Optimal Parallel Algorithms for Transitive Closure and Point Location in Planar Structures

Roberto Tamassia and Jeffrey S. Vitter

Technical Report No. CS-89-20

March 1989

Optimal Parallel Algorithms for Transitive Closure and Point Location in Planar Structures

(preliminary version)

Roberto Tamassia[1] Jeffrey Scott Vitter[2]

Dept. of Computer Science
Brown University
Providence, R. I. 02912–1910

March 1989

[1]Support was provided in part by a research grant from Cadre Technologies, Inc.
[2]Support was provided in part by an NSF Presidential Young Investigator Award with matching funds from IBM and by an NSF research grant.

Abstract

We present parallel algorithms for several graph and geometric problems, 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.

1 Summary

Planar *st*-graphs, which include series-parallel graphs as a special case, were first introduced in [LEC] in connection with a planarity testing algorithm, and they have subsequently been used in a host of applications, dealing with partial orders [KeR], planar graph embedding [CNA, Tam], graph planarization [OzT], graph drawing [DiT, DTT], floor planning [WKC], planar point location [EGS, PrTa, PrTb], visibility [OtW, RoT, TaTa, Wis], motion planning [RiU], and VLSI layout compaction [WKC].

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 EREW PRAM 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.

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 $O(\log^2 n)$ time on an EREW PRAM and $O(\log n)$ time on a CREW PRAM with $M(n)$ processors, where the best known upper bound on $M(n)$ is currently $M(n) = O(n^{2.376})$ [CoW]. 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 computing a topological ordering of the vertices of an acyclic digraph G and of deciding the reachability of a vertex v from a vertex u 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 from [PrTa, TaP] 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.

In Section 3 we give an optimal $O(\log n)$-time, $(n/\log n)$-processor algorithm for constructing the orders $<_{\mathrm{L}}$ and $<_{\mathrm{R}}$ of and 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. (A transitive closure query asks whether there is a path between two specified vertices in G.) 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 an different data structure, updates can be done in $O(1)$ time with n processors and queries take $O(1)$ time with one processor.

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 to obtain an $O(n)$-space data structure (the bridged separator-tree [EGS, LeP]) that supports point location queries in $O(\log n)$ time. This technique is also extended to construct a fully dynamic data structure for point location and to handle nonmonotone subdivisions.

This improves over the previous best results given in [ACG, DaK]. Atallah, Cole, and Goodrich [ACG] give an algorithm to construct a suboptimal $O(n \log n)$-space point location data structure in $O(\log n)$ time with n processors. Dadoun and Kirkpatrick [DaK] show that the hierarchical point location data structure of [Kira] can be constructed in $O(\log n \log^* n)$ time with n processors. 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 [EKA].

In Section 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. We give algorithms for constructing visibility representations of planar st-graph and undirected planar graphs in $O(\log n)$ time, with $n/\log n$ and n processors, respectively. 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 processors. This improves substantially over the algorithm of [JaS] that 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 conclusive remark, our parallel algorithms appear to be simple to implement and eminently practical.

2 Planar st-Graphs

Definition 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 Figure 1a. We assume in this paper, as stated in Definition 1, that the inputed 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 [PrS]. 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 [Col]. 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 on a CREW PRAM in $O(\log^2 n)$ time with n processors [KlR].

Following the development in [PrTa, TaP], 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 were demonstrated to have the following important properties [LEC, TaTa]:

