Fully Dynamic Techniques for Reachability in Planar sT-graphs

(Preliminary Report)

S. Sairam
Robert F. Cohen
Roberto Tamassia
Jeffrey S. Vitter

Department of Computer Science
Brown University
Providence, Rhode Island 02912–1910

Technical Report No. CS-91-02
December 1990

Fully Dynamic Techniques for Reachability in Planar sT-graphs

(Preliminery Report)

S. Sairam
Robert F. Cohen
Roberto Tamassia
Jeffrey S. Vitter

Department of Computer Science
Brown University
Providence, Rhode Island 02912-1910

Abstract

We present fully dynamic techniques to solve the reachability problem in single source multi sink planar *st*-graphs (*sT*-graphs). We provide an $O(n)$ space data structure which supports transitive closure queries in $O(\log^2 n)$ time. We then show how to maintain this data structure under updates (insertion of an edge and expansion of a vertex, and their inverses) to the graph G, in time $O(\log n)$.

(December 1990)

1 Summary

In this paper we investigate *planar sT-graphs*, i.e. embedded planar digraphs that admit an upward drawing and have a single source s and a set of sinks T.

Our work is an extension of the study of *planar st-graphs*, which are planar acyclic graphs embedded with exactly one source s and one sink t both on the external face. Suppose G is a planar st-graph with vertex set V, edge set E, and face set F. Then G admits two total orders (known as the *leftist* and *rightist* orders of G that are used to support reachability queries. Data structures and algorithms have been presented that allow $O(n)$ preprocessing and $O(1)$ query time for static planar st-graphs [1] [5] [7]. $O(\log n)$ update and query time for dynamic graphs[10], and optimal $O(\log n)$ running time using $n/\log n$ processors in the EREW PRAM model for parallel processing[13]. The central result of this paper is to extend these results to the larger class of planar sT-graphs.

Given a planar sT-graph G, we build an auxiliary planar st-graph G_{st} and show, in a static environment, that a transitive closure query on G can be found using a transitive closure query on G_{st}. For the dynamic and parallel environments, we build an another auxiliary planar st-graph G_1 and enhance the data structures stored for G_1. Transitive closure queries to G can be found through the enhanced data structures for G_1. we also show that an update on G can be performed through a constant number of update operations on G_1.

The rest of the paper is organized as follows: Section 2 gives definitions and reviews related previous results. Section 3 describes a sequential algorithm for determining transitive closure in planar sT-graph. Section 4 presents the graph theoretic basis for our result. Section 5 gives the dynamic algorithm.

2 Planar st-Graphs and Planar sT-Graphs

Basic definitions on graphs and posets can be found in textbooks such as [1] and [3].

Let G be a directed graph, for brevity digraph, and v a vertex of G. We denote by $d^-(v)$ the *indegree* of v, i.e. the number of incoming edges of v, and by $d^+(v)$ the *outdegree* of v, i.e. the number of outgoing edges of v. A *source* of G is vertex s with $d^-(s) = 0$. A *sink* of G is vertex t with $d^+(t) = 0$. A *transitive* edge of G is an edge $e = (u, v)$ such that there exists another directed path from u to v consisting of at least two edges.

Definition 1 A *planar st-graph* is a planar acyclic digraph G with exactly one source, s, and exactly one sink, t, which is embedded in the plane so that s and t are on the boundary of the external face.

These graphs were first introduced in the planarity testing algorithm of Lempel *et al.* [8]

Definition 2 A *planar upward drawing* of an embedding of a planar graph G is a planar drawing such that every edge $(u, v) \in G$ is a curve monotonically increasing in the vertical direction.

Given a planar st-graph G, several important properties of G are expressed by the following lemmas:

Lemma 1 *[8] Every vertex of G is on some directed path from s to t.*

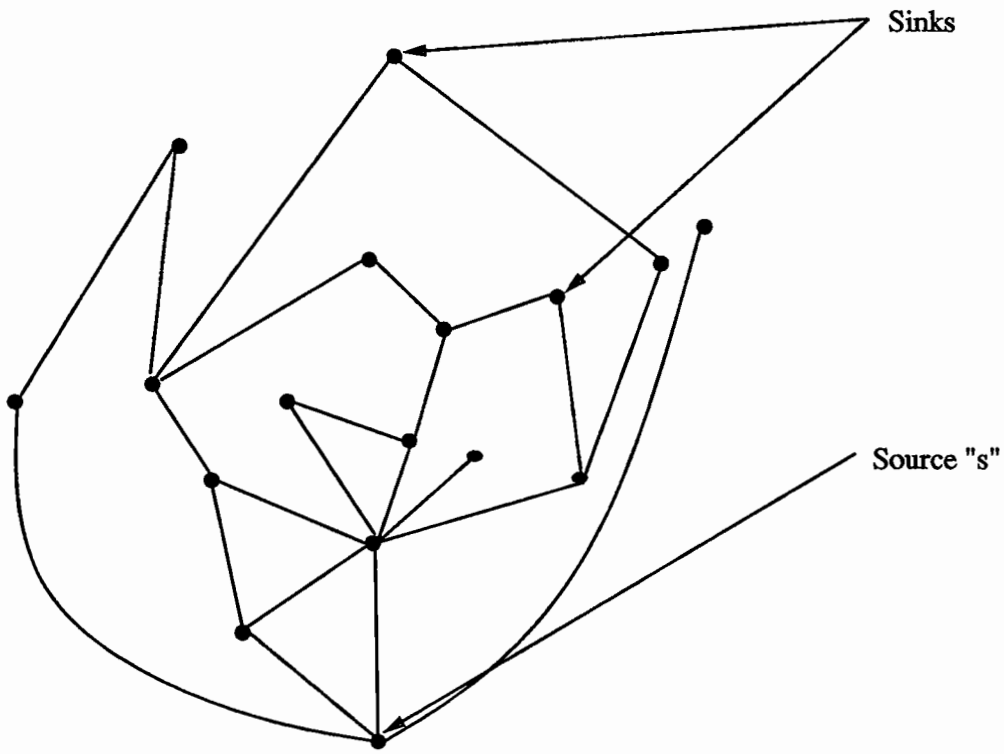

Figure 1: A planar sT-graph G

Lemma 2 *[12] For every vertex v of G, the incoming (outgoing) edges appear consecutively around v.*

Lemma 3 *[12] For every face f of G, the boundary of f consists of two directed paths with common origin and destination.*

Lemma 4 *[2] [6] G admits a planar upward drawing.*

Definition 3 A *planar sT-graph* is a planar digraph G with exactly one source, s, and a set of sinks, T, which is embedded in the plane so that G admits an upward drawing. An example is given in figure 1

Given a planar sT-graph G, similar important properties of G are expressed by the following lemmas:

Lemma 5 *G is acyclic and is embedded such that source s and at least one sink $t \in T$ on the boundary of the outer face.*

Proof: Immediate from the planar upward embedding of G. □

Lemma 6 *Every vertex of G is on a path from s to some $t \in T$.*

Proof: Suppose v is a vertex. Follow any path in G beginning at v. If we arrive at a vertex u with $d^+(u) > 0$, then continue along any outgoing edge of u. Since G is acyclic, we eventually arrive at a vertex t with $d^+(t) = 0$, and hence $t \in T$.

