# Fully Dynamic Transitive Closure in Plane Dags with one Source and one Sink

Thore Husfeldt

BRICS Report Series RS-94-30

ISSN 0909-0878 September 1994

# FULLY DYNAMIC TRANSITIVE CLOSURE IN PLANE DAGS WITH ONE SOURCE AND ONE SINK

THORE HUSFELDT

**BRICS**[*]
Department of Computer Science, University of Aarhus
Ny Munkegade, DK-8000 Århus C, Denmark

27th September 1994

Abstract. We give an algorithm for the Dynamic Transitive Closure Problem for planar directed acyclic graphs with one source and one sink. The graph can be updated in logarithmic time under arbitrary edge insertions and deletions that preserve the embedding. Queries of the form 'is there a directed path from $u$ to $v$?' for arbitrary vertices $u$ and $v$ can be answered in logarithmic time. The size of the data structure and the initialisation time are linear in the number of edges.

The result enlarges the class of graphs for which a logarithmic (or even polylogarithmic) time dynamic transitive closure algorithm exists. Previously, the only algorithms within the stated resource bounds put restrictions on the topology of the graph or on the delete operation. To obtain our result, we use a new characterisation of the transitive closure in plane graphs with one source and one sink and introduce new techniques to exploit this characterisation.

We also give a lower bound of $\Omega(\log n/\log\log n)$ on the amortised complexity of the problem in the cell probe model with logarithmic word size. This is the first dynamic directed graph problem with almost matching lower and upper bounds.

---

This work was partially supported by the ESPRIT II Basic Research Actions Program of the EC under contract no. 7141 (project ALCOM II).

[*]Basic Research in Computer Science, Centre of the Danish National Research Foundation

## 1. Introduction

**1.1. Dynamic algorithms.** Two issues motivate the search for dynamic algorithms: From a practical point of view, we want to *solve problems faster* by recomputing parts of the solution as the instance is subject to changes, rather than having to recompute the entire solution from scratch. From a theoretical point of view, we can hope for more *insight* into the nature of the problem and the dynamic realm itself.

Fully dynamic algorithms with logarithmic or polylogarithmic bounds on the update and query times are interesting from both points of view. Firstly, we can hope for implementations that are useful in practice, especially if the data structure is simple. Although impressive other asymptotically sublinear bounds for a variety of problems have been found, the applicability of many of these algorithms is dubious in sight of the complicated data structures involved.

Secondly, the evolving field of dynamic complexity theory identifies problems with these execution times with the class of 'efficiently dynamisable' problems, called **D** for 'dynamic' in [11] or, less euphonically, **incrPOLYLOGTIME** for 'incremental polylogarithmic time' in [10].

Recently, exciting progress has been made in the quest for polylogarithmic update and query times in such different areas as string matching [9], parsing [11] and expression evaluation [2, 3, 11]. The realm of graph theory is more elusive. Many basic graph problems like Spanning Trees, Connected Components, Shortest Paths, etc., reduce to Reachability, which seems to be hard in the dynamic case. For undirected graphs, one can hope for polylog-time solutions as long as the graph is plane, see [6, 5]. For directed graphs, not even that restriction is enough; the best algorithm for Dynamic Reachability on planar digraphs is due to Subramanian [15] and performs in amortised time $O(n^{2/3}\log n)$.

This is interesting to the theoretician because in the parallel realm, the Reachability Problem is easy: Recall that the problem on the general class of directed graphs is complete for **NLOGSPACE**, which is safely contained in $\mathbf{NC}^2$. Our lack of understanding of the interplay between parallel and dynamic computations (or, symbolically, **D** vs. **NC**) could be closely connected to the lack of understanding of the dynamic complexity of the Reachability Problem, see [10].

**1.2. Sketch of result.** Let us briefly state the result of this paper; Sections 2.1 and 2.2 contain more precise definitions.

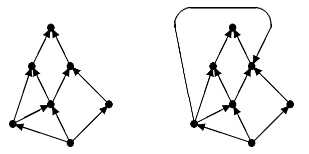


Figure 1. Two plane graphs with one source and one sink

We give an algorithm for the Transitive Closure Problem[1] on directed acyclic graphs that are drawn in the plane without intersecting edges and have exactly one source and one sink, see Figure 1. The algorithm handles queries of the form 'is there a directed path from vertex $u$ to vertex $v$?' and updates that add or remove arbitrary edges, as long as the topology and embedding of the graph are not violated. Updates and queries are processed in time logarithmic in the number of edges of the graph. The data structure can be initialised in linear time and uses linear space.

Together with an easily proved lower bound, this characterises the complexity of the Dynamic Transitive Closure Problem on this class of graphs within a $\log\log n$ factor. The algorithm is pleasantly simple and should be easy to implement efficiently (the most complicated part is the dynamic tree data structure from [14], which also contains a discussion of implentation issues). The analysis is less simple and takes up most of the paper.

**1.3. Relation to previous results.** Two partial solutions to this problem are known:

(1) Tamassia and Preparata [17] consider the special case where the source and the sink are on the same face. They allow the same update operations as the present algorithm, as long as the source and the sink remain on the same face.

(2) Tamassia and Tollis [18] give an algorithm that allows the source and the sink to be on different faces. To this end, they replace the repertory of update operations to get rid of fundamental problems with edge deletion of this approach. They show how

[1]We will use the terms transitive closure and reachability interchangeably when referring to directed graphs.

to simulate edge deletion using a *linear* number of their primitive operations.

Both of these algorithms rely on the following well-known fact: The transitive closure of a plane *st*-graph can be expressed as the intersection of two *total* orders $\leq_L$ and $\leq_R$. Symbolically,

$$u \prec v \quad \leftrightarrow \quad u \leq_L v \wedge u \leq_R v,$$

where we write $\prec$ for the transitive closure. The first paper shows that in the restricted case, $\leq_L$ and $\leq_R$ are easily maintained as the graph changes. The second paper shows under which updates the orderings remain maintainable in the general case. Kelly [8] has shown that for general planar graphs, the number of total orders needed to express the transitive closure as their intersection is unbounded.