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 ver-

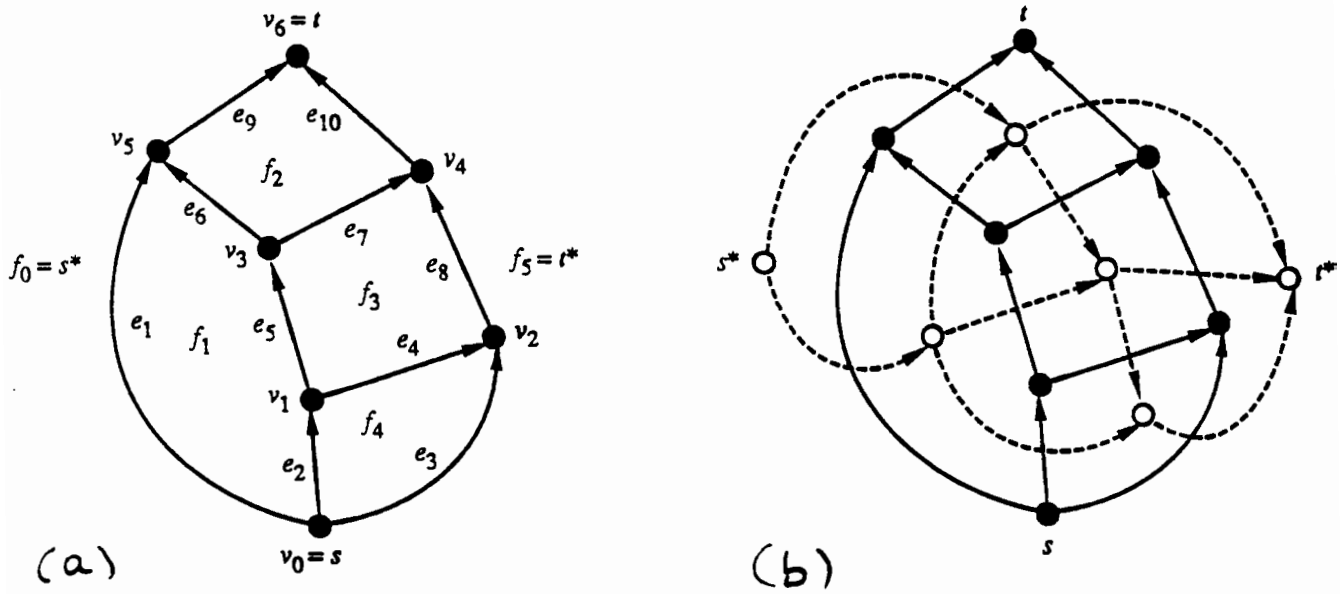

Figure 1: (a) Planar embedding of a typical planar st-graph G, depicting its vertices, faces, and edges. The source s is the lowest vertex, the sink t is the highest vertex, and all edges in G are directed upward. (b) The dual graph G^* [PrTa, TaP].

tex v in the clockwise direction is called $left(v)$, and the face separating them in the counterclockwise direction is called $right(v)$. (See Figure 2a.)

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 highest vertex $high(f)$. (See Figure 2b.)

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 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^*.

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 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 Figure 1, we have $e_2 \uparrow f_3 \uparrow t$ and $e_1 \rightarrow f_3 \rightarrow v_2$. It is shown in [PrTa, TaP] that 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 $y \rightarrow x$. This allows us to define the following two total orders:

Definition 4 The total orders $<_L$ and $<_R$ are defined as

$$x <_L y \iff x \uparrow y \text{ or } x \rightarrow y;$$
$$x <_R y \iff x \uparrow y \text{ or } y \rightarrow x.$$

We define the *left sequence of* G to be the sequence of elements of $V \cup E \cup F$ sorted with respect to $<_L$, and the *right sequence of* G to be the sequence of elements of $V \cup E \cup F$ sorted with respect to $<_R$.

For example, the left and right sequences for the graph in Figure 1 are

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

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

This left sequence is also pictured as a path in Figure 2c. The formal underpinning of the orders $<_L$ and $<_R$ can be found in the theory of planar lattices [Kam, KeR].

The importance of the total orders $<_L$ and $<_R$ is that they can be used to answer transitive closure queries:

Theorem 1 [PrTa, TaP] *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 Theorem 1 and give EREW PRAM algorithms for constructing the fully dynamic (sequential) data structure of [PrTa, TaP] in $O(\log n)$ time with only $n/\log n$ processors, and we also give parallel algorithms for dynamic queries and updates.

Theorem 2 *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.*

In addition, the subgraph consisting of all paths from u to v can be generated in $O(\log k)$ time and $n/\log n$ processors on an EREW PRAM, where k is the length of the boundary of the subgraph. Or alternatively, if only one path is desired, the shortest of the leftmost and rightmost paths of the subgraph can be generated in time proportional to the logarithm of the length of the path.

The fully dynamic data structure that is constructed is the one developed in [PrTa, TaP]. It 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.

Proof of Theorem 2: Our algorithm constructs the data structure of [PrTa, TaP] based on the left and right sequences of G. By Theorem 1, 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. 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, 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 [CoV].

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 relation:

1. Each vertex (except for s and t) constructs the ordering consisting of its rightmost incoming edge, followed by itself, followed by its leftmost outgoing edge. (See Figure 2a.)

2. Each interior face constructs the ordering consisting of its topmost left edge, followed by itself, followed by its bottommost right edge. (See Figure 2b.)

The source vertex s constructs the ordering s^*, followed by itself, followed by its leftmost outgoing edge, and t forms the ordering consisting of its rightmost incoming edge, followed by itself, followed by t^*. List ranking is then done to merge the orderings into a full sequence, as shown in Figure 2c. Lemma 1 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 [PrTa, TaP]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 1 *List ranking of the above local orderings produces the left sequence of G.*

Proof: The local orderings produced above do not induce any cycles, since each ordering is consistent with the total order $<_L$. The rest of the proof consists in showing by contradiction that the orderings 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