A similar argument shows that a reverse path must end at a source. Since s is the only source, there exists a path from s to u. □

Let Q be a poset (partially ordered set), where $\ll$ denotes the partial order on the elements of Q. The *Hasse diagram* (also called *covering digraph*) of Q is a digraph G whose vertices are the elements of Q, and such that $e = (u, v)$ is an edge of G if and only if $u \ll v$ and there is no other element x of Q such that $u \ll x \ll v$. If edge $e \in G$, we also say that $e \in \ll$. G is acyclic and has no transitive edges. Hasse diagrams are usually represented by straight-line drawings such that for each edge (u, v) the ordinate of vertex u is smaller that that of vertex v.

A *planar lattice* is a poset whose Hasse diagram is a planar st-graph. Also, every planar st-graph without transitive arcs is the Hasse diagram of some planar lattice. Several properties of planar lattices are described in. [7].

A poset Q has a *greatest lower bound* 0 if for all $u \in Q$, $0 \ll u$. The Hasse diagram of a poset with a greatest lower bound is a planar sT-graph. Everyplanar sT-graph without transitive arcs is the Hasse diagram of some poset with greatest lower bound.

A *linear extension* of a poset Q is a total order $<$ on the elements of Q such that for any two elements u and v of Q $u \ll v$ implies $u < v$. A linear extension corresponds to a topological sorting of the vertices of the Hasse diagram of Q. We say that Q has *dimension* k if G admits k linear extensions $<_1, <_2, ..., <_k$, such that $u \ll v$ if and only if $u <_1 v$, $u <_2 v$, ..., $u <_k v$, and k is minimum.

It is known that planar lattices have dimension 2 [1] [5] [7] and that posets with greatest lower bound have dimension at most 3 [14].

2.1 Review of Planar st-Graphs

Lemma 7 *[1] [5] [7] Let G be a planar st-graph with n vertices. There exist two total orders on the vertices of G, denoted $<_l$ and $<_r$ (called the leftist and the rightist orders), such that there is a directed path from u to v if and only if $u <_l v$ and $u <_r v$. Furthermore, orders $<_l$ and $<_r$ can be computed in $O(n)$ time. Subsequent transitive closure queries can be performed in $O(1)$ time per query.*

Lemma 7 is based on the fact that the underlying partial order of a planar lattice admits a "complementary" partial order (see [7]).

In the following definitions, the concepts of *left* and *right* refer to the orientation of the edges. For example, the face to the left of an edge (u, v) is the face containing edge e which appears on the left side when traversing edge (u, v) from vertex u to vertex v. Also, the reader will find it convenient to visualize the planar st-graph G as being drawn in the plane with edges monotonically increasing in the vertical direction (see Lemma 4).

Given vertices u and v of G such that there exists a path from u to v, the set of paths from u to v defines a planar st-graph with source u and sink v which is an induced subgraph of G. The two paths that form the external boundary of this subgraph will be called the *leftmost path* and *rightmost path* from u to v, respectively. For example, the external boundary of G consists of the leftmost and rightmost paths from s to t.

Let G^* be the digraph obtained from the dual graph of G as follows

1. the dual edge e^* of an edge e is directed from the face to the left of e to the face to the right of e;

2. the external face of G is dualized to two vertices of G^*, denoted s^* and t^*, which are incident with the duals of the edges on the leftmost and rightmost paths from s to t, respectively.

Vertices s^* and t^* can be thought of as being the "left" and "right external face" of G, respectively. It is simple to verify that G^* is a planar st-graph with source s^* and sink t^*. [9] [12] Notice that G^* might have multiple arcs.

Let V, E, and F denote the set of vertices, edges, and faces of G, respectively, where F has elements s^* and t^* representing the external face. We will show that the orders $<_l$ and $<_r$ can be extended to the set $V \cup E \cup F$, thereby giving a unique total order of all topological constituents of G.

First, for each element x of $V \cup E \cup F$, we define vertices $low(x)$ and $high(x)$, and faces $left(x)$ and $right(x)$, as follows:

1. If $x = v \in V$, we define $low(v) = high(v) = v$. Also, we denote by $left(v)$ and $right(v)$ the two faces that separate the incoming and outgoing edges of a vertex $v \neq s, t$. For $v = s$ or $v = t$, we conventionally define $left(v) = s^*$ and $right(v) = t^*$.

2. If $x = e \in E$, we define $low(e)$ and $high(e)$ as the tail and head vertices of e, respectively. Also, we denote by $left(e)$ and $right(e)$ the faces on the left and right side of e, respectively.

3. If $x = f \in F$ and $f \neq s^*, t^*$, we denote by $low(f)$ and $high(f)$ the two vertices that are the common origin and destination of the two paths forming the boundary of f. For $f = s^*$ or $f = t^*$, $low(f)$ and $high(f)$ are undefined. Also, we define $left(f) = right(f) = f$.

Definition 4 We say that x is *below* y, denoted $x \uparrow y$, if there is a path in G from $high(x)$ to $low(y)$. Also, we say that x is *to the left of* y, denoted $x \rightarrow y$, if there is a path in G^* from $right(x)$ to $left(y)$.

Definition 5 We define relations $<_l$ and $<_r$ on $V \cup E \cup F$, as follows:

$$x <_l y \Leftrightarrow x \uparrow y \text{ or } x \rightarrow y$$

$$x <_r y \Leftrightarrow x \uparrow y \text{ or } y \rightarrow x$$

Theorem 1 *The relations $<_l$ and $<_r$ on $V \cup E \cup F$ are total orders.*

We also note that there is a path in G from vertex u to vertex v if and only if $u <_l v$ and $u <_r v$, since such path exists if and only if $u \uparrow v$.

Definition 6 We define the *left-sequence* of G as the sequence of elements of $V \cup E \cup F$, sorted according to $<_l$ (leftist order). The *right-sequence* of G is defined similarly with respect to $<_r$ (rightist order).

[10] gives algorithms to dynamically maintain the leftist and rightist orders. In particular, they support the following dynamic operations:

1. INSERT(e,u,v,f;f_1,f_2): Add edge $e = (u, v)$ inside face f, which is decomposed into faces f_1 and f_2, with f_1 to the left of e and f_2 to its right.

