PARALLEL TRANSITIVE CLOSURE AND POINT LOCATION IN PLANAR STRUCTURES*

ROBERTO TAMASSIA† AND JEFFREY S. VITTER†

Abstract. Parallel algorithms for several graph and geometric problems are presented, including transitive closure and topological sorting in planar st-graphs, preprocessing planar subdivisions for point location queries, and construction of visibility representations and drawings of planar graphs. Most of these algorithms achieve optimal $O(\log n)$ running time using $n/\log n$ processors in the EREW PRAM model, n being the number of vertices.

Key words. parallel algorithms, parallel computation, graph algorithms, planar st-graphs, transitive closure, reachability, planar point location, computational geometry, fractional cascading, graph drawing, visibility

AMS(MOS) subject classifications. 68E05, 68C05, 68C25

1. Introduction. Planar st-graphs, which include series-parallel graphs as a special case, were first introduced by Lempel, Even, and Cederbaum [34] in connection with a planarity testing algorithm, and they have subsequently been used in a host of applications, dealing with partial orders [30], planar graph embedding [6], [14], [49], graph planarization [37], graph drawing [13], [15], floor planning [57], planar point location [19], [39], visibility [36], [42], [52], [54], [58], [59], motion planning [41], and VLSI layout compaction [57].

In this paper, we present a new technique for constructing in parallel an implicit representation of the transitive closure of a planar st-graph. This technique is further applied to obtain optimal parallel algorithms for the following problems:

(1) transitive closure, reachability, and topological sorting in planar st-graphs;

(2) preprocessing planar subdivisions for point location queries;

(3) construction of visibility representations and drawings of planar graphs.

We adopt the standard *parallel random-access machine* (PRAM) model of computation, in which processors concurrently access a shared memory [29]. Communication costs are not taken into account by this model; the time to access a memory location is constant for each processor. An exclusive-read exclusive-write (EREW) PRAM prohibits concurrent access to the same location of the shared memory. A concurrent-read exclusive-write (CREW) PRAM allows concurrency for reads but not for writes. A concurrent-read concurrent-write (CRCW) PRAM allows concurrent reading and concurrent writing, under various conventions for concurrent writing. Our algorithms use the most restrictive EREW PRAM.

Computing the transitive closure of a digraph G with n vertices can be done sequentially in linear time, but the best known parallel algorithms require $O(\log^2 n)$

* Received by the editors November 14, 1989; accepted for publication (in revised form) July 17, 1990. An extended abstract of this paper was presented at the 1989 ACM Symposium on Parallel Algorithms and Architectures, Santa Fe, New Mexico, June 1989, and at the International Workshop on Discrete Algorithms and Complexity, Fukuoka, Japan, November 1989.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912–1910. Research was supported in part by grant CCR–9007851 from the National Science Foundation, by grant DAAL03–91–G–0035 from the Army Research Office, and by the Office of Naval Research and the Defense Advanced Research Projects Agency under contract N00014–83–K–0146 and ARPA order 6320, amendment 1. The research of the first author was also supported in part by a research grant from Cadre Technologies, Inc. The research of the second author was also supported in part by a National Science Foundation Presidential Young Investigator Award CCR–8906419, with matching funds from IBM Corporation, and by National Science Foundation research grant DCR–8403613.

time on an EREW PRAM and $O(\log n)$ time on a CREW PRAM with $M(n)$ processors [29], where the best known upper bound on $M(n)$ is currently $M(n) = O(n^{2.376})$ [10]. Transitive closure is a fundamental problem, and as a result much attention is given to reducing the required number of processors. The best previous results on the related problems of deciding the reachability of a vertex v from a vertex u (transitive closure query) and of computing a topological ordering of the vertices of an acyclic digraph G have the same time/processor bounds as transitive closure.

In the next section we discuss some important properties of planar st-graphs. In particular, we recall that a planar st-graph G admits two total orders on the set $V \cup E \cup F$, where V, E, and F are the sets of vertices, edges, and faces of G, respectively. Such total orders, denoted $<_{\mathrm{L}}$ and $<_{\mathrm{R}}$, provide an implicit representation of the transitive closure of G. Also, any such order yields a topological ordering of the vertices when restricted to V [51].

In § 3 we give an optimal $O(\log n)$-time, $(n/\log n)$-processor algorithm for constructing the orders $<_{\mathrm{L}}$ and $<_{\mathrm{R}}$ of an n-vertex planar st-graph G. This algorithm can be used as a preprocessing step to set up an $O(n)$-space data structure that supports transitive closure queries in $O(1)$ sequential time. Alternatively, we can construct within the same bounds a fully dynamic data structure that supports queries and updates (insertions/deletions of vertices and edges) in $O(\log n)$ sequential time. Using a different data structure, updates take $O(1)$ time with n processors and queries take $O(1)$ time with one processor. Since the publication of the conference version of this paper, Kao and Klein [28] have developed a transitive closure algorithm for general planar graphs that runs in $O(\log^3 n)$ time using n processors on a CRCW PRAM.

Section 4 considers the classical problem of point location in a planar subdivision, a fundamental searching primitive for a variety of geometric algorithms. We show how to preprocess a monotone subdivision in $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM to obtain an $O(n)$-space data structure (the bridged separator tree [19], [33]) that supports point location queries in $O(\log n)$ time. Our technique can also be extended to construct a fully dynamic data structure for point location. Queries in the bridged separator tree can be done in optimal $O\big((\log n)/\log p\big)$ time using a p-processor CREW PRAM [56]. Nonmonotone subdivisions can be handled by our techniques by first applying a triangulation step, which takes $O(\log n)$ time using an n-processor CREW PRAM [3], [60].

Our results improve certain aspects of the previous best results [3], [9], [11], [12]. Atallah, Cole, and Goodrich [3] give an algorithm to construct a suboptimal $O(n \log n)$-space point location data structure in $O(\log n)$ time with n processors on a CREW PRAM. Dadoun and Kirkpatrick [11] show that the $O(n)$-space hierarchical point location data structure of Kirkpatrick [31] for triangulations can be constructed in $O(\log n \log^* n)$ worst-case time and $O(\log n)$ expected time on a CREW PRAM with n processors. A recent result of Cole and Zajicek [9] shows that the worst-case time can be reduced to $O(\log n)$ with $n/\log n$ processors at the expense of large constant factors. The hierarchical data structure can be modified so that it can process point location queries in $O\big((\log n)/\log p\big)$ time, but the required preprocessing takes $O(\log^2 n)$ time using $O(n^3)$ processors on a CREW PRAM [12]. An empirical analysis of the performance of several point-location data structures shows that the hierarchical point location data structure does not perform well in practice since the constant factors hidden behind the big-oh notation are large, whereas the bridged separator-tree constructed by our algorithm is very efficient [18].