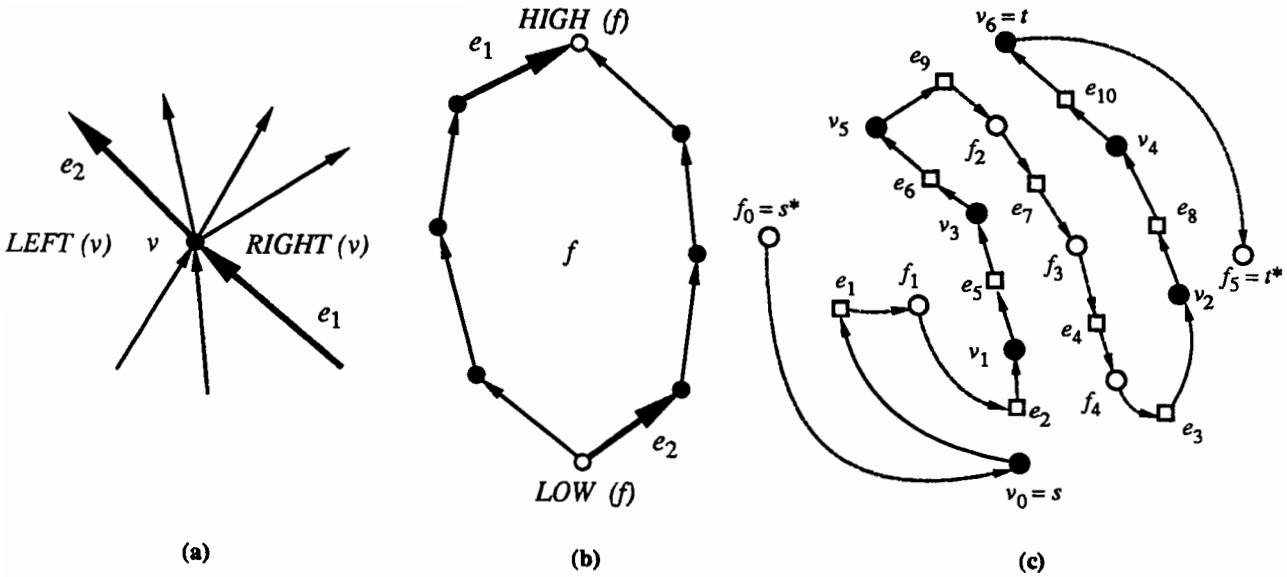

Figure 2: Parallel construction of the left sequence. (a) The ordering $e_1\ v\ e_2$ formed by Rule 1. (b) The ordering $e_1\ f\ e_2$ formed by Rule 2. (c) The left sequence in list form shown for the graph G in Figure 1.

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. 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 $<_L$ is an extension of the partial order $\uparrow$ imposed by the directed edges of the graph gives us the following corollary:

Corollary 1 *A topological ordering of the vertices in a planar st-graph can be obtained in $O(\log n)$ time using $n/\log n$ processors on an EREW PRAM, which is optimal.*

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 in [AGK]:

Corollary 2 *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 [ACG] for two-set dominance counting and obtain the following theorem.

Theorem 3 *Given a planar st-graph G, the number of vertices reachable from v can be computed by a 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?" [CEG]. Without loss of generality, assume that θ is the vertical direction. By orienting the edges of the convex subdivision from bottom to top, 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 2 for the dual graph G^*, we get the following corollary:

Corollary 3 *A fully dynamic $O(n)$-space $O(\log n)$-sequential time data structure for the contact chain query problem along a fixed direction θ can be constructed by an EREW PRAM in $O(\log n)$ time using $n/\log n$ processors, which is optimal.*

The balanced tree data structure of Theorem 2 can be also be used to handle dynamic updates and queries in parallel:

Theorem 4 *After the preprocessing discussed in Theorem 2, determining if there is a path from u to v can be answered in $O(\log\log n)$ time, and dynamic updates can be done in $O(\log^2 \log n)$ time on an EREW PRAM with $\log n/\log\log n$ processors, assuming that the processors know beforehand the ancestors of u and v in the trees in the data structure.*

Proof Sketch: There is a path from u to v if and only if in the trees associated with the left and right sequences of G leaf u precedes leaf v in the symmetric order, that is, u is in the left subtree and v is in the right subtree of the least common ancestor of u and v. This can be checked in $O(\log\log n)$ time by list ranking along the paths of the tree from the leaves u and v to their least common ancestor. Regarding updates, it is shown in [PrTa, TaP] that when a vertex or edge is inserted or deleted into G the left and right sequences can be maintained by making $O(1)$ insertions/deletions in the sequence and then swapping two portions of the sequence. If red-black balanced trees are used, these operations correspond to splits and merges of $O(\log n)$ subtrees. The merges can be done in $O(\log\log n)$ parallel stages, where each stage consists of a number of distinct pairwise merges, each requiring $O(\log\log n)$ time in order to propagate the color flips via list ranking. This gives the $O(\log^2 \log n)$ time bound for updates. ∎

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. This gives us the following algorithm:

Theorem 5 *After the preprocessing discussed in Theorem 2, determining if there is a path from u to v can be answered on an EREW PRAM in constant time with one processor, and dynamic updates can be done in in constant time with n processors.*

4 Planar Point Location

For the geometric terminology used in this section and a description of various point location techniques, the reader is referred to [PrS].

Definition 5 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* 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 [PrS]. 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 s and t are associated with the vertices at $-\infty$ and $+\infty$ of S, and each edge is directed according to increasing ordinate.

Definition 6 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 that intersects σ_1 and σ_2 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 $i < j$.

Point location can be performed by a binary search on the set of separators $\Sigma = \{\sigma_1, \ldots, \sigma_{p-1}\}$, 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 [LeP]. 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 (u, v) is stored only once, at node σ_j, 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 Figure 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 [LeP]. By applying the fractional cascading technique to the separator tree, we obtain a *bridged separator tree*, which still uses $O(n)$ space and supports queries in $O(\log n)$ time, which is optimal [EGS].

Definition 7 A *regular* subdivision is a monotone subdivision having no pair of regions r and r' such that $r \uparrow r'$. (See Figure 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 2 *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 regular subdivision S:

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.

2. Form a balanced 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. Also, preprocess T so that a lowest common ancestor query in T, denoted $lca(x, y)$, can be performed in constant sequential time.