2. DELETE(e,u,v,f_1,f_2;f): Delete edge $e = (u, v)$ and merge the two faces f_1 and f_2 formerly on the opposite sides of e into a new face f.

3. EXPAND(e,f,g,v;v_1,v_2): Expand vertex v into vertices v_1 and v_2, which are connected by a new edge e with face f to its left and face g to its right.

4. CONTRACT(e,f,g,v_1,v_2;v): Contract edge $e = (v_1, v_2)$ and merge its endpoints into a new vertex v. Faces f and g are to the left and right of g respectively.

Each operation is allowed if the resulting graph is itself a planar st-graph.
These operations are sufficient since:

Lemma 8 *[10] Let G_0 be the trivial planar st-graph consisting of a single vertex. Any planar st-graph with n vertices can be assembled starting from G_0 by means of $O(n)$* INSERT *and* EXPAND *operations, and can be disassembled to yield G_0 by means of $O(n)$* DELETE *and* CONTRACT *operations.*

The algorithm represents the orders $<_l$ and $<_r$ by means of two balanced binary search trees, denoted T_L and T_R. The left to right order of the leaves of the tree represent the leftist and rightist sequence. The dynamic operations INSERT, DELETE, EXPAND, and CONTRACT, as well as transitive closure queries can be performed in $O(n)$ time per operation. So we have:

Theorem 2 *Given a planar st-graph G, there exists a dynamic data structure that supports transitive closure queries as well as operations* INSERT, DELETE, EXPAND, *and* CONTRACT *in $O(\log n)$ time per operation using $O(n)$ space.*

3 Sequential algorithm for reachability in Planar sT-graphs

In this section we show that the result of Lemma 7 can be extended to planar sT-graphs. The strategy is given a planar sT-graph G, we build a planar st-graph G_{st} such that the leftist and rightist orders for G_{st}, along with a third total order is used to answer transitive closure queries on G.

Definition 7 The *terminating face* for a $t \in T$ is the face above t. In other words f is to the left of the leftmost incoming edge of t and to the right of the rightmost incoming edge of t.

Definition 8 Given a planar sT-graph $G = (V, E)$ define the *planar st completion* of G to be $G_{st} = (V_{st}, E_{st})$ such that:

1. $V_{st} = V \cup \{t_{st}\}$ where $t_{st} \notin V$
2. $E_{st} = E \cup E_e \cup E_i$ with:

(a) $E_e = \{(x, t_{st}) | x \in T$ and x on the external face of $G\}$

(b) $E_i = \{(x, HIGH(f)) | x \in T$, f is the terminating face of $x\}$

Notice that the edges in $E_i \cup E_e$ partition the faces they cross. Suppose that f is a face of G and $e_1, \ldots, e_j \in E_i \cup E_e$ are edges that cross f in left to right order. Each e_i terminates at the same vertex. This partitions f into $f_0, \ldots, f_j$ also in left to right order.

Lemma 9 *G_{st} is a planar st-graph*

Proof: G_{st} clearly has a single source s and a single sink t_{st} since we added outgoing edges from every $t \in T$. Given an upwardly oriented embedding of G we can draw T_{st} such that it's y coordinate is greater than that of all vertices of G. In this way, the edges of $E_i \cup E_e$ will be oriented upward when added to this embedding. Therefore G is acyclic. □

Definition 9 The total orders $<_l$ and $<_r$ for a planar sT-graph G are defined to be the total orders $<_l$ and $<_r$ (as defined by Tamassia) for the planar st completion of G, G_{st}.

Since G has dimension 3, we need to find a third dimension $<_3$ such that $<_l \cap <_r \cap <_3 = G$ (minus transitive edges). $<_3$ will need to be a total order that preserves E while eliminating E_e and E_i.

For each $x \in V_{st} \cup E_{st} \cup F_{st}$ define $dist(v, p)$ to be the number of edges of $E_e \cap E_i$ in the path p from $HIGH(x)$ to t_{st}. Define $mind(x) = \min\{dist(x, p) |$ all paths p from $HIGH(v)$ to $t_{st}\}$. d divides $V_{st} \cup E_{st} \cup F_{st}$ into equivalence classes with $x \sim y$ if $mind(x) = mind(y)$.

Define $<_3$ as follows:

1. If $x \sim y$ then $x <_3 y$ and $y <_3 x$
2. if $mind(x) < mind(y)$ then $x <_3 y$.

Lemma 10 $<_l \cap <_r \cap <_3 = G$

Proof: Since $E_{st} = E \cup E_i \cup E_e$ we need to show the following:

1. If $e \in E$ then $e \in <_l \cap <_r \cap <_3$
2. If $e \in E_i \cup E_e$ then $e \notin <_l \cap <_r \cap <_3$

By the above argument, we have $<_l \cap <_r \cap <_3 = G_{st} \cap <_3$. Suppose $e = (x, y) \in E$ then $e \in G_{st}$. Since some (but not necessarily all) paths from x to t_{st} cross e, $mind(x) \leq mind(y)$ so $x <_3 y$ and $e \in <_l \cap <_r \cap <_3$.

Suppose $e = (x, y) \in E_i \cup E_e$ then $e \in G_{st}$. This implies that $x \in T$ so the outdegree of x in G_{st} is 1, namely e. This means that $mind(x) = mind(y) + 1$. Therefore, $mind(x) > mind(y)$. This implies that $y <_3 x$, but not $x <_3 y$. So $e \notin <_l \cap <_r \cap <_3$. □

Theorem 3 *There exists a sequential algorithm to find the transitive closure of a planar sT-graph. The algorithm requires $O(n)$ preprocessing time and $O(1)$ time per query.*

Proof: Lemma 7 shows a $O(n)$ algorithm to find $<_l$ and $<_r$. The equivalence classes defined by $mind(v)$ can be found in $O(n)$ time using a modified breadth-first search starting at t_{st}.

The correctness of the algorithm is a direct consequence of lemma 7. □

4 Dynamic Algorithm for reachability in planar sT-graph

In the previous section we saw a sequential algorithm to find the transitive closure in a planar sT-graph; unfortunately the third order we created is not amenable to dynamic updates, so in this section we take a slightly different approach to find a dynamic algorithm for the same problem. We create a data structure which can be updated in $O(\log n)$ time, at the expense of making the query time $O(\log^2 n)$. We assume that an upward embedding of the input graph is given, that is for each vertex the cyclic ordering of it's neighbors is given. The embedding is represented in the standard form by doubly connected lists. If the embedding information is not given it can be determined in $O(n^2)$ time [4].

The strategy for solving the reachability problem in sT-Graphs is to build an auxiliary st-graph G_1 from the given sT-graph, and determine the reachability in G by making queries on the leftist and rightist sequences of G_1.