The present algorithm subsumes and extends the results from [17, 18] in that it removes the restrictions of both. To this end, we use a different characterisation of reachability. Let us contrast it with the above approach: We maintain two orders (call them $\leq_S$ and $\leq_T$ for a moment) with the property that

$$u \prec v \quad \leftrightarrow \quad \exists w \in V : u \leq_T w \wedge w \leq_S v.$$

It is by no means clear how to handle the existential quantifier over the vertices $V$ of the graph in logarithmic time. Indeed, our algorithm will not be able to identify such a $w$, but merely determines its existence.

**1.4. Roadmap.** This report is organised as follows: Below, we give some preliminary definitions and state the problem precisely. We also derive a lower bound for the problem, using known techniques. In Section 3, we precisely state the above characterisation of the transitive closure in *st*-graphs and briefly re-prove the result of [17]. Section 4 gives an algorithm for the general case that performs well in the *amortised* sense. We then remove the amortisation in Section 5 to get worst-case bounds.

## 2. Preliminaries

**2.1. Graphs.** A graph is *embeddable* on a surface if it can be drawn on the surface such that the edges do not intersect except at their endpoints. A graph is *planar* if it is embeddable in the plane. Using the stereographic projection, it is easily shown that a graph is planar if and only if it is embeddable on the sphere. For a more thorough coverage of planar graphs, see any text on graph algorithms, e.g. [19].

For node $v$ of a digraph we let $\deg^+(v)$ and $\deg^-(v)$ denote its out- and indegree, respectively. A vertex $v$ is a *source* if $\deg^-(v) = 0$, and a *sink* if $\deg^+(v) = 0$. We are now ready to define the class of graphs studied in this paper. The terminology is somewhat awkward (but standard).

**Definition 2.1.** A directed acyclic graph is an *st-graph* if it has exactly one source and one sink. A *spherical st-graph* is a planar *st*-graph that is embedded in the plane. If in that embedding the source and the sink are on the same face, the graph is a *plane st-graph*.

We require *st*-graphs to be acyclic, which agrees with the definition of [18] and disagrees with the one from [16]. Figure 1 shows two spherical *st*-graphs, the left of which is also a plane *st*-graph. The following properties of spherical *st*-graphs can be shown; the last two items may excuse 'spherical' and 'plane' the above definition.

(1) Every vertex is on a simple directed path from $s$ to $t$, called an *st-path*.
(2) In every embedding, the incoming edges to any vertex appear consecutively around the vertex, and so do the outgoing edges; this determines the *left face* $\mathrm{left}(v)$ and the *right face* $\mathrm{right}(v)$ of a vertex, see Figure 2. This implicitly defines an order of the edges appearing around $v$, say, from the leftmost outgoing edge to the leftmost incoming edge in the clockwise direction. We will sometimes refer to this order as the *ordering of the edges around v*.
(3) The boundary of every face consists of two directed paths with common origin and terminus vertices, see Figure 2.
(4) Every spherical *st*-graph can be embedded on the sphere such that all edges are directed upward (i.e., their projection on some fixed direction in the plane is positive). For example, we could embed the graph from Figure 1 by placing the curved arc on the opposite side of the sphere.
(5) Every plane *st*-graph can be embedded in the *plane* such that all edges are directed upward.

In the rest of this paper, $G = (V, E)$ will denote a spherical *st*-graph with source $s$ and sink $t$, vertices $V$ and edges $E$, unless otherwise stated.

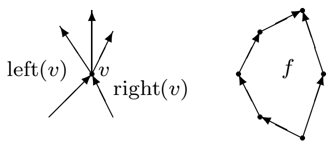


Figure 2. A vertex and a face in a spherical *st*-graph

Often, $n$ will denote the size of the problem, i.e. the number of edges in the graph. For brevity, we will sometimes use the notation $u \prec v$ if there is a path from $u$ to $v$. We will write $u \parallel v$ if neither $u \prec v$ nor $v \prec u$.

**2.2. Dynamic Transitive Closure.** We consider the *Dynamic Transitive Closure Problem* for spherical *st*-graphs. Namely, we present a data structure that handles the following operations (for clarity, we have spelt out the embedding-preserving restrictions on the update operations):

**Insert**$(u, v)$**:** Insert an edge from vertex $u$ to vertex $v$ if they are on the same face and the new edge does not induce a directed cycle,

**Delete**$(u, v)$**:** Delete the edge from $u$ and $v$ provided $\deg^+(u) \geq 2$ and $\deg^-(v) \geq 2$,

**Query**$(u, v)$**:** 'Is there is a path from $u$ to $v$?'

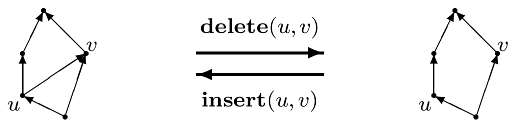


Figure 3. Updates

Alternatively, we could also allow all possible insertion and deletion operations and let the data structure decide which updates violate the restrictions. To this end, we could use the planarity testing data structure of Tamassia [16] to decide if $u$ and $v$ are on the same face. The acyclicity condition is of course easily checked using our own data structure: Edge $(u, v)$ induces a cycle if and only if there is a path from $v$ to $u$. The restriction on the deletion operation is easily checked by maintaining the in- and outdegree with each vertex.

**2.3. Lower bound.** Our update operations are sufficiently versatile to admit a lower bound proof for the problem. The model is the *cell probe model* with logarithmic word size [20]. Fredman and Saks give a lower bound of $\Omega(\log n/\log\log n)$ on the amortised complexity of the *Dynamic Parity Prefix Problem*: Given a vector $x_1, \ldots, x_n$ of bits, maintain a data structure that is able to react to the following operations for all $j = 1, \ldots, n$:

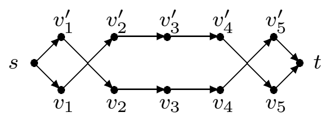
**Flip($j$):** Negate the value of $x_j$.

**Query($j$):** Return $\bigoplus_{i=1}^{j} x_i$, the parity of the first $j$ elements.

We reduce this problem to the Dynamic Transitive Closure Problem introduced above; similar reductions have recently also been used by Miltersen *et al.* [10] and Rauch [13] for other graph problems. We give the full proof to gain more familiarity with the topology and the update operations. Note that there is no obvious way to transform the proof to the case of *plane st*-graphs or to the update repertory of [18].