In § 5, we investigate the problem of constructing visibility representations of

planar graphs, where the vertices are represented by horizontal segments and the edges by vertical segments. Such representations find applications in VLSI layout, motion planning, and graph drawing, and their combinatorial properties have been extensively investigated [16], [42], [52], [54], [58], [59]. We give algorithms for constructing visibility representations of planar st-graphs and undirected planar graphs in $O(\log n)$ time with $n/\log n$ processors. Also, we show that algorithms for drawing planar graphs that are based on the intermediate construction of visibility representations can be efficiently parallelized. We present algorithms that construct planar drawings with vertices placed at integer coordinates and asymptotically optimal area in $O(\log n)$ time with $n/\log n$ processors. This improves substantially over the algorithm of Ja'Ja' and Simon [26], which uses $M(n)$ processors to construct in $O(\log^2 n)$ time a planar drawing with vertices placed at real coordinates and no known bound on the area.

As a final remark, our parallel algorithms appear to be simple to implement and eminently practical.

2. Planar st-graphs.

DEFINITION 2.1. A *planar st-graph* G is a planar acyclic directed graph G with exactly one source vertex s and exactly one sink vertex t, which is embedded in the plane such that s and t are on the boundary of the external face.

An example is pictured in Fig. 1. We assume in this paper, as stated in Definition 2.1, that the input graph representation is embedded, that is, for each vertex the cyclical ordering of its neighbors is given. The embedding is represented in standard form by doubly-connected edge lists [38]. If the embedding information is not available, but a planar straight-line drawing is given, the embedding can be determined on an EREW PRAM in $O(\log d)$ time with n processors by sorting, where d is the maximum vertex degree [7]. This is optimal in the worst case, since sorting can be reduced to computing the embedding. If neither the embedding nor a drawing is given, the embedding can be determined as follows: We first add the directed edge (s,t) to G if it does not already exist. Let $\widehat{G}$ be the undirected planar graph corresponding to G. We can compute an embedding of $\widehat{G}$ on a CRCW PRAM in $O(\log n)$ time using the same number of processors needed to determine graph connectivity and to do bucket sorting in $O(\log n)$ time [40]; the best known processor bound for this uses $n \log\log n$ processors deterministically [8], [24]. The resulting embedding is consistent with having any particular edge of $\widehat{G}$ appear on the external face, so we can assume that the edge (s,t), and thus vertices s and t, are on the external face. If the edge (s,t) was added in our construction earlier, it can be removed from $\widehat{G}$, and the orientations of the edges can be reintroduced to get an embedding of the planar st-graph G.

Following the development of Tamassia and Preparata [51], we will consider a planar embedding of G with s as the lowest vertex and t as the highest vertex, and with all edges directed upwards. Planar st-graphs have the following important properties [34], [52]:

(1) Every vertex is on a directed path from s to t.

(2) The incoming edges for each vertex appear consecutively around the vertex, and so do the outgoing edges. The face separating the incoming and outgoing edges of vertex v in the clockwise direction is called $left(v)$, and the face separating them in the counterclockwise direction is called $right(v)$. (See Fig. 2(a).)

(3) The boundary of each face f consists of two directed paths enclosing f, each starting from the unique lowest vertex $low(f)$ and ending at the unique high-

FIG. 1. *A planar st-graph G (solid lines) and its dual graph G^* (dashed lines).*

est vertex $high(f)$. (See Fig. 2(b).)

The terminology can be extended by defining vertices $low(x)$ and $high(x)$ and faces $left(x)$ and $right(x)$ for all elements in $V \cup E \cup F$, where V is the set of vertices, E is the set of edges, and F is the set of faces of G. For each vertex v, we define $low(v) = high(v) = v$ and $left(v)$ and $right(v)$ as above. For each edge $e = (u, v)$, we define $low(e) = u$, $high(e) = v$, and we define $left(e)$ to be the face to the left of e and $right(e)$ to be the face to the right of e. For each face f, we define $low(f)$ and $high(f)$ as above and $left(f) = right(f) = f$.

DEFINITION 2.2. The dual graph G^* of a planar st-graph G is the directed graph formed as follows: For each face of G, there is a vertex of G^*. In addition, the external face of G corresponds to two vertices s^* and t^* of G^*, which represent the "left" and "right" external faces of G. For each edge e in G, there is an edge $(left(e), right(e))$ in G^*. (See Fig. 1.)

It is easy to show that the dual graph G^* is also a planar st-graph. Partial orders $\uparrow$ and $\rightarrow$ can be defined on $V \cup E \cup F$ as follows:

DEFINITION 2.3. We say x is *below* y (denoted $x \uparrow y$) if there is a path from $high(x)$ to $low(y)$ in G, and we say x is *to the left of* y (denoted $x \rightarrow y$) if there is a path from $right(x)$ to $left(y)$ in the dual graph G^*.

For example, in Fig. 1, we have $e_2 \uparrow f_3 \uparrow t$ and $e_1 \rightarrow f_3 \rightarrow v_2$. For each $x, y \in V \cup E \cup F$, exactly one of the following relations hold: $x \uparrow y$, $y \uparrow x$, $x \rightarrow y$, or $y \rightarrow x$ [51]. This allows us to define the following two total orders:

DEFINITION 2.4. The total orders $<_{\text{L}}$ and $<_{\text{R}}$ are defined as

$$x <_{\text{L}} y \iff x \uparrow y \text{ or } x \rightarrow y;$$
$$x <_{\text{R}} y \iff x \uparrow y \text{ or } y \rightarrow x.$$

We define the *left sequence of* G to be the sequence of elements of $V \cup E \cup F$ sorted with respect to $<_{\text{L}}$, and the *right sequence of* G to be the sequence of elements of $V \cup E \cup F$ sorted with respect to $<_{\text{R}}$.

FIG. 2. *Parallel construction of the left sequence.* (a) *The order relations* $e_1 <_L v <_L e_2$ *formed by Rule* 1. (b) *The order relations* $e_1 <_L f <_L e_2$ *formed by Rule* 2. (c) *The left sequence in list form shown for the graph* G *in Fig.* 1.

For example, the left and right sequences for the graph in Fig. 1 are respectively

$$f_0\, v_0\, e_1\, f_1\, e_2\, v_1\, e_5\, v_3\, e_6\, v_5\, e_9\, f_2\, e_7\, f_3 e_4\, f_4\, e_3\, v_2\, e_8\, v_4\, e_{10}\, v_6\, f_5,$$

and

$$f_5\, v_0\, e_3\, f_4\, e_2\, v_1\, e_4\, v_2\, e_8\, f_3\, e_5\, v_3\, e_7\, v_4 e_{10}\, f_2\, e_6\, f_1\, e_1\, v_5\, e_9\, v_6\, f_0.$$

This left sequence is also pictured as a path in Fig. 2(c). The formal underpinning of the orders $<_{\text{L}}$ and $<_{\text{R}}$ can be found in the theory of planar lattices [27], [30].