G_1 is constructed as follows:

1 For every face f with one or more sinks in it, introduce a new vertex v_f called the face vertex.

2 Join v_f and the highest vertex h_f of f by a pseudo edge e_f called the green edge. For the outermost face we introduce a new vertex t called the "sink" of the resulting graph which is used in lieu of h_f.

3 Join every sink x on the boundary of f (except possibly h_f) to v_f by the edge e_x. For every sink x (except the right most one) name the face to the right of e_x f_x.

G_1 is an st-Graph where the source is the same as in G and sink is the new node t, an example is shown in figure 2

We denote by L_1 (R_1) the leftist (rightist) order of G_1. Due to the extra edges added each face f gets split into many sub-faces; we denote by f_l and f_r the leftmost and rightmost such subfaces. A trivial fact is that for all divided faces f_l (f_r) follows f_r (f_l) in the leftist (rightist) orders.

Before we describe necessary and sufficient conditions to test for reachability in G in terms of the auxiliary graph G_1, we need to define a few more terms.

Definition 10 A green edge e_f is called a "blocking edge" for a vertex u if all the paths from u to t go through e_f.

Definition 11 A pair (f_l, f_r) is called a "blocking pair" for a vertex u in G_1 if the relative ordering of f_l, f_r and u in L_1 and R_1 are

L_1: $f_l\ u\ f_r$

R_1: $f_r\ u\ f_l$

Definition 12 A bounding box for a face f (of the original graph G) in G_1 is a region defined as follows (figure 3):

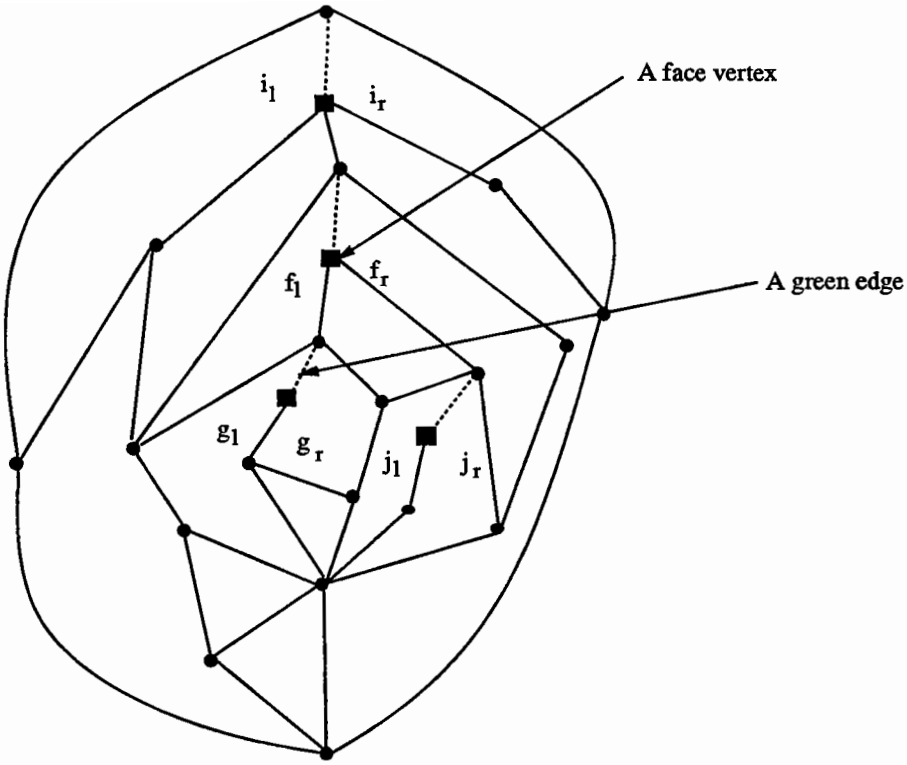

Figure 2: The Auxiliary st-Graph G_1 created from G

Left-Path: In the reverse order take the left bounding edge (into h_f) of f_l and from then on take the rightmost non green incoming edge (in reverse) until s is reached. We denote this by Bl_f.

Right-Path: Take the right bounding edge (into h_f) of f_r and from then on take the leftmost incoming non green edge (in reverse) until s is reached. We denote this by Br_f.

We are guaranteed that such "Left" and "Right" paths exist because G is a sT-Graph. We denote by "bounding box of f" the region enclosed by the left and right paths; and by sB_f the "low" of this region.

Lemma 11 *If a vertex v is reachable from another vertex u in G then it is also reachable in G_1.*

Proof: Obvious since G_1 is constructed from G by adding edges. □

We now look at a lemma which shows us that the (f_l, f_r) pairs behave like nested parenthesis, in both leftist and rightist orders, that is they never overlap.

Lemma 12 *For all (f_l, f_r) and (g_l, g_r) pairs in G_1 such that g_l follows f_l in L_1 (g_r follows f_r in R_1), the only possible relative orderings in L_1 and R_1 are*

L_1: $f_l\ g_l\ g_r\ f_r$, $f_l\ f_r\ g_l\ g_r$

R_1: $f_r\ g_r\ g_l\ f_l$ *or* $f_r\ f_l\ g_r\ g_l$

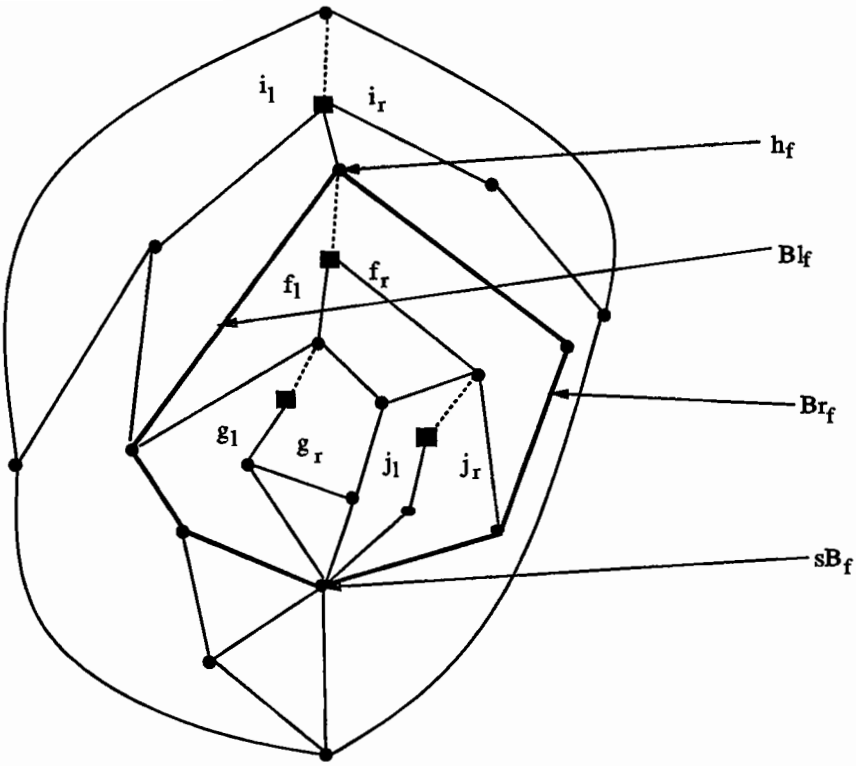

Figure 3: Bounding box of face f in G_1

Proof: We show the proof for the leftist order; the one for the rightist order is similar. To prove the lemma we need to consider two cases:

1. g_l is between f_l and f_r : We have to show that g_r is between them as well. By construction we know that h_g is either inside the bounding box of f or on the right boundary. In either case h_g occurs between f_l and f_r in the ordering. However g_r has the same high vertex as g_l by construction; therefore g_r is between f_l and f_r as well.

2. g_l occurs after f_r: It is obvious then that g_r occurs after f_r as well since it occurs after g_l in the leftist order.

□

Lemma 13 *If u is a vertex inside the bounding box of a face f, then there are no paths from u to either the left bounding path Bl_f or to the right bounding path Br_f (except to the high vertex h_f of the face).*

Proof: Let us suppose that there is a path p from u to a vertex v in Bl_f (the case of Br_f is similar). By the definition of the bounding box we know that p can't be a non green edge path. Look at the first green edge e_h when traveling on this path in reverse from v to u. By definition of the bounding box we construct Bl_f by always taking the rightmost incoming non green edge. The first green edge in p (in reverse) therefore enters v. This however means that there is a non green edge to the right of e_h (by construction); we would therefore have taken that edge while constructing Bl_f (note since v is not h_f this non green edge is not on the right boundary Br_f) which is a contradiction. Therefore there are no such paths. □

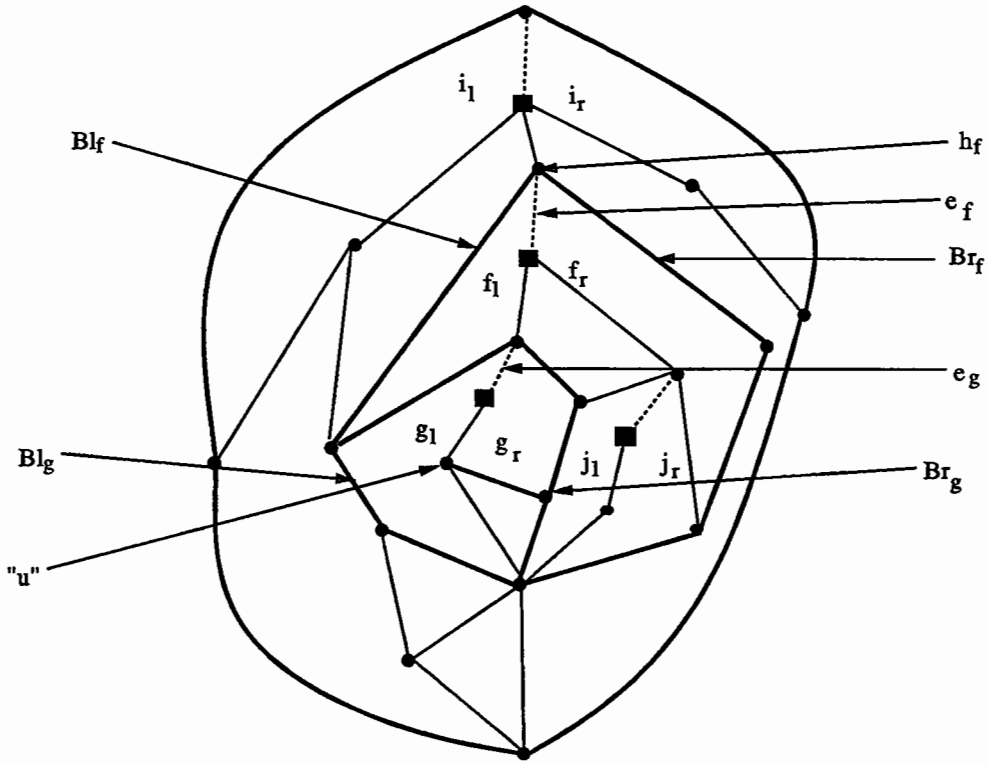

Figure 4: Nested Bounding boxes of faces i, f and g in G_1

Lemma 14 *A pair (f_l, f_r) is a blocking pair for the vertex u if and only if u is inside the bounding box of f.*

Proof:

- $\Rightarrow$ Let u be inside the bounding box of f. From lemma 13 we know that there are no paths from u to either of the bounding paths. This simply means that u occurs in between f_l and f_r in both the leftist and rightist orders and hence (f_l, f_r) is a blocking pair for u.
- $\Leftarrow$ Let (f_l, f_r) be a blocking pair for u. From the leftist ordering we know that u is either inside the bounding box of f or on the right boundary Br_f. From the rightist ordering we can deduce that u is either inside the bounding box or on the left boundary Bl_f. Therefore we conclude that u is inside the bounding box of f.

□

Next we look at a lemma which tells us that the bounding boxes of various faces never overlap; that is either they nest nicely, or they are separate (see figure 4)

Lemma 15 *Let (f_l, f_r) and (g_l, g_r) be two blocking pairs for a vertex u then the relative orderings of the two pairs in L_1 is*

L_1: $f_l\ g_l\ u\ g_r\ f_r$

if and only if the corresponding relative ordering in R_1 is

R_1: f_r g_r u g_l f_l

Proof: We show only the ($\Rightarrow$) direction the other one is similar. To prove that, consider the bounding boxes of faces f and g. Let e be the last edge on Bl_g (that is the last edge on the left boundary of the face g). By the relative ordering of f_l and g_l in L_1 we know that e is either inside the bounding box of f or on the right boundary Br_f (since that is the only way g_l can occur between f_l and f_r). This is equivalent to saying that the high vertex of face g h_g is either inside the bounding box of f or on the right boundary. Since (g_l, g_r) and (f_l, f_r) are blocking pairs for u, u is inside the bounding box for both g and f. Therefore by construction, there are paths from u to both h_g and h_f. This however implies that h_g cannot be on the right boundary of f's bounding box because then we would have a path from u to the right boundary (Br_f) of f's bounding box, which is a contradiction of lemma 13. Therefore h_g is inside the bounding box of face f This implies that in the rightist order g_r occurs between f_r and f_l (see figure 4) and thus the required rightist ordering between f_l, g_l, f_r, g_r follows by virtue of lemma 12. □

Lemma 16 *A pair (f_l, f_r) is a blocking pair for a vertex u if and only if the corresponding green edge e_f is a blocking edge for the same vertex.*

Proof:

- ($\Rightarrow$) Let e_f be a blocking edge of u. Consider the bounding box of f. There are two options either u is inside the bounding box or outside. If it is inside then by lemma 14 (f_l, f_r) is a blocking pair of u. If u is outside the bounding box of f, then there is a path from u to Bl_f orBr_f (because there is a path from u to e_f), but then we get a path from u to t which is not blocked by e_f which is a contradiction.