**Theorem 2.1.** *The Dynamic Transitive Closure Problem on spherical st-graphs requires amortised time $\Omega(\log n/\log\log n)$ in the cell probe model with logarithmic word size.*

*Proof.* Let $x_1, \ldots, x_n$ be an instance of the Dynamic Parity Prefix Problem. Construct the planar $st$-graph $G = (V, E)$ as follows: The vertex set $V$ contains source $s$ and sink $t$ as well as $2n+2$ vertices $v_1, \ldots, v_{n+1}$ and $v'_1, \ldots, v'_{n+1}$. Intuitively, $v_i$ and $v'_i$ correspond to variable $x_i$. The edge set $E$ is constructed from the values of the variables: If $x_i$ is false then $E$ includes the edges $(v_i, v_{i+1})$ and $(v'_i, v'_{i+1})$, else it contains $(v_i, v'_{i+1})$ and $(v'_i, v_{i+1})$. The figure below gives an example for $(x_1, \ldots, x_4) = (1, 0, 0, 1)$.



We embed the crossing edges of $G$ by mapping one of them to the opposite side of the sphere. It is not hard to see that we can simulate every update operation to the vector $x_1, \ldots, x_n$ using a constant number of insert and delete operations on $G$ without violating its topology. For the query operation, observe that

$$\bigoplus_{i=1}^{j} x_i = 1 \quad \leftrightarrow \quad v_1 \prec v'_{j+1}, \qquad j = 1, \ldots, n.$$

Thus a lower bound on the Prefix Problem implies a lower bound on the Transitive Closure Problem. □

**2.4. Related work.** Italiano *et al.* [7] present a dynamic reachability algorithm for *series parallel* digraphs; apart from these and the class studied in the present paper, no other class of digraphs is known to the author that allows fully dynamic reachability algorithms within polylogarithmic time bounds. The only other nontrivial upper bound is the already cited $O(n^{2/3} \log n)$ for plane graphs from [15]. It is easy to see that the $\Omega(\log n / \log \log n)$ lower bound from this paper applies to that problem; no better lower bound is known.

Other dynamic problems on planar *st*-graphs are studied in [1] and [16]. Reference [17] contains pointers to a vast number of applications of these graphs within visibility representations, graph drawing and embedding, motion planning, computational geometry, lattice theory, and VLSI design.

## 3. Properties of Spherical *st*-Graphs

**3.1. Two trees.** We employ an idea used in many polylog-time dynamic graph algorithms: Decompose the graph into a number of trees such that all the necessary information can also be derived from the trees.

**Definition 3.1.** The tree $S_G$ is the subgraph of $G$ constructed by removing all edges that are not the leftmost *incoming* edge of any vertex. Similarly, the tree $T_G$ is constructed by removing all edges that are not the leftmost *outgoing* edge to any vertex. When the graph is fixed, we will drop the subscripts on $S$ and $T$.

See Figure 4, which shows $S$ and $T$ for the graph from Figure 1. Observe the following facts:

(1) $S$ and $T$ are indeed trees,

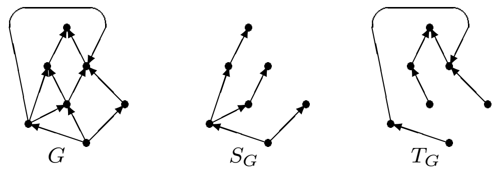


Figure 4. A graph $G$ with corresponding trees $S_G$ and $T_G$.

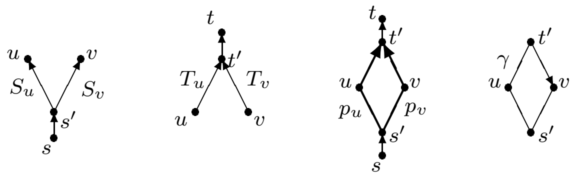
(2) $S$ is divergent and rooted at $s$, while $T$ is convergent and rooted at $t$ (hence the names),

(3) no subpath of $T$ can ever leave another path to the right, and no subpath of $S$ can ever enter another path from the right.

Let us emphasise the last innocent-looking and obvious item, since we will use it quite often:

**Fact 3.1.** *If a subpath of $T$ crosses a subpath of $S$, it does so from right to left.*

We need some notation. For vertex $v \in V$ we let $S_v$ denote the unique path from $s$ to $v$ in $S$ and let $T_v$ denote the unique path from $v$ to $t$ in $T$. For $u, v \in V$ we let $s'$ denote the last vertex that is on both $S_v$ and $S_u$. Let $t'$ denote the first vertex that is on both $T_v$ and $T_u$. The path $p_u$ is the subpath of the concatenation of $S_u$ and $T_u$ from $s'$ to $t'$. Symmetrically, $p_v$ is the sub-path of the concatenation of $S_v$ and $T_v$ from $s'$ to $t'$. The figure below depicts this construction.



Whenever it seems convenient, we will also refer to the two paths as $p_l$ and $p_r$, such that $p_l$ is the path leaving $s'$ to the left and $p_r$ is the other path.

We will boldly confuse the edges of $G$ with their embedding to alleviate notation. Namely, we introduce the curve $\gamma$ which is the concatenation of (the embeddings of) $p_l$ and $p_r$. The orientation of $\gamma$ will be such that it agrees with the direction of $p_l$ and the reversed direction of

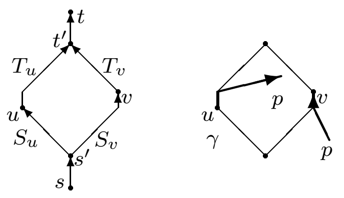
$p_r$. Recall that a curve is *closed* if its endpoints coincide, it is *simple* if it does not intersect itself except at its endpoints. Note that $\gamma$ is closed and not necessarily simple.

**3.2. Reachability in Spherical *st*-graphs.** The next lemma is the *crux* of our algorithm. It captures the following fact about reachability in spherical *st*-graphs: To get from vertex $u$ to vertex $v$ one can always choose a path whose first half stays in $T$ and whose last half stays in $S$.