The importance of the total orders $<_{\text{L}}$ and $<_{\text{R}}$ is that they can be used to answer transitive closure queries:

THEOREM 2.5. [51] *There is a path from vertex u to vertex v in a planar st-graph G if and only if u precedes v in both the left and right sequences of G.*

3. Transitive closure. The *transitive closure query problem* for a digraph G consists of answering queries of the form "Is there a path from vertex u to vertex v in G?". In the dynamic problem, the digraph can be updated by insertions and deletions, and the queries can be interspersed with the updates. In this section, we exploit the properties of planar st-graphs and give EREW PRAM algorithms for constructing the fully dynamic (sequential) data structure of Tamassia and Preparata [51] in $O(\log n)$ time with $n/\log n$ processors. The data structure consists of a pair of balanced trees associated with the left and right sequences and requires $O(n)$ space. When used sequentially, it is fully dynamic and handles queries and updates in $O(\log n)$ time. We also give parallel algorithms for dynamic queries and updates.

THEOREM 3.1. *Let G be a planar st-graph with n vertices. A fully dynamic data structure for the transitive closure query problem for G can be constructed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

Proof. Our algorithm constructs the data structure of Tamassia and Preparata [51] based on the left and right sequences of G. By Theorem 2.5, we can determine if there is a path from u to v in G by checking whether u is before v in both sequences. Each sequence is stored in the leaves of a balanced red-black tree [22]. Dynamic updates require a sequence of splits and splices in the tree.

Without loss of generality, let us restrict our attention to computing the left sequence of G. First we construct the dual graph G^*. The edges on the right boundary (respectively, left boundary) of each face f can identify a common representative vertex, say vertex $low(f)$, in parallel simultaneously for each face f, as follows: We construct a local order relation among the edges. If an edge is the leftmost (rightmost) edge incoming into a vertex, its successor is defined as the leftmost (rightmost) edge outgoing from that vertex. This order relation induces a set of ordered paths, corresponding to the right boundaries (left boundaries) of the faces. By list ranking [2], [8], the edges in the right boundary (left boundary) of each face can simultaneously identify a common vertex in $O(\log n)$ time with $O(n/\log n)$ processors.

To construct the left sequence of G, we note that, except for the very beginning and very end of the sequence, every other element in the sequence is an edge. We can form the sequence in $O(\log n)$ time with $O(n/\log n)$ processors by creating the following local order relations:

(1) Each vertex $v \neq s, t$ constructs the order relations $e_1 <_{\text{L}} v <_{\text{L}} e_2$, where e_1 is the rightmost incoming edge of v, and e_2 is the leftmost outgoing edge of v. (See Fig. 2(a).)

(2) Each interior face f constructs the two order relations $e_1 <_{\text{L}} f <_{\text{L}} e_2$, where e_1 is the topmost left edge of f, and e_2 is the bottommost right edge of f. (See Fig. 2(b).)

The source vertex s constructs the order relations $s^* <_{\mathrm{L}} s <_{\mathrm{L}} e_2$, where e_2 is the leftmost outgoing edge of s, and t forms the order relations $e_1 <_{\mathrm{L}} t <_{\mathrm{L}} t^*$, where e_1 is the rightmost incoming edge of t. List ranking is then done to combine the order relations into a fully ordered sequence, as shown in Fig. 2(c). Lemma 3.2 below shows that this sequence is the left sequence of G. The right sequence can be constructed analogously.

Given the left and right sequences of G, the dynamic data structure of Tamassia and Preparata [51] can be constructed easily in parallel. It consists of two balanced search trees, whose leaves consist of the elements of $V \cup E \cup F$. In one tree the leaves are ordered from left to right according to the left sequence, and in the other tree the leaves are ordered according to the right sequence. □

LEMMA 3.2. *List ranking of the above local order relations produces the left sequence of G.*

Proof. The local order relations produced above do not induce any cycles, since each order relation is consistent with the total order $<_{\mathrm{L}}$. The rest of the proof consists of showing by contradiction that the order relations induce a linear order on $V \cup E \cup F$. Suppose, during the application of the above two rules, that some edge e is chosen twice as the head of two different subsequences. One of the subsequences must be formed by rule 1 above, and the other subsequence by rule 2, since two different vertices cannot have the same outgoing left edge, and two different faces cannot have the same bottommost right edge. Let us denote these two subsequences by $e' \, v \, e$ and $e'' \, f \, e$, for some vertex $v \neq s$ and some interior face f. By rule 1, e is the leftmost outgoing edge of v. By rule 2, e is the bottommost right edge of an interior face f, which implies that $low(f) = v$. This means that e is to the right of f, but there are no edges to the left of f, and hence f is not an interior face — a contradiction. We can show in a similar way that an edge cannot be chosen as the tail of two different subsequences. □

The fact that the total order $<_{\mathrm{L}}$ is an extension of the partial order $\uparrow$ imposed by the directed edges of the graph gives us the following corollary:

COROLLARY 3.3. *A topological ordering of the n vertices of a planar st-graph G can be computed in $O(\log n)$ time using $n/\log n$ processors on an* EREW PRAM, *which is optimal. Specifically, we can compute the rank of each vertex in the vertex subsequence of the left or right sequence of G.*

Proof. First we compute the left-sequence (or right-sequence) of G, and then we extract the subsequence consisting of all the vertices by list ranking. □

Series-parallel graphs are a subclass of planar st-graphs, and thus we get the following corollary, which is an improvement over the $O(\log^2 n)$-time, n-processor CREW PRAM algorithm given by Afrati, Goldin, and Kanellakis [1]:

COROLLARY 3.4. *Reachability in series-parallel graphs can be computed on an* EREW PRAM *in $O(\log n)$ time with $n/\log n$ processors.*

Our technique can be extended to solve the following problem posed by Kao: Given a planar st-graph G, compute for each vertex v the number of vertices reachable from v by paths in G. By associating each vertex v with a point $p(v)$ in the plane whose x- and y-coordinates are given by the ranks of v in the left and right sequences, respectively, we find that a vertex w is reachable from v if and only if the x- and y-coordinates of $p(w)$ are both greater than the corresponding ones of $p(v)$. Hence, we can apply the algorithm of Atallah, Cole, and Goodrich [3] for two-set dominance counting and obtain the following theorem.

THEOREM 3.5. *Given a planar st-graph G with n vertices, the number of vertices*

reachable from each vertex can be computed by an EREW PRAM *in* $O(\log n)$ *time using* n *processors.*

The *contact chain query* problem for a convex subdivision and a direction θ consists of questions of the form: "If region r' is pushed in direction θ, will region r'' be moved?" [5]. Without loss of generality, assume that θ is the horizontal direction. By orienting the edges of the convex subdivision from bottom to top, and denoting by s and t the lowest and highest vertices, respectively, we get a planar st-graph G. (The subdivision is perturbed slightly if necessary to ensure that there are no horizontal edges.) It is easy to see that pushing r' will cause r'' to be moved if and only if there is a path from r' to r'' in the dual graph G^*. By Theorem 3.1 for the dual graph G^*, we get the following corollary.