- ($\Leftarrow$) Let (f_l, f_r) be a blocking pair of u. From lemma 14 we know that u is inside the bounding box of f and from lemma 13 there are no paths from u to either Bl_f or Br_f. Consider any path p from u to t; since it cannot use edges of either Bl_f or B_f p it is constrained to use vertex v_f to reach t. Therefore by construction e_f is an edge on p. □

Lemma 17 *If (f_l,f_r) and (g_l g_r) are two blocking pairs of a vertex u then their relative orderings in L_1 are*

L_1: f_l g_l u g_r f_r

if and only if the blocking edge e_g occurs before e_f on any path p from u to t in G_1. A similar statement can be made about the rightist order R_1.

Proof:

- ($\Rightarrow$) Since all the paths from u to t are blocked by the edges e_f and e_g either e_f occurs before e_g in all the paths or e_g occurs before e_f in all of them (otherwise we will get a directed cycle which is a contradiction). Let us suppose e_f occurs before e_g in all the paths. By construction we know that f_r (g_r) occurs immediately after e_f (e_g) in the leftist ordering (see figure 4). Also by the fact that e_g can be reached from e_f we know that e_g occurs after e_f in L_1. This implies the relative orderings of e_f, e_g, f_r, g_r in L_1 are:

$L_1 : e_f\ f_r\ e_g\ g_r$

which is a contradiction, hence the claim holds.

- ($\Leftarrow$) From lemma 16 we know that (f_l,f_r) and (g_l g_r) are blocking pairs of the vertex u, therefore the leftist ordering has to be as claimed, otherwise we will contradict the previous case.

□

Theorem 4 *For all vertices u in $G \cap G_1$ there exists a unique blocking edge e_f such that no other blocking edge e_g occurs before e_f on any path p from u to t. We will call e_f the closest blocking edge of u and the corresponding blocking pair (f_l, f_r) the closest blocking pair.*

Proof: Let (f_l, f_r) be the closest blocking pair of u if

1. (f_l, f_r) is a blocking pair for u and
2. There is no other blocking pair (g_l, g_r) of u such that the relative ordering of f_l, f_r, g_l and g_r in L_1 (or R_1) is $g_l\ f_l\ f_r\ g_r$.

Such a blocking pair is well defined because according to lemmas 12 and 15 the blocking pairs are nicely nested. We claim that the corresponding blocking edge e_f is the closest blocking edge. This follows from lemmas 16 and 17 since every other blocking pair (g_l g_r) envelopes this closest blocking pair (in the leftist and rightist orders) and therefore it's edge e_g occurs after e_f on any path p from u to t. □

Lemma 18 *If y is reachable from x in G_1 then it is reachable in G as well if and only if the relative ordering of i_l, i_r, x and y in L_1 and R_1 (where (i_l, i_r) is the closest-blocking pair of x) are*

L_1: $i_l\ x\ y\ i_r$

R_1: $i_r\ x\ y\ i_l$

Proof:

- ($\Rightarrow$) By the relative ordering of i_l, x, y and i_r we know that both x and y are in the bounding box of i. Let us assume that y is not reachable from x. This means that every path from x to y contains at least one green edge. Consider a minimum green edge path p from x to y. Let e_j be the first green edge on this path. Since (i_l, i_r) is a blocking pair for both x and y there are no paths from either x or y to the boundaries Bl_i and Br_i. This means the path from x to y is fully contained in the bounding box of i. Both j_l and j_r therefore occur between i_l and i_r in L_1 and R_1. Consider the bounding box of j, there are two cases (figure 5)
  1. x is inside the bounding box of j: This would imply that (g_l, g_r) is a closer blocking pair of x than (i_l, i_r) which is a contradiction

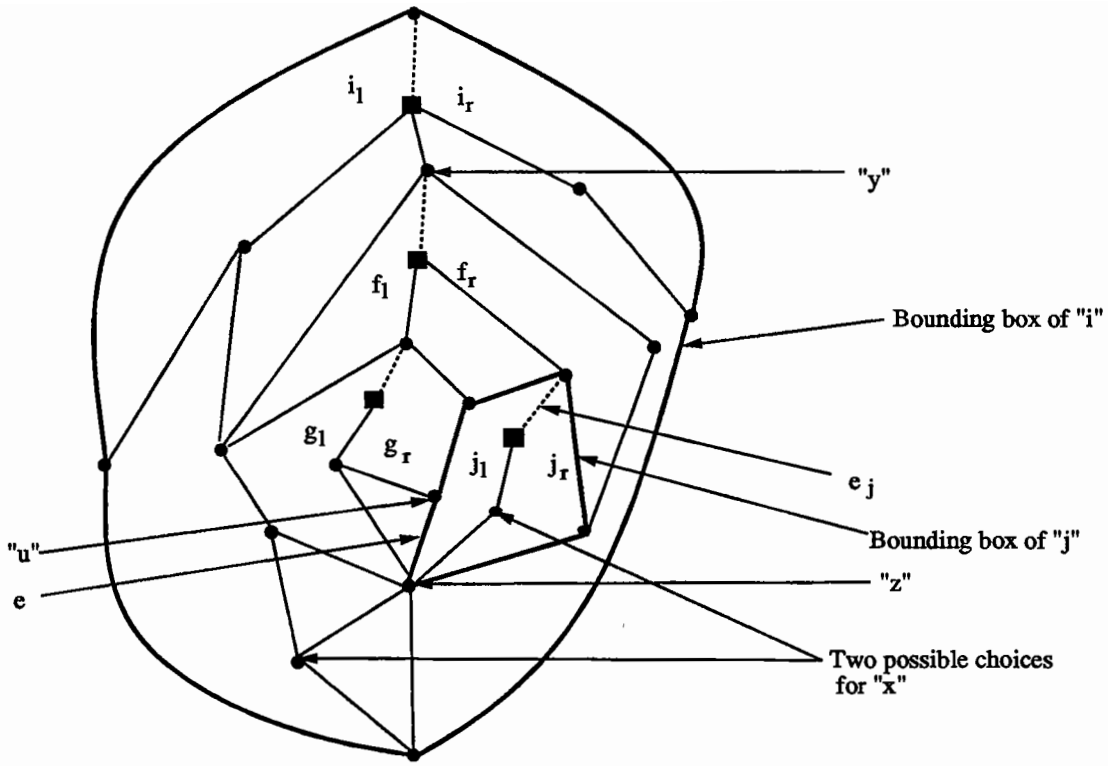

Figure 5: Reachability between vertices x and y

2. x is outside the bounding box of j: This would mean that the path p cuts the bounding box of j (say at a vertex z on Bl_j). But then we can construct a new path p' which consists of the path p until x, then a segment of Bl_j till h_j (the end point of e_j) and then the rest p. We therefore get a path p' from x to y with fewer than minimum number of green edges, a contradiction.