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 the only incoming edge of v;
   if $lca(left(u, v), right(u, v)) = lca(left(v, w), right(v, w))$ **then**
   connect (u, v) to (v, w) by means of bidirectional pointers;
   end;
   if $outdeg(w) = 1$ **then begin**
   let (w, z) be the only outgoing edge of w;
   if $lca(left(w, z), right(w, z)) = lca(left(v, w), right(v, w))$ **then**
   connect (w, z) to (v, w) by means of bidirectional pointers
   end
   end;

   Store each doubly-connected list of edges so obtained into the node of T that is the lowest common ancestor of the edges. Each node's list is the set of proper edges of that node, sorted from bottom to top.

4. Connect the lists of proper edges stored in adjacent nodes of T by means of bridges, according to the fractional cascading scheme.

The correctness of the algorithm follows from Lemma 2. Step 1 is performed using the techniques developed in the previous section. Using the technique of [ScV], Step 2 takes $O(\log n)$ time with $n/\log n$ processors. In Step 3, the iterations of the **for**-loop are independent, so that we can allocate one processor per group of $\log n$ edges and do the computation in $O(\log n)$ steps with $n/\log n$ processors. Finally, we apply the parallel fractional cascading technique of [ACG] to complete the parallel construction of the bridged separator tree. This proves the following:

Lemma 3 *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.*

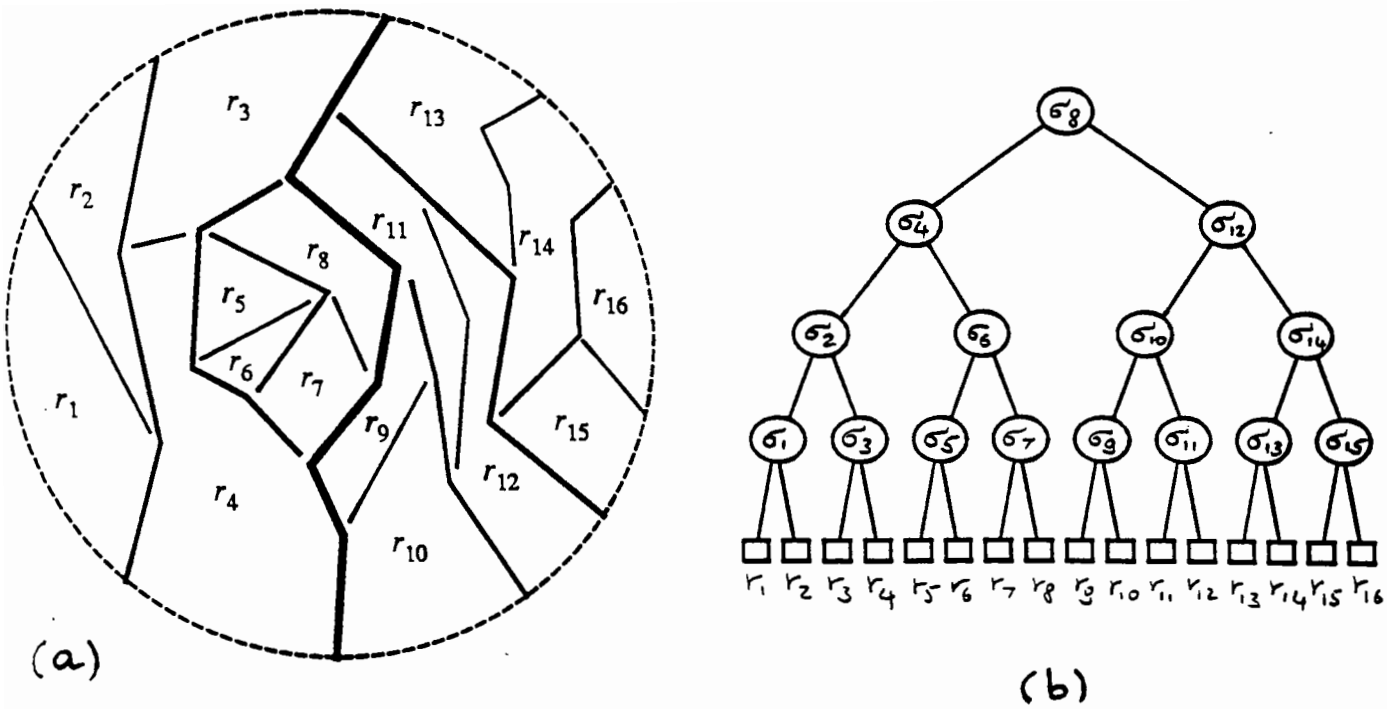

Figure 3: Construction of the separator tree for a regular subdivision: (a) Regular subdivision S with the chains of proper edges visualized. (b) Separator tree for S.

Definition 8 We call two regions r' and r'' *vertically consecutive* if $r' \uparrow r''$ and there is no region r with $r' \uparrow r \uparrow r''$. It can be shown that there is a unique monotone chain from $high(r')$ and $low(r'')$, called a *channel*, and that all channels are vertex disjoint [PrTa, PrTb].

If the subdivision S is monotone, but not regular, we transform S into an equivalent regular subdivision by duplicating some edges, as in [PrTa]. 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 Figure 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 $r_1, r_2, \ldots, r_p$ of regions from the left sequence and determine the vertically consecutive pairs.

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.