COROLLARY 3.6. *A fully dynamic* $O(n)$*-space* $O(\log n)$*-sequential time data structure for the contact chain query problem along a fixed direction* θ *in an* n*-vertex convex subdivision can be constructed by an* EREW PRAM *in* $O(\log n)$ *time using* $n/\log n$ *processors, which is optimal.*

An alternate simple data structure for parallel use stores the left and right sequences as linear arrays. In array form, the shifts and swaps needed for dynamic maintenance can clearly be done in constant time using n processors. If n^ϵ processors are available for updates, where $0 < \epsilon < 1$, the arrays can be replaced by B-trees [4] with nodes of degree $\Theta(n^\epsilon)$ and hence $O(1/\epsilon)$ height. This gives us the following result.

THEOREM 3.7. *Let* G *be a planar* st*-graph with* n *vertices. For any constant* $0 < \epsilon \leq 1$*, a data structure for the transitive closure query problem for* G *can be constructed by an* EREW PRAM *in* $O(\log n)$ *time using* $n/\log n$ *processors, such that a transitive closure query can be answered on an* EREW PRAM *in* $O(1/\epsilon)$ *time with one processor, and dynamic updates can be done in* $O(1/\epsilon)$ *time with* n^ϵ *processors.*

COROLLARY 3.8. *Let* G *be a planar* st*-graph with* n *vertices. After the preprocessing of Theorem* 3.7*, the subgraph* H *consisting of all paths from a vertex* u *to a vertex* v *can be generated in* $O(\log n)$ *time using* $n/\log n$ *processors on an* EREW PRAM*, and in constant time with* n *processors on a* CREW PRAM*.*

Proof. We assign one processor to each vertex and edge in the graph and broadcast the positions of u and v in $<_{\text{L}}$ and $<_{\text{R}}$ to all the processors. We form the desired subgraph H by including all the vertices and edges such that there is a path from u to v using that vertex or edge. This can be done using Theorem 2.5, by including all vertices and edges that come between u and v with respect to both $<_{\text{L}}$ and $<_{\text{R}}$. □

The shorter of the leftmost and rightmost paths from u to v can be generated in $O(\log n)$ time on an EREW PRAM, and in $O(\log k)$ time on a CREW PRAM, where k is the length of the path. We form the dual graph H^* of H. For each edge e in H, we test to see if it is on the leftmost (respectively, rightmost) path in H by checking if *left*(e) (respectively, *right*(e)) is not between u and v in either $<_{\text{L}}$ or $<_{\text{R}}$. This identifies the edges along the two paths. The shorter of the two paths can then be found by doing list ranking in parallel for each path.

4. Planar point location. In this section, we present fast parallel algorithms for constructing data structures to handle point location queries. The queries themselves can be done either serially or in parallel using concurrent read. The reader is referred to the book of Preparata and Shamos [38] for the geometric terminology used in this section and a description of various point location techniques. Our approach is based on the *separator-method* for point location [19], [33].

DEFINITION 4.1. A *monotone chain* is a polygonal chain such that each horizontal line intersects it in at most one point. A polygon is *monotone* if its boundary is partitionable into two monotone chains. A (*planar*) *subdivision* S is a partition of the entire plane into polygons, called the *regions* of S. We assume a standard representation for the subdivision S and its embedding, such as a doubly-connected edge list representation [38]. A *monotone subdivision* is such that all its regions are monotone polygons.

A monotone subdivision S is therefore associated with a planar st-graph G, where each edge is directed according to increasing ordinate, and s and t are associated with the vertices at $-\infty$ and $+\infty$ of S. That is, an upward (respectively, downward) ray of S originating at vertex v corresponds to edge (v, t) (respectively, (s, v)) of G.

DEFINITION 4.2. Given a monotone subdivision S, a *separator* σ of S is a monotone chain of S between vertices at infinity, that is, a directed path of G from s to t. Given separators σ_1 and σ_2, we say that σ_1 is *to the left of* σ_2 if every horizontal line intersects σ_1 at or to the left of σ_2.

Let $r_1, r_2, \ldots, r_p$ be the regions of S, sorted according to some total order compatible with relation $\rightarrow$, that is, $r_i \rightarrow r_j$ implies $i < j$. The common boundary of the regions with index $\leq i$ and of the regions with index $> i$ is a separator of S, which we denote σ_i. Clearly, σ_i is to the left of σ_j, for $i < j$.

One approach to point location is to perform a type of binary search on the set of separators $\Sigma = \{\sigma_1, \cdots, \sigma_{p-1}\}$, where each separator σ_i is assigned to a node (called *node* σ_i) of a balanced binary tree T (called the *separator tree*), whose leaves are the regions of S [33]. The sequence of the nodes of T in symmetric order is $r_1, \sigma_1, r_2, \sigma_2, \ldots, \sigma_{p-1}, r_p$. An edge (u, v) of S belongs to the interval of separators $\sigma_i, \sigma_{i+1}, \ldots, \sigma_k$ such that $r_i = \mathit{left}(u, v)$ and $r_{k+1} = \mathit{right}(u, v)$; but for reasons of space efficiency (u, v) is stored only once, at node $\sigma_j = \mathit{lca}(r_i, r_{k+1})$, the lowest common ancestor of leaves r_i and r_{k+1}. The edges stored at a node σ_i, which are a subset of the edges of separator σ_i, are called the *proper edges* of σ_i. An example is shown in Fig. 3.

The separator tree uses $O(n)$ space and supports point location queries in $O(\log^2 n)$ time, where n is the number of vertices of S [33]. To perform a query, we trace a path in the separator tree from the root to the leaf r_i containing the query point q. At each internal node σ_i we discriminate q against separator σ_i and branch left or right according to whether q is to the left or right of σ_i. The discrimination of q against σ_i is performed by searching for the smallest value $\geq y(q)$ in the catalog of σ_i. The catalog consists of the y-coordinates of the proper edges of σ_i, along with the dummy value $+\infty$. Each catalog entry is associated with the proper edge e (if it exists) whose top vertex has that y-coordinate. If the search for $y(q)$ returns the y-coordinate associated with edge e, then e is horizontally visible from q; we branch left if q is to the left of e, and right otherwise. When there is no edge associated with the y-coordinate returned, then $y(q)$ is in a "gap" between two proper edges of σ_i. In this case, the branching direction is determined as follows: If $y(q)$ is immediately above (respectively, below) proper edge e of σ_i, let σ_k be the ancestor of σ_i in the separator tree that stores the first nonproper edge of σ_i above (respectively, below) e. We branch left if σ_i is to the left of σ_k, and right otherwise. This information can be precomputed and stored in the catalog. Thus, the necessary branching can always be determined in constant time from the information associated with the y-coordinate returned as a result of the search in the catalog. Point location in this context consists merely of a sequence of catalog searches.