**Lemma 3.1.** *Let $\leq_S$ and $\leq_T$ denote the predecessor relation in $S$ and $T$, respectively. Then*

$$u \prec v \quad \leftrightarrow \quad \exists w \in V\colon u \leq_T w \wedge w \leq_S v.$$

*Proof.* Assume for contradiction that there is a path $p$ from $u$ to $v$ even though $S_v$ and $T_u$ are vertex-disjoint.



Note that $S_u$ crosses neither $S_v$ (else $S$ would not be a tree) nor $T_u$ (else $G$ would have a cycle). Similarly, $T_v$ crosses neither $T_u$ nor $S_v$ nor $S_u$ (the latter would form a cycle with $p$). So we have the situation depicted to the left in the above figure modulo the symmetrical case where $u$ appears to the right of $v$.

Without loss of generality, we can split $p$ into three parts $p_u$, $p'$ and $p_v$, such that $p_u$ is a (possibly empty) sub-path of $T_u$, $p_v$ is a (possibly empty) sub-path of $S_v$ and $p'$ (which contains at least one vertex) has no vertices in common with either $T_u$ or $S_v$.

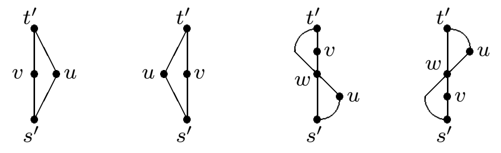
Note that $p'$ leaves $T_u$ before $t'$ (else there would be a cycle in $G$) and does so to the right by Fact 3.1. Similarly, $p'$ enters $S_v$ after $s'$ and does so from the right. The right part of the figure above conveys the absurdity of this: Part of $p'$ is in the interior of $\gamma$, while another part is in the exterior. Hence $p'$ must cross $\gamma$ somewhere, but cannot by construction. $\square$

**3.3. The plane case.** To see some of the present machinery in motion and to get our hand dirty before we study the full problem, let us derive an algorithm for the case of *plane st*-graph.

We must handle the existential quantifier of the last lemma without searching all of $V$. We will show that the existence of $w$ 'between $u$ and $v$' can be read off the edges around $s'$ and $t'$.

**Lemma 3.2.** *In a plane st-graph, the reachability information between $u$ and $v$ is uniquely determined by the appearance of $p_u$ and $p_v$ around $s'$ and $t'$.*

*Proof.* The proof is a case analysis on the behaviour of $p_u$ and $p_v$ between $s'$ and $t'$. We shall see that there are only four cases, depicted below.



First note that if $s' = u$ then there is a path from $u$ to $v$ and we are done. Similarly, the cases $s' = v$, $t' = u$, and $t' = v$ are trivial.

Assume first that $p_u$ leaves $s'$ to the right of $p_v$. There are two cases: Either $p_u$ stays to the right of $p_v$ (until the two paths finally meet at $t'$) or it does not. In the former case (the leftmost example in the figure), there cannot be a path from $v$ to $u$ by Lemma 3.1.

In the latter case, $p_u$ must cross $p_v$ at some point to get to the other side. It cannot enter it anywhere except between $s'$ and $t'$, by acyclicity of $G$ and construction of $t'$, hence it enters at some vertex $w \neq t'$. Since $w$ is on both $p_u$ and $p_v$, one of the following must hold: (i) $u \prec w$ and $v \prec w$, (ii) $u \prec w$ and $w \prec v$, (iii) $w \prec u$ and $v \prec w$, or (iv) $w \prec u$ and $w \prec v$. The reader should check that all possibilities but the second contradict Fact 3.1 or induce an undirected cycle in $S$ or $T$. Hence, by transitivity of $\prec$, we have $u \prec v$. Similar arguments show that once $p_u$ has reached the left side of $p$, it cannot come back; hence it enters $t'$ left of $p_v$. This is the third example in the figure above.

We can repeat the analysis for the case where $p_u$ leaves $s'$ left of $p_v$ (depicted by the second and fourth examples), to complete Table 1.

Put succinctly, $u$ and $v$ are connected if and only if $p_u$ and $p_v$ 'switch sides.' $\square$

| $p_u$ leaves $s'$ right of $p_v$ | y | y | n | n |
|---|---|---|---|---|
| $p_u$ enters $t'$ right of $p_v$ | y | n | y | n |
| Reachability | $u \parallel v$ | $u \prec v$ | $v \prec u$ | $u \parallel v$ |

Table 1. Reachability in the plane case

3.3.1. *Data Structures.* We maintain the following information:

(1) With every vertex $v$: Two sequences of the incoming and outgoing edges of $v$, respectively, ordered according to the cyclic ordering around $v$ (see the remarks after Definition 2.1). We can used balanced search trees for this.
(2) The trees $S$ and $T$ using the *dynamic tree* data structure of Sleator and Tarjan [14].

3.3.2. *Updates.* After each insertion or deletion we must reorganise our data structures. An edge can be inserted into or deleted from the edge list around a vertex in time $O(\log n)$; maintaining the two dynamic trees is a standard technique.

3.3.3. *Queries.* Evert $u$ and $v$ in $S$ to find their nearest common ancestor $s'$, see [14]. Evert $u$ and $v$ in $T$ to find their nearest common ancestor $t'$. From the edge lists around $s'$ and $t'$ we see which of $p_u$ and $p_v$ appears rightmost. By Table 1, this yields the reachability information.

In summary, we have re-proved the following theorem due to Tamassia and Preparata [17], using a different characterisation.

**Theorem 3.1.** *The Dynamic Transitive Closure Problem for plane st-graphs can be solved in time* $O(\log n)$*, where* $n$ *denotes the number of edges. The data structure uses linear space and can be initialised in linear time.*

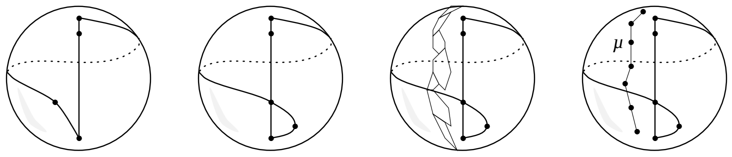


Figure 5. The sphere: problems (left) and remedy (right).