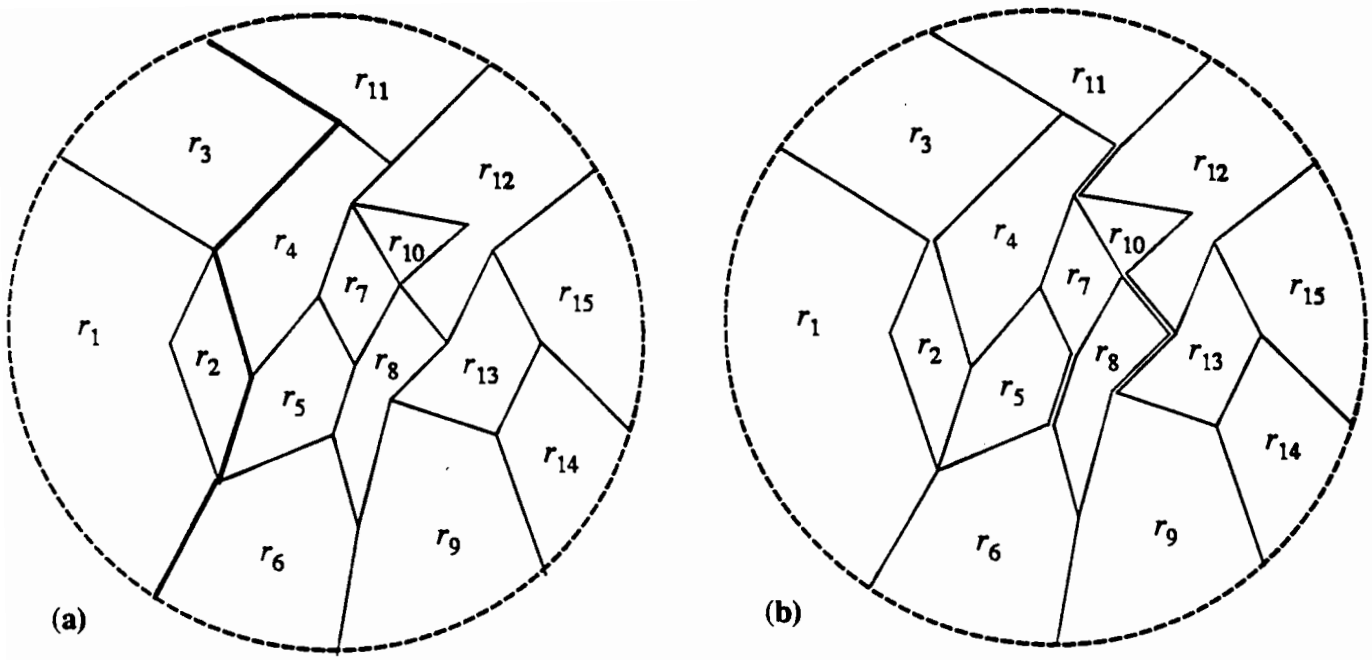

Figure 4: (a) A monotone subdivision S and (b) the corresponding regular subdivision S^*. Notice the clusters of regions $r_2 \cup r_3$, $r_6 \cup r_7$, and $r_9 \cup r_{10} \cup r_{11}$ [PrTa].

The algorithm referred to in Theorem 2 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 *The regular subdivision S^* associated with a monotone subdivision S can be computed by a 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 3 and 4, we obtain the following theorem:

Theorem 6 *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 above algorithm can be modified to construct the fully dynamic point location data structure described in [PrTa] 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 [Col]. 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 [Kirb].

For nonmonotone subdivisions we perform a preliminary triangulation step and then apply the technique for monotone subdivisions. Triangulation can be performed by an EREW PRAM in $O(\log n)$ time with n processors [ACG]. We get the following:

Theorem 7 *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 an EREW PRAM in $O(\log n)$ time using n processors.*

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 [O'Ro], VLSI layout [HsP, SLM, WKC], motion planning [GuY, RiU], and graph drawing [DiT, TaTb].

Definition 9 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 10 A *visibility representation* Γ for a directed graph G maps each vertex v of G to a horizontal segment $\Gamma(v)$ and each edge (u, v) to a vertical segment $\Gamma(u, v)$ that has its lower endpoint on $\Gamma(u)$, its upper endpoint on $\Gamma(v)$, and does not intersect any other horizontal segment. (See Figure 5a.) If G is an undirected planar graph, a visibility representation 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 [DHV, TaTa, Wis].

The visibility graph of a set of n segments can be computed in $O(n \log n)$ sequential time [SLM], which is optimal. It can also be constructed in parallel by a CREW PRAM in $O(\log n)$ time with n processors [ACG], or by a EREW PRAM in $O(\log^2 n)$ time with $n/\log n$ processors [SaW]. As regards the construction of visibility representations, sequential $O(n)$-time algorithms are presented in [DiT, RoT, TaTa].