- ($\Leftarrow$) Let us assume that y is outside the bounding box of i. This however means that every path from x to y contains e_i (since the only way out of the bounding box from u is to go through the high vertex h_i. This leads us to a contradiction because y is reachable from x. Therefore y is inside the bounding box of i and the claim on the orderings follows.

□

Theorem 5 *A vertex v is reachable from another vertex u in G if and only if it is reachable in G_1 and the relative ordering of f_l, f_r, u and v in L_1 and R_1 (where (f_l, f_r) is the closest-blocking pair of u) are*

L_1: $f_l\ u\ v\ f_r$

R_1: $f_r\ u\ v\ f_l$

Proof: Follows directly from lemmas 18 and 11. □

To find whether a vertex v is reachable from a vertex u in G all we need to do is to test for their reachability in G_1 and if so see whether the closest-blocking pair of u includes

v in the leftist and rightist orders. This is simple in a parallel setting since the leftist and rightist orders for a planar st-graphwith n nodes can optimally be found in $O(\log n)$ time with $n/\log n$ processors on a EREW PRAM [13]. Therefore all we need to do is to check each (f_l, f_r) pair to see if it is a blocking pair, and then find out which of those is the closest blocking pair. All of these can be done in $O(\log n)$ time with $n/\log n$ processors. Therefore we get:

Theorem 6 *There exists a optimal EREW PRAM algorithm to compute transitive closure queries on a planar sT-graphin $O(\log n)$ time with $n/\log n$ processors.*

In the dynamic setting it is difficult to do a search which depends on two linear orders, we therefore need a little more machinary before we can do queries quickly, the next lemma gives us just that.

We know that if a pair (f_l, f_r) encloses u in L_1 then either u is in the bounding box of f or u is on Br_f. If u is on Br_f then we call the (f_l, f_r) pair an *impostor* pair for u.

Lemma 19 *Let (f_l, f_r) be a blocking pair for u and (g_l, g_r) be an impostor pair, then the relative ordering of f_l, f_r, g_l and g_r in L_1 is*

L_1: $f_l\ g_l\ g_r\ f_r$

Proof: Since (g_l, g_r) is an impostor pair, u is on Br_g (see figure 5). Since both pairs enclose u in L_1, by lemma 12 the only possible relative orderings in L_1 are

1. $f_l\ g_l\ g_r\ f_r$
2. $g_l\ f_l\ f_r\ g_r$

Assume that the (g_l, g_r) pair encloses the (f_l, f_r) pair. By assumption u is on Br_g, and is inside the bounding box of f. Let e be the edge on Br_g that enters u and let h be the face to the right of e. Since u is on Br_g h comes after g_r in L_1, but since u and hence e is inside the bounding box of f, h comes before f_r in L_1, which is a contradiction. □

This gives us a method to find the closest blocking pair of u by conducting a binary search on L_1. The details are as follows

step 1. In L_1 assign $+1$ to f_l for every face f of the original graph G and -1 to f_r and 0 to all the other elements in the sequence.

step 2. With this weight assignment find the rank r of u in L_1. This quantity gives us the total number of blocking and impostor pairs that enclose u.

step 3. Let $k = \lceil r/2 \rceil$.

step 4. Find the longest prefix of L_1 (ending before u) with value $k-1$. The item just to the right is f_l for the k^{th} innermost (f_l, f_r) that encloses u.

step 5. Query R_1 to determine whether (f_l, f_r) is a blocking pair or an impostor pair for u.

step 6. If (f_l, f_r) is a blocking pair then recursively conduct the search to the right of f_l, else recurse to the left.

Each step runs in at most $O(logn)$ time, and the search steps $O(logn)$ steps, therefore the total time is $O(log^2 n)$.

Lemma 20 *The algorithm outlined above correctly pinpoints the closest blocking pair of* u.

Proof: Follows from lemma 19 □

Using this data structure we can perform transitive closure queries in time $O(log^2 n)$.

The next section shows how to dynamically maintain this data structure.

5 Dynamic Updates to sT-Graphs

In this section we define a complete set of update operations on a planar sT-graph G and show the resulting restructuring of the leftist and rightist orders on the associated auxiliary planar st-graphG_1. The update operations are defined as follows:

Insert(e,u,v,f;g,h): Add edge $e = (u, v)$ inside face f, which is decomposed into faces g and h, with g to the left of e and h to its right.

Delete(e,u,v,g,h;f): Delete edge $e = (u, v)$ and merge the two faces g and h formerly on the opposite sides of e into a new face f.

Expand(e,f,g,v;v_1,v_2): Expand vertex v into vertices v_1 and v_2, which are connected by a new edge e with face f to its left and face g to its right. If $f = g$ thenv_2 will be a sink.

Contract(e,f,g,v_1,v_2;v): Contract edge $e = (v_1, v_2)$ and merge its endpoints into a new vertex v. Faces f and g are to the left and right of g respectively.

Each operation is allowed if the resulting graph is a planar sT-graph.

The following lemma shows that these operations are sufficient to build or disassemble any planar sT-graph:

Lemma 21 *Let* G_0 *be the trivial planar sT-graph consisting of a single vertex. Any planar sT-graph* G *with* n *vertices can be constructed starting from* G_0 *by means of* $O(n)$ Insert *and* Expand *operations and can be disassembled my means of* $O(n)$ Delete *and* Contract *operations.*

Proof: By induction on the number of vertices: Suppose $G = (V, E)$ is a planar sT-graph with n vertices. Since G is planar there must exist a vertex v such that $d^-(v) + d^+(v) < 5$. Let G' be the graph induced by $V - \{v\}$. Then G can be constructed from G' by a single Expand operation followed by at most three Insert operations.

A similar argument shows the result for disassembling G.