**3.4. Additional concepts for spherical graphs.** Let us reiterate the gist of the last section:

(1) If $u$ and $v$ are connected, then $p_u$ and $p_v$ intersect,
(2) If $p_u$ and $p_v$ intersect, then they 'switch sides,' i.e., they appear around $s'$ in another order than they do around $t'$.

The first item still holds in the spherical case. The second does not. The first two figures above show why the sphere is much more difficult than the plane: Paths can wrap around; the reader can easily check that both examples contradict Table 1. The remedy is to keep track of the globe-trotting of $\gamma$ by maintaining a chain of faces between the poles, as indicated in the third figure; it is helpful to view this chain of faces as a path $\mu$ in the dual of the graph. The chain is called *the meridian* and formally introduced in Section 4. First, we introduce some additional concepts to be able to formalise what we just sketched.

**Definition 3.2.** A *region* is a maximal topologically connected subset in the complement of $\gamma$. A curve is *proper* if it intersects $\gamma$ only at points where $\gamma$ does not intersect itself. We define the function Ind that maps points to integers as follows: For $x$ in a region the *index* $\operatorname{Ind}(x)$ is the minimum number of intersections between $\gamma$ and $\mu$ over all proper curves $\mu$ from $s$ to $x$. Note that Ind is constant on every region, vanishes on the region of $s$, and in the plane case, also on the region of $t$.

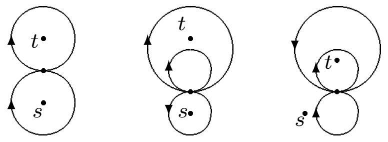
For $\operatorname{Ind}(t) > 0$, we define the *orientation of* $t$ as follows: Let $x$ be a point in a region incident to the region of $t$ such that $\operatorname{Ind}(x) = \operatorname{Ind}(t) - 1$. Let $\mu$ be a proper curve from $x$ to $t$ that crosses $\gamma$ only once. Then the orientation of $t$ is *positive* if $\mu$ crosses $\gamma$ from left to right, and *negative* otherwise.

Perhaps more intuitively, the orientation of $t$ is the direction of the closed curve that separates the region of $t$ from its neighbouring region

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_r$ right of $p_l$ at $t'$ | y | n | n | y | y | n | n | n | y | y | n | n |
| $p_u$ right of $p_v$ at $s'$ | – | – | y | n | y | n | y | n | y | n | y | n |
| Index of $t$ | 0 | 1 | 0 | 1+ | 1+ | 2+ | 1+ | 0 | 1+ | 1+ | 2+ | 1+ |
| Orientation of $t$ | – | $\circlearrowright$ | – | $\circlearrowright$ | $\circlearrowleft$ | $\circlearrowright$ | $\circlearrowleft$ | – | $\circlearrowright$ | $\circlearrowleft$ | $\circlearrowright$ | $\circlearrowleft$ |
| Reachability | $u \parallel v$ | | $u \prec v$ | | | | | $v \prec u$ | | | | |

Table 2. Reachability in the spherical case.

with lower index. If this curve is oriented clockwise, the orientation of $t$ is positive. The figure below shows some examples where $\mathrm{Ind}(t) = 2$ and the orientation of $t$ is positive.



The next lemma, which is the spherical analogue to Lemma 3.2, states that the concepts we introduced suffice to characterise the reachability information.

**Lemma 3.3.** *The reachability information between $u$ and $v$ is uniquely determined by (i) the index of $t$, (ii) the orientation of $t$, and (iii) the appearance of $p_u$ and $p_v$ around $s'$ and $t'$.*

As Table 1 did in the plane case, Table 2 shows the precise connection (dashes denote arbitrary or undefined entries). Note that indeed the reachability information is uniquely determined by the information above the rule. As one would expect, the case analysis is considerably more complicated than for the plane case. Figure 6 shows the possible behaviour of $p_u$ and $p_v$ and can be used as a graphical proof of the lemma. The reader should check that all cases are consistent with Table 2.

Obviously, the sceptical reader should have no reason to believe that the examples in Figure 6 exhaust all possible cases. Unfortunately, the formal proof is somewhat tedious and unintuitive. We confine it to the next section. At first reading the reader may simply choose to accept the result and continue to Section 4.

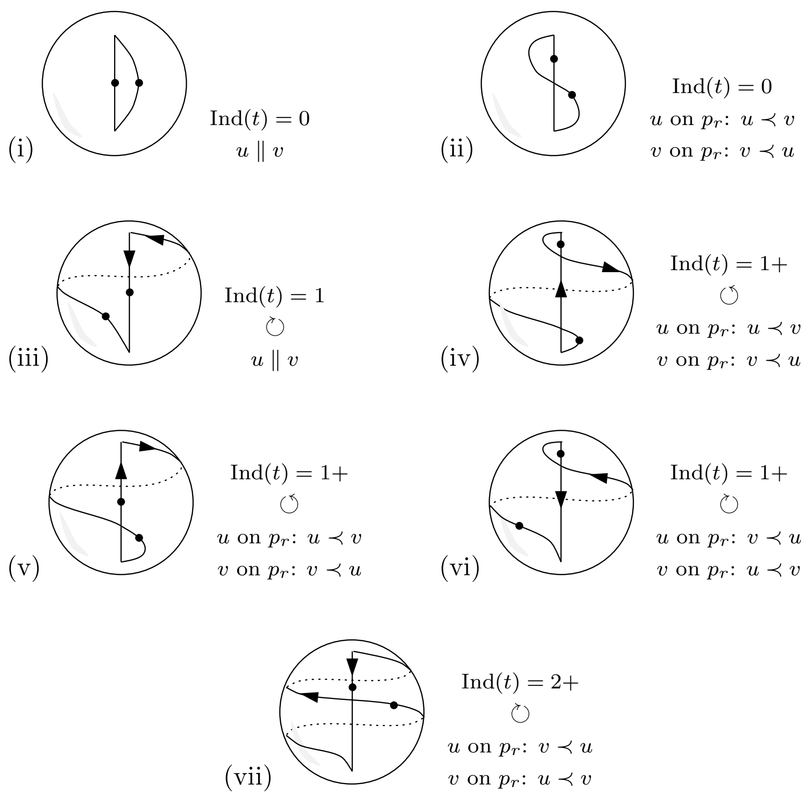
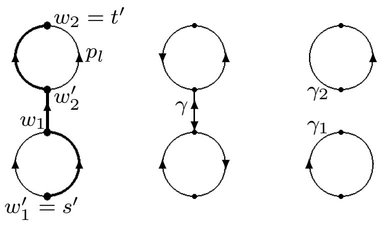