FIG. 3. *Construction of the separator tree for a regular subdivision:* (a) *Regular subdivision S with the chains of proper edges visualized.* (b) *Separator tree for S.*

By applying the fractional cascading technique to the catalogs of the separator tree, we obtain a *bridged separator tree* (also called *layered dag*), which still uses $O(n)$ space and supports queries in $O(\log n)$ time, which is optimal [19]. (Our method in the previous paragraph for determining the branching in "gaps" yields a slight simplification of the algorithm.)

DEFINITION 4.3. A *regular* subdivision is a monotone subdivision having no pair of regions r and r' such that $r \uparrow r'$. (See Fig. 3.)

It follows that in a regular subdivision the relation $\rightarrow$ is a total order. Below, we show how to efficiently construct in parallel the bridged separator tree for a regular monotone subdivision, and then we extend the technique to arbitrary monotone subdivisions and general nonmonotone subdivisions.

LEMMA 4.4. *Each vertex of a regular subdivision has either indegree* 1 *or outdegree* 1.

Proof. If some vertex v of a regular subdivision has $indeg(v) \geq 2$ and $outdeg(v) \geq 2$, then there are regions r and r' such that $v = high(r) = low(r')$, which implies $r \uparrow r'$, a contradiction. □

The following algorithm constructs a bridged separator tree for a regular subdivision S. Without loss of generality, we assume that the number of regions p is a power of two.

(1) Construct the planar st-graph G associated with S, and compute its left and right sequences. Also, compute $indeg(v)$ and $outdeg(v)$ for each vertex v, and store with each edge (u, v) the indices i and j of the regions $r_i = left(u, v)$ and $r_j = right(u, v)$.

(2) Form a complete binary tree T whose leaves are associated with the regions of S (the faces of G), sorted from left to right according to their order in the left sequence of G. Hence, region r_i is the ith leaf from left to right. Also, construct an array of pointers to the internal nodes of T such that the ith element of the array points to the internal node of T associated with separator σ_i.

(3) Form the sets of proper edges of the internal nodes of T, as follows:

```
foreach edge (v, w) do begin
    if indeg(v) = 1
        then begin
            let (u, v) be v's only incoming edge;
            if lca(left(u, v), right(u, v)) = lca(left(v, w), right(v, w))
                then connect (u, v) to (v, w) bidirectionally
        end;
    if outdeg(w) = 1
        then begin
            let (w, z) be w's only outgoing edge;
            if lca(left(w, z), right(w, z)) = lca(left(v, w), right(v, w))
                then connect (w, z) to (v, w) bidirectionally
        end
    end;
```

(4) Store each doubly-connected list of edges obtained in Step 3 into the node of T that is the lowest common ancestor of the regions to the left and right of all the edges in the list. Each list is the set of proper edges of that node, sorted from bottom to top.

(5) Convert the lists of proper edges into arrays, called *catalogs*, by means of list

ranking. Establish bridges between the catalogs stored in adjacent nodes of T, according to the fractional cascading scheme of Atallah, Cole, and Goodrich [3].

The correctness of the algorithm follows from Lemma 4.4. Step 1 is performed using the techniques developed in the previous section. Step 2 can be easily done in $O(\log n)$ time with $n/\log n$ processors. In Step 3, we use a simple technique for computing in $O(1)$ time the inorder rank of the lowest common ancestor of two leaves of a complete binary tree, given the ranks of such leaves in their left-to-right order [19]. Hence, the test

$$lca(left(u,v), right(u,v)) = lca(left(v,w), right(v,w))$$

can be done in $O(1)$ time using only the indices of the regions to the left and right of (u,v) and (v,w). By Step 1, such indices are stored locally at the edges (u,v) and (v,w). Since the iterations of the **for**-loop are independent, we conclude that we can allocate one processor per group of $\log n$ edges and perform the computation of Step 3 in $O(\log n)$ time with $n/\log n$ processors. In Step 4, the assignment of the lists of proper edges to the corresponding internal nodes of T is done as follows. First, we pick any edge (u,v) of the list and compute in $O(1)$ time the rank of node $lca(left(u,v), right(u,v))$ in the symmetric order [19]. Next, from the rank, we access the node using the array constructed in Step 2. Such computation can be performed in $O(\log n)$ time with $n/\log n$ processors.

The parallel fractional cascading technique of Atallah, Cole, and Goodrich [3] takes $O(\log n)$ time with $n/\log n$ processors to complete the construction of the bridged separator tree. This technique can be applied because, as described earlier, point location consists precisely of a series of catalog searches, where each node σ_i in the separator tree contains a catalog of y-coordinate values. One property of a regular subdivision is that the proper edges of each separator in the separator tree are connected, so that there are no "gaps" in the middle of a separator, but only at the top and bottom [39]. Thus, all but the first and last catalog entries are associated with a proper edge e of σ_i, and this simplifies the algorithm. This proves the following.

LEMMA 4.5. *Let S be a regular subdivision with n vertices. The bridged separator tree for point location in S can be constructed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

DEFINITION 4.6. We call two regions r' and r'' *vertically consecutive* if $r' \uparrow r''$ and there is no region r with $r' <_{\mathrm{L}} r <_{\mathrm{L}} r''$. It can be shown that there is a unique monotone chain from $high(r')$ to $low(r'')$, called a *channel*, and that all channels are vertex disjoint [39].

If the subdivision S is monotone, but not regular, we transform S into an equivalent regular subdivision by duplicating some edges [39]. Given two vertically consecutive regions r and r', we can imagine duplicating the channel from r to r', viewing the measure-zero region delimited by the two replicas as a degenerate polygon joining r and r' and merging them into a new region $r \cup r'$. By merging all sequences of vertically consecutive pairs in this way we obtain a regular subdivision S^* whose regions are *clusters* of regions of S. (See Fig. 4.)

The algorithm for constructing subdivision S^* is as follows:

(1) Construct the planar st-graph G associated with S, and compute its left and right sequences.

(2) Extract the subsequence $r_1, r_2, \cdots, r_p$ of regions from the left sequence and determine the vertically consecutive pairs.

FIG. 4. (a) *A monotone subdivision* S *and* (b) *the corresponding regular subdivision* S^*. *Notice the clusters of regions* $r_4 \cup r_5$ *and* $r_8 \cup r_9$.

(3) For each vertically consecutive pair (r_i, r_{i+1}), mark the vertices and edges that are between $high(r_i)$ and $low(r_{i+1})$ in the left sequence.
(4) Duplicate all the vertices and edges that are marked and update the subdivision accordingly.