Our parallel algorithm for constructing a visibility representation Γ for a planar st-graph G is a variation of the sequential ones given in [DiT, RoT, TaTa]:

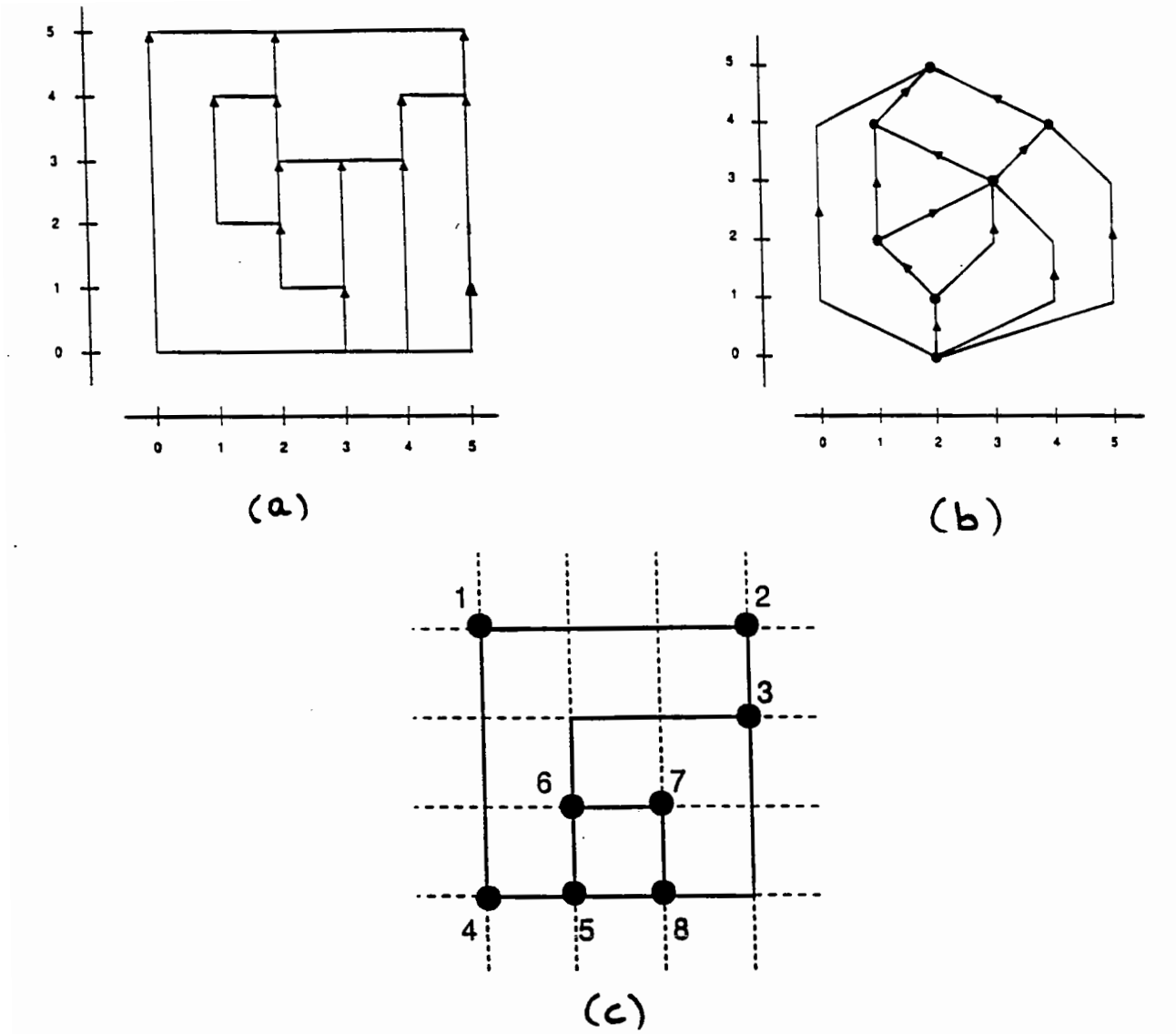

Figure 5: (a) Visibility representation for a planar st-graph G. (b) A planar upward polyline grid drawing of G. (c) A planar orthogonal grid drawing of an undirected graph.

1. Construct the dual graph G^* of G. Also, compute $low(x)$, $high(x)$, $left(x)$ $right(x)$, for each vertex or edge x of G.

2. Compute a topological ordering of the vertices of G, and let $\alpha(v)$ be the rank of a vertex v.

3. Compute a topological ordering of the vertices of the dual graph G^*, and let $\beta(f)$ be the rank of a face f.

4. For each vertex v, let $\Gamma(v)$ be the segment with endpoints $(\beta(left(v)), \alpha(v))$ and $(\beta(right(v)), \alpha(v))$.

5. For each edge (u, v), let $\Gamma(u, v)$ be the segment with endpoints $(\beta(left(u, v)), \alpha(u))$ and $(\beta(left(u, v)), \alpha(v))$.

Each step can be performed by an EREW PRAM in $O(\log n)$ time using $n/\log n$ processors: Step 1 uses the construction of G^* outlined in the proof of Theorem 2. Steps 2 and 3 are based on Corollary 1, and Steps 4 and 5 require constant work independently performed at each vertex and edge. This gives us the following:

Theorem 8 *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.*

Given a 2-connected undirected planar graph G, we orient the edges of G so that the resulting digraph is a planar *st*-graph, and then apply the previous theorem. Such an orientation of G can be computed by a CRCW PRAM in $O(\log n)$ time with n processors using the *st*-numbering algorithm of [MSV]. By well-known simulation results this yields $O(\log^2 n)$ time complexity on a EREW PRAM.