(i)
$\mathrm{Ind}(t) = 0$
$u \parallel v$
(ii)
$\mathrm{Ind}(t) = 0$
$u$ on $p_r$: $u \prec v$
$v$ on $p_r$: $v \prec u$
(iii)
$\mathrm{Ind}(t) = 1$
↻
$u \parallel v$
(iv)
$\mathrm{Ind}(t) = 1+$
↺
$u$ on $p_r$: $u \prec v$
$v$ on $p_r$: $v \prec u$
(v)
$\mathrm{Ind}(t) = 1+$
↺
$u$ on $p_r$: $u \prec v$
$v$ on $p_r$: $v \prec u$
(vi)
$\mathrm{Ind}(t) = 1+$
↻
$u$ on $p_r$: $v \prec u$
$v$ on $p_r$: $u \prec v$
(vii)
$\mathrm{Ind}(t) = 2+$
↻
$u$ on $p_r$: $v \prec u$
$v$ on $p_r$: $u \prec v$

Figure 6. Canonical examples of the behaviour of $p_u$ and $p_r$ on the sphere. The two topmost cases appear also in the plane, while the five other cases exploit the possibility to travel around the sphere. In all cases we give the index of $t$, and, if the latter is nonzero, the orientation of the region of $t$. In these cases, the orientation of $\gamma$ is depicted by arrows. Fat dots indicate the possible positions of $u$ and $v$. Examples (iii) to (vii) each represent an infinite number of cases in which the paths cross any number of times; in all those cases, the orientation and the reachability information is the same.

**3.5. Towards a proof of Lemma 3.3.** We have chosen to split the proof into a series of (easy) lemmas. We begin with some concepts that give a more fine-grained view of $\gamma$. Assume that $p_r$ enters $p_l$ at vertices $w_1, \dots, w_k$, with $w_k = t'$, and leaves it at vertices $w'_1, \dots, w'_k$, with $w'_1 = s'$ (the ordering agrees with the topological ordering of the vertices). Then for $i = 1, \dots, k$, the curve $\gamma_i$ consists of the subpath of $p_l$ from $w'_i$ to $w_i$ and the (reversed) subpath of $p_r$ from $w_i$ to $w'_i$.



The figure above gives an example. Note that all $\gamma_i$ are subcurves of $\gamma$. On the other hand, not all of $\gamma$ is necessarily part of some $\gamma_i$. The following lemma follows easily from the construction.

**Lemma 3.4.** *Let $\gamma_1, \dots, \gamma_k$ be a collection of curves as above. Then*

(1) *every $\gamma_i$ is a simple closed curve,*
(2) *for $i \neq j$, the curves $\gamma_i$ and $\gamma_j$ are disjoint except for the case $j = i + 1$, where they may intersect at $w_i = w'_{i+1}$.*

*Proof.* Clearly, every $\gamma_i$ is closed. Moreover, it consists of a part from $p_r$ that cannot intersect itself (else there would be a cycle in $G$) and does not intersect $p_l$ before $w_i$ by construction; likewise, $p_l$ does not intersect itself, so $\gamma_i$ is simple. The same argument shows that two curves cannot intersect except as stated. □

Let us introduce a shorthand notation that captures the way $p_l$ and $p_r$ cross. The *entrance sequence* $E$ of $p_r$ and $p_l$ is a string of $k$ letters from $\{\mathrm{R}, \mathrm{L}\}$ defined according to how the two paths cross. There is a letter in the sequence for every $w_i$, and that letter is an R if $p_r$ enters $p_l$ from the *right* at $w_i$, and an L if it enters from the *left*. Note that $p_r$ enters $p_l$ at least once, namely at $t'$, so the entrance sequence is nonempty. The entrance sequence for the example above is RL. Let us show that all letters but possibly the last are the same.

**Lemma 3.5.** $E \in \mathrm{R}^+ \cup \mathrm{L}^+\mathrm{R} \cup \mathrm{R}^+\mathrm{L} \cup \mathrm{L}^+$.

*Proof.* Assume without loss of generality that $u$ is on $p_l$. Assume first that LR is a substring but not a suffix of the sequence, so $p_r$ crosses $p_l$ first from left to right (say, at vertex $w_i$) and then from right to left (at vertex $w_{i+1}$). From Fact 3.1 we learn that $u \prec w_i$ and $w_{i+1} \prec u$ which contradicts the ordering of the $w_i$. The case RL is analogous. □

The next lemma is obvious, now that we have split $\gamma$ into simple curves. We leave the proof to the reader.

**Lemma 3.6.** *Let $E$ denote an entrance sequence of length $k$. Then the $k$ curves $\gamma_1, \dots, \gamma_k$ satisfy:*

(1) *$\gamma_1$ separates $s$ from $t$ iff $E$ begins with an* L*,*
(2) *$\gamma_i$ separates $s$ from $t$ for $i = 2, \dots, k-1$,*
(3) *$\gamma_k$ separates $s$ from $t$ iff* LL *or* RR *is a suffix of $E$ or $E =$* L*.*

*Moreover, $\gamma_i$ is oriented clockwise iff $E_i =$* L*.*

**Lemma 3.7.** *There is only one curve if and only if $u \parallel v$. Otherwise, $u \prec v$ if and only if $E_1 =$* R *and $p_u = p_r$ or $E_1 =$* L *and $p_v = p_r$.*

*Proof.* If $u$ and $v$ are connected then $\gamma$ is non-simple from Lemma 3.1, so the first part of the statement holds. Assume $E_1 = \mathrm{R}$ and $p_u = p_r$, so $p_u$ crosses $p_v$ from right to left. From Fact 3.1 we see that $u \prec w_1$ and $w_1 \prec v$ and are done by transitivity. The other cases are symmetrical. □

*Proof of Lemma* 3.3. The proof is an easy but slightly tedious case analysis on the four different types of entrance sequences. The last two lemmas yield the number of cycles that separate $s$ from $t$, their orientation and the reachability information. By inspection, all cases are seen to be consistent with Table 2. □

## 4. Algorithm for Sequences of Updates

**4.1. The Meridian.** We use the results of the last section to construct an algorithm that performs well in the amortised sense, i.e., a sequence of $m$ updates and queries takes time $\mathrm{O}(m \log n)$.