The transitive closure algorithm referred to in Theorem 3.1 is used for the preprocessing in Step 1. The subsequence of regions can be formed using a standard binary tree communication scheme. We can verify whether two regions r_i and r_{i+1} are vertically consecutive by comparing the y-coordinates of vertices $high(r_i)$ and $low(r_{i+1})$. The remaining computations in the algorithm can be done easily in parallel. This proves the following.

LEMMA 4.7. *The regular subdivision S^* associated with a monotone subdivision S with n vertices can be computed by an* EREW PRAM *in $O(\log n)$ time with $n/\log n$ processors.*

The complete algorithm for preprocessing a monotone subdivision S consists of constructing S^* from S, and then building the bridged separator tree T^* for S^*. In practice, Step 1 for constructing the bridged separator tree can be bypassed, since the ordered list of regions in S^*, sorted according to the left sequence, can be obtained directly from the corresponding list in S by contracting regions that are merged together into a cluster. The indegrees and outdegrees can be obtained directly also. Each leaf χ of T^* corresponds to a region of S^*, which in turn consists of some cluster of regions $r'_1, r'_2, \ldots, r'_k$ of S. We add to each leaf χ of T^* a pointer to a balanced search tree that stores the regions $r'_1, r'_2, \ldots, r'_k$, sorted from bottom to top.

To perform point location in S, we first determine the cluster χ containing the query point q by searching in T^*. Next, we search in the balanced tree pointed to by leaf χ in order to determine which region r_i of χ contains q. Hence, by combining the results of Lemmas 4.5 and 4.7, we obtain the following theorem.

THEOREM 4.8. *Let S be a monotone subdivision with n vertices. An $O(n)$-space data structure supporting $O(\log n)$-time point location queries in S can be constructed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

The algorithm used in Theorem 4.8 can be modified to construct the fully dynamic point location data structure of Preparata and Tamassia [39] within the same time/processor bounds.

For subdivisions that are represented without embedding information (e.g., by unsorted lists of vertices and edges), we need a preliminary step to compute its embedding, which consists of sorting the neighbors of each vertex v in clockwise order around v. This can be done in $O(\log n)$ time using n processors [7]. Note that if the embedding of S is not given as part of the input, there is an $\Omega(n \log n)$ lower bound on the amount of work needed to compute the embedding in the worst case [32].

For nonmonotone subdivisions we perform a preliminary triangulation step and then apply the technique for monotone subdivisions. Triangulation can be performed by a CREW PRAM in $O(\log n)$ time with n processors [3], [60].

We get the following theorem.

THEOREM 4.9. *Let S be a subdivision with n vertices. An $O(n)$-space data structure supporting $O(\log n)$-time point location queries in S can be constructed by a* CREW PRAM *in $O(\log n)$ time using n processors.*

The bridged separator tree data structure can also be used to process the queries in parallel. We show in a companion paper [56] that an $O(n)$-space data structure can be constructed with an EREW PRAM in $O(\log n)$ time using $n/\log n$ processors such that, for any $2 \leq p \leq n$, point location queries can be done in $O\big((\log n)/\log p\big)$

time using a CREW PRAM with p processors. This algorithm improves upon the one of Dadoun and Kirkpatrick [12]. It achieves the same query time, but it is simpler and uses less preprocessing. The query time of $O\big((\log n)/\log p\big)$ is optimal since we can reduce the problem of dictionary searching to planar point location, and thus the lower bound of Snir [46] applies.

5. Visibility representations and graph drawing. The concept of *visibility* plays a fundamental role in a variety of geometric problems and applications, such as art gallery problems [35], VLSI layout [25], [44], [57], motion planning [23], [41], and graph drawing [13], [53].

DEFINITION 5.1. Given a collection H of horizontal segments in the plane, the (vertical) *visibility graph* of H is the graph G whose vertices are the segments of H and whose edges are pairs of segments that see each other in the vertical direction. The edges of G can be oriented from bottom to top to yield an acyclic digraph.

DEFINITION 5.2. A *visibility representation* , for a directed graph G maps each vertex v of G to a horizontal segment , (v) and each edge (u, v) to a vertical segment , (u, v) that has its lower endpoint on , (u), its upper endpoint on , (v), and does not intersect any other horizontal segment. (See Fig. 5(a).) If G is an undirected planar graph, a visibility representation for G is defined as a visibility representation for some orientation of G.

Besides having many applications, visibility graphs and representations are also of intrinsic theoretical interest, and their combinatorial properties have been extensively investigated [16], [52], [54], [58], [59].

The visibility graph of a set of n segments can be computed in $O(n \log n)$ sequential time and $O(n)$ space [44], which is optimal. It can also be constructed in parallel by an EREW PRAM in $O(\log n)$ time and $O(n \log n)$ space with n processors [3], or in $O(\log^2 n)$ time and $O(n)$ space with $n/\log n$ processors [43]. As regards visibility representations, there are sequential $O(n)$-time algorithms for their construction [13], [42], [52].

THEOREM 5.3. *Let G be a planar st-graph with n vertices. A visibility representation for G with integer coordinates and $O(n^2)$ area can be computed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

Proof. A visibility representation for G can be constructed by the following variation of previous algorithms [13], [42], [52].

(1) Compute a topological ordering $Y(v)$ of the vertices of G.
(2) Compute a topological ordering $X(f)$ of the vertices of G^*, the dual graph of G.
(3) Draw each vertex-segment , (v) at ordinate $Y(v)$ and between abscissae $X(\mathit{left}(v))$ and $X(\mathit{right}(v)) - 1$.
(4) Draw each edge-segment , (e) at abscissa $X(\mathit{left}(e))$ and between ordinates $Y(\mathit{low}(e))$ and $Y(\mathit{high}(e))$.

By Corollary 3.3, Steps 1 and 2 take $O(\log n)$ time using $n/\log n$ processors. The parallel computation of Steps 3 and 4 within the same bounds is straightforward. □

Given a 2-connected embedded undirected planar graph G, we choose s and t to be two adjacent vertices (which we can assume to be on the external face) and orient the edges of G so that the resulting digraph is a planar st-graph, and then we apply the previous theorem. Such an orientation of G can be computed by an EREW PRAM in $O(\log n)$ time with $n/\log n$ processors using the st-numbering algorithm of Gazit [21].

THEOREM 5.4. *Let G be a 2-connected embedded (undirected) planar graph with*

FIG. 5. (a) *Visibility representation for a planar st-graph G.* (b) *A planar upward polyline grid drawing of G.* (c) *A planar orthogonal grid drawing of an undirected graph.*

n vertices. A visibility representation for G with integer coordinates and $O(n^2)$ area can be computed by an EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors.*

A number of data presentation problems involve drawing graphs so that they are easy to read and understand. Examples include circuit schematics, algorithm animation, and diagrams for information systems analysis and design. The literature on graph drawing algorithms is spread over the broad spectrum of computer science [17], [50]. This problem has received increasing theoretical interest in recent years (cf. [15], [20], [45]).