Theorem 9 *Let G be a 2-connected (undirected) planar graph with n vertices. A visibility representation for G with integer coordinates and $O(n^2)$ area can be computed by a CRCW PRAM in $O(\log n)$ time using 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, which is spread over the broad spectrum of computer science, is surveyed in [EaT, TBB]. This problem has received in recent years increasing theoretical interest (see, e.g., [DTT, FPP]).

Definition 11 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 *polyline drawing* is such that 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.

Orthogonal drawings are typical in circuit layout applications. Grid drawings of small area are important because of the finite resolution of display devices (and of the human eye). Upward drawings are commonly used in the representation of hierarchic structures, where all edges "flow" in the same direction. Examples include PERT networks, ISA hierarchies, and subroutine-call graphs. Examples of planar grid drawings are shown in Figure 5b,c.

The computation of a visibility representation is a fundamental step for several graph drawing algorithms [DiT, TaTb, TaTc], and the results of Theorems 8–9 can be used to design efficient parallel algorithms for graph drawing:

Theorem 10 *Let G be a planar st-graph with n vertices. A planar upward polyline 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, which is optimal.*

If we have a 2-connected undirected planar graph G, we can convert it to planar *st* graph as we did to get Theorem 8. This gives us the following result:

Theorem 11 *Let G be a 2-connected (undirected) planar graph with n vertices. A planar orthogonal grid drawing for G with $O(n)$ bends and $O(n^2)$ area can be computed by a CRCW PRAM in $O(\log n)$ time using n processors.*

The bounds on the area and the number of bends are asymptotically optimal [DTT, Sto]. Theorems 9 and 11 can also be extended to 1-connected graphs.

6 References

[AGK] F.N. Afrati, D.Q. Goldin & P.C. Kanellakis, "Efficient Parallelism for Structured Data: Directed Reachability in S-P Dags," Dept. Computer Science, Brown Univ., Tecnical Report CS-88-07, 1988.

[ACG] M.J. Atallah, R. Cole & M.T. Goodrich, "Cascading Divide-and Conquer: A Technique for Designing Parallel Algorithms," *Proc. 28th IEEE Symp. on Foundations of Computer Science* (1987).

[CEG] B. Chazelle, H. Edelsbrunner & L.J. Guibas, "The Complexity of Cutting Convex Polytopes," *Proc. 19th ACM Symp. on Theory of Computing* (1987).

[CNA] N. Chiba, T. Nishizeki, S. Abe & T. Ozawa, "A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees," *J. of Computer and System Sciences* 30 (1985), 54–76.

[Col] R. Cole, "Parallel Merge Sort," *Proc. 27th IEEE Symp. on Foundations of Computer Science* (1986).

[CoV] R. Cole & U. Vishkin, "Approximate and Exact Parallel Scheduling with Applications to List, Tree, and Graph Problems," *Proc. 27th IEEE Symp. on Foundations of Computer Science* (1986).

[CoW] D. Coppersmith & S. Winograd, "Matrix Multiplication via Arithmetic Progression," *Proc. 28th ACM Symp. on Theory of Computing* (1987).

[DaK] N. Dadoun & D.G. Kirkpatrick, "Parallel Processing for Efficient Subdivision Search," *Proc. ACM Symp. on Computational Geometry* (1987).

[DiT] G. Di Battista & R. Tamassia, "Algorithms for Plane Representations of Acyclic Digraphs," *Theoretical Computer Science* 61 (1988), 175–198.

[DTT] G. Di Battista, R. Tamassia & I.G. Tollis, "Area Requirement and Symmetry Display in Drawing Graphs," *Proc. ACM Symp. on Computational Geometry* (1989 (to appear)).

[DHV] P. Duchet, Y. Hamidoune, M. Las Vergnas & H. Meyniel, "Representing a Planar Graph by Vertical Lines Joining Different Levels," *Discrete Mathematics* 46 (1983), 319–321.

[EaT] P. Eades & R. Tamassia, "Algorithms for Automatic Graph Drawing: An Annotated Bibliography," Dept. of Computer Science, Brown Univ., Technical Report CS-89-09, 1989.

[EKA] M. Edahiro, I. Kokubo & T. Asano, "A New Point-Location Algorithm and its Practical Efficiency – Comparison with Existing Algorithms," *ACM Trans. on Graphics* 3 (1984), 86–109.

[EGS] H. Edelsbrunner, L.J. Guibas & J. Stolfi, "Optimal Point Location in a Monotone Subdivision," *SIAM J. Computing* 15 (1986), 317–340.

[FPP] H. de Fraysseix, J. Pach & R. Pollack, "Small Sets Supporting Fary Embeddings of Planar Graphs," *Proc. 20th ACM Symp. on Theory of Computing* (1988).

[GuY] L.J. Guibas & F.F. Yao, "On Translating a Set of Rectangles," in *Advances in Computing Research, vol. 1*, F.P. Preparata (Ed.), ed., JAI Press Inc., Greenwich, CT, 1983, 61–77.

[HsP] M.Y. Hsueh & D.O. Pederson, "Computer-Aided Layout of LSI Circuit Building-Blocks," *Proc. IEEE Int. Symp. on Circuits and Systems* (1979).

[JaS] J. Ja'Ja' & J. Simon, "Parallel Algorithms in Graph Theory: Planarity Testing," *SIAM J. Comput.* 11 (1982), 314–328.