As mentioned in the last section, one of the main ideas behind our algorithm is to maintain a chain of faces between the poles, which we will now define.

**Definition 4.1.** A *meridian* $(F^0, E^0)$ consists of a sequence of *meridian faces* $F^0 = \langle f_1, \ldots, f_m \rangle$ and *meridian edges* $E^0 = \langle e_1, \ldots, e_{m-1} \rangle$ such that

(1) for $i = 1, \ldots, m-1$, edge $e_i$ is on the boundaries of $f_i$ and $f_{i+1}$,
(2) $f_i \neq f_j$ for $i \neq j$ (this implies $e_i \neq e_j$).

Moreover, $f_1 = \text{left}(s)$ and $f_m = \text{left}(t)$.

It is easy to see that the meridian corresponds to a *proper* curve $\mu$ in the sense of Definition 3.2 by viewing the meridian as a path in the dual $G^*$ of $G$ and overlaying the embeddings of $G^*$ and $G$ in a straightforward way. We only have to observe that a path in $G^*$ can never contain a point that embeds a vertex from $G$. Recall the right half of Figure 5 on page 13 for an example.

**4.2. How to count wrap-arounds.** For curves $\alpha$ and $\beta$ we let $\phi_r(\alpha, \beta)$ denote the number of times $\alpha$ crosses $\beta$ from right to left. Symmetrically, $\phi_l(\alpha, \beta)$ denotes the number of times $\alpha$ crosses $\beta$ from left to right.

Note that $\phi_l$ and $\phi_r$ have the nice property that if we decompose $\alpha$ into proper curves $\alpha_1, \ldots, \alpha_k$ then we have, e.g.,

$$\phi_l(\alpha, \beta) = \sum_{i=1}^{k} \phi_l(\alpha_i, \beta). \tag{4.1}$$

If $\alpha$ is a closed curve and $\beta$ is a proper curve (with respect to $\alpha$) whose endpoints are on the same region (with respect to $\alpha$), then $\beta$ must leave the region bounded by $\alpha$ as often as it enters it, so

$$\phi_l(\alpha, \beta) - \phi_r(\alpha, \beta) = \phi_l(\beta, \alpha) - \phi_r(\beta, \alpha) = 0.$$

These properties are exploited in the proof of the following lemma.

**Lemma 4.1.** *The index and the orientation of $t$ are given by the absolute value and the sign of*

$$\phi_r(\mu, p_l) + \phi_l(\mu, p_r) - \phi_l(\mu, p_l) - \phi_r(\mu, p_r),$$

*respectively.*

*Proof.* Observe that the meridian connects a point in the region of $s$, namely left$(s)$, to a point in the region of $t$, namely left$(t)$. Let $\gamma_1, \ldots, \gamma_k$, with $k = \mathrm{Ind}(t)$, denote the simple closed subcurves of $\gamma$ that separate $s$ from $t$. It is an easy corollary to lemmas 3.5 and 3.6 that the curves have the same orientation. Note that the meridian must cross all $k$ curves at least once, but may take a detour: It can go back across a previously crossed curve and return later. Thus the index of $t$ is given by

$$\mathrm{Ind}(t) = \Big| \sum_{i=1}^{k} \phi_r(\mu, \gamma_i) - \phi_l(\mu, \gamma_i) \Big|.$$

We can split each $\gamma_i$ into appropriately indexed subpaths $p_l^i$ and $p_r^i$ of $p_l$ and $p_r$ (and remember to reverse the direction of the latter) to derive

$$\mathrm{Ind}(t) = \Big| \sum_{i=1}^{k} \phi_r(\mu, p_l^i) + \phi_l(\mu, p_r^i) - \phi_l(\mu, p_l^i) - \phi_r(\mu, p_r^i) \Big|.$$

All other subpaths of $p_l$ and $p_r$ form a number of closed curves that do not influence $\phi(\mu, \cdot)$, so we can extend the above sum to include all of $p_l$ and $p_r$ without changing the result. This proves the first statement.

For the second statement, observe that the orientation of $t$ is positive if and only if all $\gamma_i$ are oriented clockwise. In that case, the value of

$$\sum_{i=1}^{k} \phi_r(\mu, \gamma_i) - \phi_l(\mu, \gamma_i)$$

is negative, else it is positive. Indeed, the expression evaluates to either $\mathrm{Ind}(t)$ or $-\mathrm{Ind}(t)$, depending on the orientation of $t$. □

**4.3. Data Structure.** We extend the data structure of Section 3.3.1, keeping the sequences of outgoing and incoming edges around every vertex and the dynamic trees for $S$ and $T$. The extensions are:

(1) We maintain the sequences of meridian faces $F^0$ and edges $E^0$ under insertion and deletion of subsequences, e.g., using balanced trees.
(2) With every edge $e$ that is in either $S$ or $T$, we store

$$\phi_r(\mu, e) = \begin{cases} 1, & \text{if } e = e_i \text{ for some } e_i \in E^0 \text{ and right}(e) = f_i, \\ 0, & \text{otherwise,} \end{cases}$$

which tells us if $e$ is crossed by the meridian from right to left. Symmetrically, we store $\phi_l(\mu, e)$, which can be derived analogously. Using (4.1) above, we can now in time $\mathrm{O}(\log |E|)$ calculate the value of $\phi_r(\mu, p)$ and $\phi_l(\mu, p)$ for every *dynamic path* $p$ of $S$ or $T$; see [14] for the details and terminology.

(3) With every face, we keep a topologically ordered sequence of the edges on the two paths that bound the face.