□

In the previous section we showed that transitive closure queries on G can be found by accessing the leftist and rightist orders for the auxiliary graphG_1. We now show how to maintain The leftist and right orders for G_1 when updates are made to G. We show here the algorithms for updating G_1. We show that each dynamic operation on G translates to at most a constant number of dynamic operations on the planar st-graphG_1:

For each face of G we maintain a balanced tree of it's subfaces ordered according to the embedding. Dynamic operations on this tree can be done in $O(\log n)$ time in the worst case.

- Insert(e,u,v,f; g, h):

  Edge insertion can only be done across a single face, since we are limited to a given embedding of G. Suppose e is an edge being inserted. If e does not cross a sink edge in G_1, we simply have to insert e in G_1. Otherwise, both new faces created by inserting e will contain sinks, and hence face vertices. Notice that the destination vertex of e will be the high vertex of the lower of the new faces.

  The following performs the modifications to G_1:

  Let f_u (f_v) be the subface containing u (v) on it's boundary (assume either f_u is to the left of f_v or $f_u = f_v$, the other case is analogous). If f_u and f_v are the same then we simply call INSERT(e,u,v,f; g, h).

  Otherwise, let e_u be the rightmost edge incoming to v_f in f_u and e_v be the leftmost edge incoming to v_f in f_v. The following sequence of operations on G_1 then takes care of the insert:

  EXPAND(e_{temp}, f_u, f_v, v_f; v_h, v_g)

  INSERT(e_h, v_h, v, f_v; g_L, g_R) and color e_h green.

  DELETE(e_{temp}, v_h, v_g, f_u, g_L; g_L)

  INSERT(e, u, v, g_L;f_u, g_L)

- Delete(e,u,v,g,h;f):

  Edge deletion causes adjacent faces to be combined. Suppose e is the edge to be deleted. A new sink may be created, and two face vertices may need to be combined.

  Since v has indegree at least 2, it is the high of either face g or h or both. Without loss of generality suppose $v = HIGH(h)$. Then e is on the left bounding path of h, and g is the face to the left of e. Let g_e be the face to the left of e in G_1 (h_L or h will be to the right). We perform the following steps to handle deletion:

  If the outdegree of u is 1, INSERT(e_u, u, v_g, g_e; g_{new}, g_e). If v_g does not exist we create it by inserting the edge from u to $HIGH(g)$ then expanding u to get v_g.

  DELETE(e,u,v,g_e,h_L;g_e). If v_h does not exist we use h instead of h_l

  Combine v_g and v_h (if they exist) and connect the new face vertex v_f to the high of the new face. This can be done with a constant number of dynamic operations on G_1.

  Relabel the new face f.

- Expand(e,f,g,v;v_1,v_2):

  Expanding a vertex in G may create a new sink. If so, the face vertex must be maintained in G_1. Otherwise, we only need expand a single vertex in G_1:

  If $f = g$ then v_2 will be a new sink. If v already is a sink then Then issue EXPAND(e,f_a,f_b,v;v_1,v_2), where f_a is to the left of (v, v_f) and f_b is to the right. If v is not a sink then INSERT(e_v, v, v_f, f_v; f_c, f_d) creates the new vertex, where f_v is the subface of f containing v, and f_c and f_d are new subfaces. (If v_f does

not exist we create it by inserting the edge from v to $HIGH(f)$ then expanding v to get v_f.

If $f \neq g$ then simply issue EXPAND(e,x,y,v;v_1,v_2), where x and y are determined as follows:

If f has a green edge attached to v then $x = f_R$. Similarly, if g has a green edge attached to v then $y = g_L$.

Else If f (g) has a face vertex then $x = f_v$ ($y = f_g$) where f_v (g_v) is the subface of $f(g)$ containing v.

Else $x = f$ ($y = g$).

- Contract(e,f,g,v_1,v_2;v):

  Contracting an edge in G may remove a sink. If so, the face vertex must be maintained in G_1. Otherwise, we only need contract a single edge in G_1:

  If $f = g$ then v_2 is a sink. Then CONTRACT(e,f_a,f_b,v_1,v_2;v), where f_a is to the left of (v_1, v_2) and f_b is to the right. If $outdegree(v_1) > 1$ then v is not a sink so DELETE(e_v, v, v_f, f_a, f_b; f_c) removes the edge (v, v_f). (If v_2 was the only sink in f then we need to remove v_f. To do this we contract v_f into v then deleting the edge $(v, HIGH(f))$.

  If $f \neq g$ then simply issue CONTRACT(e,x,y,v_1,v_2;v), where x and y are determined as follows:

  If f has a green edge attached to v then $x = f_R$. Similarly, if g has a green edge attached to v then $y = g_L$.

  Else If f (g) has a face vertex then $x = f_v$ ($y = f_g$) where f_v (g_v) is the subface of $f(g)$ containing v.

  Else $x = f$ ($y = g$).

Each dynamic operation is implemented with a constant number of dynamic planar st-graph operations. By maintaining T_L and T_R as dynamic trees [11], we can maintain the prefix sum information in $O(\log n)$ time per dynamic operation. Therefore, we have $O(\log n)$ time per operation.

Combining all this, we get:

Theorem 7 *Given a planar sT-graph G, there exists a dynamic data structure that supports transitive closure queries in $O(\log^2)$ time as well as operations* INSERT, DELETE, EXPAND, *and* CONTRACT *in $O(\log n)$ time per operation using $O(n)$ space.*

References

[1] G. Birkhoff, "Lattice Theory," *American Mathematical Society Colloquium Publications* 25 (1979).

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

[3] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[4] M. D. Hutton, "Upward Planar Drawing of Single Source Acyclic Digraphs," University of Waterloo, Masters Thesis , 1990.

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

[6] D. Kelly, "Fundamentals of Planar Ordered Sets," *Discrete Mathematics* 63 (1987), 197–216.

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

[8] A. Lempel, S. Even, and I. Cederbaum, "An Algorithm for Planarity Testing of Graphs," *Theory of Graphs, Int. Symposium* (1966), 215–232.

[9] R.H.J.M. Otten and J.G. van Wijk, "Graph Representations in Interactive Layout Design," *Proc. IEEE Int. Symp. on Circuits and Systems* (1978), 914–918.

[10] F.P. Preparata and R. Tamassia, "Dynamic Planar Point Location with Optimal Query Time," *Theoretical Computer Science* 74 (1990), 95–114.

[11] D.D. Sleator and R.E. Tarjan, "A Data Structure for Dynamic Trees," *J. Computer Systems Sciences* 24 (1983), 362–381.

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

[13] R. Tamassia and J.S. Vitter, "Parallel Transitive Closure and Point Location in Planar Structures," *SIAM J. Computing* 20 (1991 (to appear)).

[14] W.T. Trotter and J. Moore, "The Dimension of Planar Posets," *J. Combinatorial Theory, Series B* 22 (1977), 54–67.