DEFINITION 5.5. A *drawing* of a graph maps each vertex into a point of the plane, and each edge (u, v) into a simple open curve between the points associated with the vertices u and v. A *planar drawing* has no crossing edges. A *straight-line drawing* is such that every edge is drawn as a line segment. In a *polyline drawing*, every edge is drawn as a polygonal chain. An *orthogonal drawing* is a polyline drawing whose edges are chains of horizontal and vertical segments. A *grid drawing* is a polyline drawing such that the vertices and the bends of the edges have integer coordinates. An *upward drawing* for an acyclic digraph G is such that every edge (u, v) is a curve monotonically increasing in the vertical direction. (See examples in Figs. 5(b),(c).)

An edge (u, v) of a digraph is said to be *transitive* if there exists a directed path from u to v that does not contain the edge (u, v). A digraph is said to be *reduced* if it has no transitive edges. A reduced planar st-graph G admits a planar upward straight-line drawing such that the x- and y-coordinates of a vertex v are the ranks of v in the restriction to the vertices of the left- and right-sequence of G , respectively [15]. Hence, a reduced planar st-graph can be efficiently drawn in parallel from the result of Corollary 3.3.

To draw a nonreduced planar st-graph we insert a new dummy vertex v along each transitive edge (u, w) and draw the resulting reduced planar st-graph G' considering the dummy vertices as bends. To identify transitive edges in parallel we use the following lemma, where we say that edge (u, v) is the *long edge* of face f if $u = low(f)$ and $v = high(f)$.

LEMMA 5.6. *An edge e of a planar st-graph is transitive if and only if it is the long edge of either $left(e)$ or $right(e)$.*

By Euler's formula a planar graph has at most $2n - 5$ interior faces, so that Lemma 5.6 implies that a planar st-graph has at most $2n - 5$ transitive edges.

Hence, we have the following theorem:

THEOREM 5.7. *Let G be a planar st-graph with n vertices. A planar upward polyline grid drawing for G with $2n - 5$ bends and $O(n^2)$ area can be computed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

Now, we consider planar orthogonal drawings of undirected graphs. Such drawings are typical of circuit layout, and are widely used in data presentation applications because of their regularity. Sequential algorithms for planar orthogonal drawings are given by Storer [47], Tamassia [48], and Tamassia and Tollis [53].

THEOREM 5.8. *Let G be a 2-connected embedded (undirected) planar graph with n vertices, each of degree at most four. A planar orthogonal grid drawing for G with $O(n)$ bends and $O(n^2)$ area can be computed by an* EREW PRAM *in $O(\log n)$ time using $n/\log n$ processors.*

Proof. As shown by Tamassia and Tollis [53], a planar orthogonal grid drawing can be constructed from a visibility representations by local replacements performed at each vertex. Because of its locality, this transformation can be easily parallelized. Hence, the result follows from Theorem 5.4. □

The bounds on the area and the number of bends are asymptotically optimal [47]. The bound on the number of bends can be improved to the exact worst-case optimal $2n + 4$ and the algorithm can be extended to 1-connected graphs [55].

Our results improve upon the previous parallel drawing algorithm presented by Ja'Ja' and Simon [26], which constructs a straight-line planar drawing in $O(\log^2 n)$ time with $M(n)$ processors, using real arithmetic for the computation of the coordinates of the vertices. It is not known whether this algorithm can be modified to construct grid drawings with area bounded by a polynomial in n.

Acknowledgments. We would like to thank the referees for several useful comments and suggestions.

REFERENCES

[1] F. N. Afrati, D. Q. Goldin, and P. C. Kanellakis, *Efficient parallelism for structured data: directed reachability in S-P DAGS*, Tech. Report CS-88-07, Department of Computer Science, Brown University, Providence, RI, 1988.

[2] R. J. Anderson and G. L. Miller, *Deterministic parallel list ranking*, in VLSI Algorithms and Architectures, Lecture Notes in Computer Science, Vol. 319, Springer-Verlag, Berlin, New York, 1988, pp. 81–90.

[3] M. J. Atallah, R. Cole, and M. T. Goodrich, *Cascading divide-and-conquer: a technique for designing parallel algorithms*, SIAM J. Comput., 18 (1989), pp. 499–532..

[4] R. Bayer and E. M. McCreight, *Organization and maintenance of large ordered indices*, Acta Informatica, 1 (1972), pp. 173–189.

[5] B. Chazelle, H. Edelsbrunner, and L. J. Guibas, *The complexity of cutting convex polytopes*, in Proc. 19th ACM Symposium on Theory of Computing, 1987, pp. 66–76.

[6] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa, *A linear algorithm for embedding planar graphs using PQ-trees*, J. Comput. System Sci., 30 (1985), pp. 54–76.

[7] R. Cole, *Parallel merge sort*, in Proc. 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 511–516.

[8] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree, and graph problems*, in Proc. 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 478–491.

[9] R. Cole and O. Zajicek, *An optimal parallel algorithm for building a data structure for point location*, J. Parallel Distributed Comput., to appear.

[10] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progression*, in Proc. 28th ACM Symposium on Theory of Computing, 1987, pp. 1–6.

[11] N. Dadoun and D. G. Kirkpatrick, *Parallel processing for efficient subdivision search*, in Proc. 3rd ACM Symposium on Computational Geometry, 1987, pp. 205–214.

[12] ———, *Cooperative subdivision search algorithms with applications*, in Proc. 27th Annual Allerton Conference, Monticello, IL, 1989, pp. 538–547.

[13] G. Di Battista and R. Tamassia, *Algorithms for plane representations of acyclic digraphs*, Theoret. Comput. Sci., 61 (1988), pp. 175–198.

[14] ———, *Incremental planarity testing*, in Proc. 30th IEEE Symposium on Foundations of Computer Science, 1989, pp. 436–441.

[15] G. Di Battista, R. Tamassia, and I. G. Tollis, *Area requirement and symmetry display in drawing graphs*, in Proc. 5th ACM Symposium on Computational Geometry, 1989, pp. 51–60.

[16] P. Duchet, Y. Hamidoune, M. L. Vergnas, and H. Meyniel, *Representing a planar graph by vertical lines joining different levels*, Discrete Math., 46 (1983), pp. 319–321.

[17] P. Eades and R. Tamassia, *Algorithms for automatic graph drawing: an annotated bibliography*, Tech. Report CS-89-09, Department of Computer Science, Brown University, Providence, RI, 1989.

[18] M. Edahiro, I. Kokubo, and T. Asano, *A new point-location algorithm and its practical efficiency — comparison with existing algorithms*, ACM Trans. Graphics, 3 (1984), pp. 86–109.

[19] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, *Optimal point location in a monotone subdivision*, SIAM J. Computing, 15 (1986), pp. 317–340.

[20] H. de Fraysseix, J. Pach, and R. Pollack, *Small sets supporting fary embeddings of planar graphs*, in Proc. 20th ACM Symposium on Theory of Computing, 1988, pp. 426–433.