[Kam] T. Kameda, "On the Vector Representation of the Reachability in Planar Directed Graphs," *Information Processing Letters* 3 (1975), 75–77.

[KeR] D. Kelly & I. Rival, "Planar Lattices," *Canadian J. Mathematics* 27 (1975), 636–665.

[Kira] D.G. Kirkpatrick, "Optimal Search in Planar Subdivisions," *SIAM J. Computing* 12 (1983), 28–35.

[Kirb] D.G. Kirkpatrick, "Establishing Order in Planar Subdivisions," *Discrete & Computational Geometry* 3 (1988), 267–280.

[KlR] P.N. Klein & J.H. Reif, "An Efficient Parallel Algorithm for Planarity," *Proc. 27th IEEE Symp. on Foundations of Computer Science* (1986).

[LeP] D.T. Lee & F.P. Preparata, "Location of a Point in a Planar Subdivision and its Applications," *SIAM J. Computing* 6 (1977), 594–606.

[LEC] A. Lempel, S. Even & I. Cederbaum, "An Algorithm for Planarity Testing of Graphs," *Theory of Graphs, Int. Symposium*, Rome (1966).

[MSV] Y. Maon, B. Schieber & U. Vishkin, "Parallel Ear Decomposition Search (EDS) and *st*-Numbering in Graphs," *Theoretical Computer Science* 47 (1986), 277–298.

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

[OtW] R.H.J.M. Otten & J.G. van Wijk, "Graph Representations in Interactive Layout Design," *Proc. IEEE Int. Symp. on Circuits and Systems*, New York (1978).

[OzT] T. Ozawa & H. Takahashi, "A Graph-planarization Algorithm and its Applications to Random Graphs," *Lecture Notes in Computer Science (Graph Theory and Algorithms)* 108 (1981), 95–107.

[PrS] F.P. Preparata & M.I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[PrTa] F.P. Preparata & R. Tamassia, "Fully Dynamic Techniques for Point Location and Transitive Closure in Planar Structures," *Proc. 29th IEEE Symp. on Foundations of Computer Science* (1988).

[PrTb] F.P. Preparata & R. Tamassia, "Fully Dynamic Point Location in a Monotone Subdivision," *SIAM J. Computing* 18 (1989 (to appear)).

[RiU] I. Rival & J. Urrutia, "Representing Orders by Translating Convex Figures in the Plane," *Order* 4 (1988), 319–339.

[RoT] P. Rosenstiehl & R.E. Tarjan, "Rectilinear Planar Layouts of Planar Graphs and Bipolar Orientations," *Discrete & Computational Geometry* 1 (1986), 342–351.

[SaW] J.E. Savage & M.G. Wloka, "Parallel Constraint Graph Generation," *Proc. Decennial Caltech Conf. on VLSI* (1989).

[ScV] B Schieber & U. Vishkin, "On Finding Lowest Common Ancestors: Simplification and Parallelization," *VLSI Algorithms and Architectures (Proc. AWOC '88, Corfu, Greece, 1988), J.H. Reif (Ed.), Lecture Notes in Computer Science* 319 (1988), 111–123.

[SLM] M. Schlag, F. Luccio, P. Maestrini, D.T. Lee & C.K. Wong, "A Visibility Problem in VLSI Layout Compaction," in *Advances in Computing Research, vol. 2*, F.P. Preparata (Ed.), ed., JAI Press Inc., Greenwich, CT, 1985, 259–282.

[Sto] J.A. Storer, "On Minimal Node-Cost Planar Embeddings," *Networks* 14 (1984), 181–212.

[Tam] R. Tamassia, "A Dynamic Data Structure for Planar Graph Embedding," *Automata, Languages and Programming (Proc. 15th ICALP, Tampere, Finland, 1988), T. Lepisto and A. Salomaa (Eds.), Lecture Notes in Computer Science* 317 (1988), 576–590.

[TBB] R. Tamassia, G. Di Battista & C. Batini, "Automatic Graph Drawing and Readability of Diagrams," *IEEE Transactions on Systems, Man and Cybernetics* SMC-18 (1988), 61–79.

[TaP] R. Tamassia & F.P. Preparata, "Dynamic Maintenance of Planar Digraphs, with Applications," *Algorithmica* (1989 (to appear)).

[TaTa] R. Tamassia & I.G. Tollis, "A Unified Approach to Visibility Representations of Planar Graphs," *Discrete & Computational Geometry* 1 (1986), 321–341.

[TaTb] R. Tamassia & I.G. Tollis, "Efficient Embedding of Planar Graphs in Linear Time," *Proc. IEEE Int. Symp. on Circuits and Systems*, Philadelphia (1987).

[TaTc] R. Tamassia & I.G. Tollis, "Planar Grid Embedding in Linear Time," *IEEE Trans. on Circuits and Systems* CAS-36 (1989 (to appear)).

[WKC] S. Wimer, I. Koren & I. Cederbaum, "Floorplans, Planar Graphs, and Layouts," *IEEE Trans. on Circuits and Systems* 35 (1988), 267–278.

[Wis] S. Wismath, "Characterizing Bar Line-of-Sight Graphs," *Proc. ACM Symp. on Computational Geometry*, Baltimore, Maryland (1985).