bhagavathi, S. Olariu, J. L. Schwing, W. Shen, L. Wilson, and J. Zhang, Convexity problems on meshes with multiple broadcasting, *J. Parallel Distrib. Comput.* **27** (1995), 142–156.
9. D. Bhagavathi, V. Bokka, H. Gurla, S. Olariu, J. L. Schwing, and I. Stojmenović, Time-optimal visibility-related problems on meshes with multiple broadcasting, *IEEE Trans. Parallel Distrib. Systems* **6** (1995), 687–703.
10. D. Bhagavathi, H. Gurla, S. Olariu, J. Schwing, and J. Zhang, Square meshes are not optimal for convex hull computation, *IEEE Trans. Parallel Distrib. Systems* **7** (1996), 545–554.
11. T. Blank, The MasPar MP-1 architecture, *in* "Proc. IEEE Computer Conference," pp. 20–24, San Francisco, February 1990.
12. S. Bokhari, On the mapping problem, *IEEE Trans. Comput. C* **30** (1981), 207–214.
13. Y. C. Chen, W. T. Chen, G. H. Chen, and J. P. Sheu, Designing efficient parallel algorithms on mesh connected computers with multiple broadcasting, *IEEE Trans. Parallel Distrib. Systems* **1** (1990), 241–246.
14. Y. C. Chen, W. T. Chen, and G.-H. Chen, Efficient median finding and its application to two-variable linear programming on mesh-connected computers with multiple broadcasting, *J. Parallel Distrib. Comput.* **15** (1992), 79–84.
15. S. A. Cook, C. Dwork, and R. Reischuk, Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM J. Comput.* **15** (1986), 87–97.

16. D. W. Dearholt and M. J. Lipman, “Proc. of Third Workshop on Proximity Graphs,” Technical Report MSU-960923, Mississippi State University, September 1996.

17. R. O. Duda and P. E. Hart, “Pattern Classification and Scene Analysis,” Wiley, New York, 1973.

18. H. Gurla, Leftmost-one computation on meshes with row broadcast, *Inform. Process. Lett.* **47** (1993), 261–266.

19. J. JáJá, “An Introduction to Parallel Algorithms,” Addison-Wesley, Reading, MA, 1991.

20. V. P. Kumar and C. S. Raghavendra, Array processor with multiple broadcasting, *J. Parallel Distrib. Comput.* **2** (1987), 173–190.

21. V. P. Kumar and D. I. Reisis, Image computations on meshes with multiple broadcast, *IEEE Trans. Pattern Anal. Mach. Intelligence* **11** (1989), 1194–1201.

22. R. Lin, S. Olariu, J. L. Schwing, and J. Zhang, Simulating enhanced meshes, with applications, *Parallel Process. Lett.* **3** (1993), 59–70.

23. R. Lin, S. Olariu, and J. L. Schwing, An efficient VLSI architecture for digital geometry, *in* “Proc. of IEEE Conference on Application-Specific Array Processors,” pp. 392–403, San Francisco, August 1994.

24. M. Maresca, Polymorphic processor arrays, *IEEE Trans. Parallel Distrib. Systems* **4** (1993), 490–506.

25. M. Maresca and H. Li, Connection autonomy and SIMD computers: A VLSI implementation, *J. Parallel Distrib. Comput.* **7** (1989), 302–320.

26. S. Olariu, A simple linear-time algorithm for computing the RNG and MST of unimodal polygons, *Inform. Process. Lett.* **7** (1989), 243–247.

27. S. Olariu, On the unimodality of convex polygons, *Inform. Process. Lett.* **29** (1988), 289–292.

28. S. Olariu, The morphology of convex polygons, *Comput. Math. Appl.* **24** (1992), 59–68.

29. S. Olariu, J. L. Schwing, and J. Zhang, Optimal convex hull algorithms on enhanced meshes, *BIT* **33** (1993), 396–410.

30. S. Olariu and I. Stojmenović, Time-optimal proximity problems on meshes with multiple broadcasting, *J. Parallel Distrib. Comput.* **36** (1996), 144–155.

31. D. Parkinson, D. J. Hunt, and K. S. MacQueen, The AMT DAP 500, *in* “33rd IEEE Comp. Soc. International Conf.,” pp. 196–199, 1988.

32. J. L. Potter, “The Massively Parallel Processor,” MIT Press, Cambridge, MA, 1985.

33. F. P. Preparata and M. I. Shamos, “Computational Geometry–An Introduction,” Springer-Verlag, Berlin, 1988.

34. S. F. Reddaway, A. Wilson, and A. Horn, Fractal graphics and image compression on a SIMD processor, *in* “Proc. 2nd Symposium on Frontiers of Massively Parallel Computation,” pp. 265–274, Fairfax, Virginia, February 1988.

35. K. J. Supowit, The relative neighborhood graph with an application to minimum spanning trees, *J. Assoc. Comput. Mach.* **30** (1983), 428–448.

36. G. T. Toussaint, The relative neighborhood graph of a finite planar set, *Pattern Recogn.* **12** (1980), 261–268.

37. G. T. Toussaint, Complexity, convexity and unimodality, *Internat. J. Comput. Inform. Sci.* **13** (1984), 197–217.

38. G. T. Toussaint, The symmetric all-furthest neighbor problem, *Comput. Math. Appl.* **9** (1983), 747–754.

Received November 4, 1996; revised December 22, 1997; accepted January 5, 1998