[21] H. Gazit, *Optimal EREW parallel algorithms for connectivity, ear decomposition, and st-numbering of planar graphs*, Manuscript, Department of Computer Science, Duke University, Durham, NC, 1990.

[22] L. J. Guibas and R. Sedgewick, *A dichromatic framework for balanced trees*, in Proc. 19th IEEE Symposium on Foundations of Computer Science, 1978, pp. 8–21.

[23] L. J. Guibas and F. F. Yao, *On translating a set of rectangles*, in Advances in Computing Research, Vol. 1, F. P. Preparata, ed., JAI Press Inc., Greenwich, CT, 1983, pp. 61–77.

[24] T. Hagerup, *Towards optimal parallel bucket sorting*, Tech. Report 02/1987, Universität des Saarlandes, Saarbrücken, Germany, January 1987.

[25] M. Y. Hsueh and D. O. Pederson, *Computer-aided layout of LSI circuit building-blocks*, in Proc. IEEE Internat. Symposium on Circuits and Systems, 1979, pp. 474–477.

[26] J. Ja'Ja' and J. Simon, *Parallel algorithms in graph theory: planarity testing*, SIAM J. Comput., 11 (1982), pp. 314–328.

[27] T. Kameda, *On the vector representation of the reachability in planar directed graphs*, Inform. Process. Lett., 3 (1975), pp. 75–77.

[28] M. -Y. Kao and P. N. Klein, *Towards overcoming the transitive-closure bottleneck: efficient parallel algorithms for planar digraphs*, in Proc. 22nd ACM Symposium on Theory of Computing, 1990, pp. 181–192.

[29] R. M. Karp and V. Ramachandran, *A survey of parallel algorithms for shared memory machines*, in Handbook of Theoretical Computer Science, North-Holland, Amsterdam, 1990.

[30] D. Kelly and I. Rival, *Planar lattices*, Canad. J. Math., 27 (1975), pp. 636–665.

[31] D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. Computing, 12 (1983), pp. 28–35.

[32] ———, *Establishing order in planar subdivisions*, Discrete & Computational Geometry, 3 (1988), pp. 267–280.

[33] D. T. Lee and F. P. Preparata, *Location of a point in a planar subdivision and its applications*, SIAM J. Computing, 6 (1977), pp. 594–606.

[34] A. Lempel, S. Even, and I. Cederbaum, *An algorithm for planarity testing of graphs*, in Theory of Graphs, Internat. Symposium, Rome, Italy, 1966, pp. 215–232.

[35] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, London, 1987.

[36] R. H. J. M. Otten and J. G. van Wijk, *Graph representations in interactive layout design*, in Proc. IEEE Internat. Symposium on Circuits and Systems, 1978, pp. 914–918.

[37] T. Ozawa and H. Takahashi, *A graph-planarization algorithm and its applications to random graphs*, in Graph Theory and Algorithms, Lecture Notes in Computer Science, Vol. 108, Springer-Verlag, Berlin, New York, 1981, pp. 95–107.

[38] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, Berlin, New York, 1985.

[39] F. P. Preparata and R. Tamassia, *Fully dynamic point location in a monotone subdivision*, SIAM J. Computing, 18 (1989), pp. 811–830.

[40] V. Ramachandran and J. H. Reif, *An optimal parallel algorithm for graph planarity*, in Proc. IEEE Symposium on Foundations of Computer Science, 1989, pp. 282–287.

[41] I. Rival and J. Urrutia, *Representing orders by translating convex figures in the plane*, Order, 4 (1988), pp. 319–339.

[42] P. Rosenstiehl and R. E. Tarjan, *Rectilinear planar layouts of planar graphs and bipolar orientations*, Discrete & Comput. Geom., 1 (1986), pp. 343–353.

[43] J. E. Savage and M. G. Wloka, *Parallel constraint graph generation*, in Proc. Decennial Caltech Conference on VLSI, MIT Press, Cambridge, MA, 1989, pp. 241–259.

[44] M. Schlag, F. Luccio, P. Maestrini, D. T. Lee, and C. K. Wong, *A visibility problem in VLSI layout compaction*, in Advances in Computing Research, Vol. 2, F. P. Preparata, ed., JAI Press Inc., Greenwich, CT, 1985, pp. 259–282.

[45] W. Schnyder, *Embedding planar graphs on the grid*, in Proc. 1st ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 138–148.

[46] M. Snir, *On parallel searching*, SIAM J. Computing, 14 (1989), pp. 688–708.

[47] J. A. Storer, *On minimal node-cost planar embeddings*, Networks, 14 (1984), pp. 181–212.

[48] R. Tamassia, *On embedding a graph in the grid with the minimum number of bends*, SIAM J. Computing, 16 (1987), pp. 421–444.

[49] ———, *A dynamic data structure for planar graph embedding*, in Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 317, Springer-Verlag, Berlin, New York, 1988, pp. 576–590.

[50] R. Tamassia, G. D. Battista, and C. Batini, *Automatic graph drawing and readability of diagrams*, IEEE Trans. Systems Man Cybernet., 18 (1988), pp. 61–79.

[51] R. Tamassia and F. P. Preparata, *Dynamic maintenance of planar digraphs, with applications*, Algorithmica, 5 (1990), pp. 509–527.

[52] R. Tamassia and I. G. Tollis, *A unified approach to visibility representations of planar graphs*, Discrete & Comput. Geom., 1 (1986), pp. 321–341.

[53] ———, *Planar grid embedding in linear time*, IEEE Trans. Circuits and Systems, 36 (1989), pp. 1230–1234.

[54] ———, *Representations of graphs on a cylinder*, SIAM J. Discrete Math., 4 (1991).

[55] R. Tamassia, I. G. Tollis, and J. S. Vitter, *Parallel construction of planar graph layouts*, Manuscript, 1990.

[56] R. Tamassia and J. S. Vitter, *Optimal cooperative search in fractional cascaded data structures*, in Proc. 2nd ACM Symposium on Parallel Algorithms and Architectures, 1990, pp. 307–316.

[57] S. Wimer, I. Koren, and I. Cederbaum, *Floorplans, planar graphs, and layouts*, IEEE Trans. Circuits and Systems, 35 (1988), pp. 267–278.

[58] S. K. Wismath, *Characterizing bar line-of-sight graphs*, in Proc. 1st ACM Symposium on Computational Geometry, 1985, pp. 147–152.

[59] ———, *Weighted visibility graphs of bars and related flow problems*, in Algorithms and Data Structures, Lecture Notes in Computer Science, Vol. 382, Springer-Verlag, Berlin, New York, 1989, pp. 325–334.

[60] C. K. Yap, *Parallel triangulation of a polygon in two calls to the trapezoidal map*, Algorithmica, 3 (1988), pp. 279–288.