4.3.1. *Queries.* For the query operation, we again evert $u$ and $v$ in $S$ and $T$ to find their order around $s'$ and $t'$. Using Lemma 4.1 and the data structure above, we find the index and orientation of $t$. Finally, we refer to Table 2 for the answer.

4.3.2. *Insertions.* Consider the case where a new edge $e$ is inserted into face $f$, splitting it into $f'$ and $f''$. The edge lists around $f'$ and $f''$ are easily derived from the edge lists around $f$. The meridian is unaffected if $f \notin F^0$. Otherwise, one or both of $f'$ and $f''$ may become part of the updated meridian, depending on where the meridian edges appear around $f$ (we use the edge list around $f$ to decide which case we are in). For example, if there is a meridian edge on both $f'$ and $f''$, they both become part of the meridian and $e$ becomes a new meridian edge. In any case, there are only a constant number of updates to the meridian lists.

A straightforward analysis shows that all operations can be performed in logarithmic time, including the updates to the values of $\phi_l$ and $\phi_r$ stored in $S$ and $T$.

4.3.3. *Deletions.* Consider the case where deletion of the edge $e$ between faces $f'$ and $f''$ creates a new face $f$. Creating the edge list around $f$ is handled as above.

In contrast, the meridian may change drastically. The change occurs when both $f'$ and $f''$ are meridian faces: We cannot just merge them into one, as that would violate the second condition of Definition 4.1 — put more graphically, the meridian curve $\mu$ would no longer be simple.

To remedy this, we must remove everything between $f'$ and $f''$ from the meridian, as shown in Figure 7. Even though the data structure for the meridian face and edge lists can be updated in logarithmic time, the values $\phi_l(\mu, e)$ and $\phi_r(\mu, e)$ at every removed meridian edge $e$ also have to be changed, which takes time $\mathrm{O}(n \log n)$ in the worst case. However,

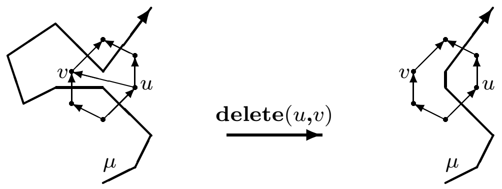


Figure 7. Deletion of an edge that separates two meridian faces

an easy amortisation argument (store a credit with each meridian edge) shows that a sequence of $m$ updates and queries can be executed in time $O(m \log n)$.

In summary, we have the following theorem:

**Theorem 4.1.** *The Dynamic Transitive Closure Problem on spherical st-graphs can be solved in amortised time* $O(\log n)$*, where* $n$ *denotes the number of edges. The data structure uses linear space and can be initialised in linear time.*

## 5. Worst case time bounds

**5.1. Sketch of technique.** We will now remove the amortisation, a task that involves some rather tedious arguments. We start with a rough sketch: Obviously, the major problem is that we do not have time to remove the meridian cycles arising from a delete operation. However, it is not very hard to believe that such meridian cycles can be shown not to influence the proof of Lemma 4.1: In a nutshell, whenever a path crosses a such a merdian *cycle*, it most re-cross the same cycle later in the other direction (meridian cycles cannot seperate $s$ from $t$). Hence we choose to let sleeping dogs lie. We do *not* remove the meridian cycles but instead just make sure that they stay cycles as the graph undergoes further changes.

The minor problem left is that this results in more and more meridian cycles as we go, so we use 'global rebuilding' [12] to construct an unpolluted data structure in the background.

Now for the details.

**5.2. False meridians.** We introduce some more meridians $(E^k, F^k)$ for $r > 0$. To distinguish them from the meridian $(E^0, F^0)$ of Definition 4.1, we from now on refer to the latter as the *prime meridian*.

**Definition 5.1.** A *false meridian* $(F^k, E^k)$ for $r > 0$ of size $l$ consists of a sequence of faces $F^k = \langle f_1^k, \ldots, f_l^k \rangle$ and $E_m^k = \langle e_1^k, \ldots, e_l^k \rangle$ such that

(1) for $i = 1, \ldots, l-1$, edge $e_i^k$ is on the boundaries of $f_i^k$ and $f_{i+1}^k$,
(2) edge $e_l$ is on the boundaries of $f_l^k$ and $f_1^k$,
(3) $f_i^k \neq f_j^k$ for $i \neq j$ (this implies $e_i^k \neq e_j^k$).

Thus the difference between a false meridian and the prime meridian is that the former is cyclic in the sense that the last face is incident to the first. Also, a false meridian need not contain left$(s)$ nor left$(t)$. The embedding of a false meridian is a closed proper curve.

Our algorithm will not be able to distinguish false meridians from the prime one. More precisely, when $\gamma$ crosses a meridian at some point, the algorithm cannot *locally* deduce whether this meridian is the prime meridian or some other. Let us argue that this does not matter.

Denote by $\mu^k$ the curves that correspond to false meridians. Since these curves are closed we can use the discussion from Section 4.2 to derive

$$\phi_r(\mu^k, \gamma) - \phi_l(\mu^k, \gamma) = 0,$$

for all $\mu^k$. Hence we can add the vanishing term

$$\sum_{k>0} \phi(\mu^k, p_l) + \phi_l(\mu^k, p_r) - \phi_l(\mu^k, p_p) - \phi_r(\mu^k, p_r),$$

where the sum is over all false meridians, to expression (4.2) without changing the result.

**5.3. Data structure.** Now that we have seen that the false meridians do not mess up our analysis, let us see that they even make life simpler.

We modify the data structure from the amortised case as follows:

(1) With every edge we store the value

$$\sum_{k\geq 0} \phi_r(\mu^k, e), \tag{5.1}$$

where the sum is over all meridians including the prime. Likewise, we store $\sum \phi_l(\mu_k, e)$.

(2) The two balanced trees for each face that maintain the two sequences of edges around the face are modified so that each internal node computes the sum of the values stored at its children. This allows us to calculate the value

$$\sum_i \sum_k \phi_r(\mu^k, e_i)$$

for each sequence of faces $\langle e_i \rangle$ that appear consecutively around the face in time logarithmic in the length of the sequence. Likewise for $\phi_l$.

Note that we do *not* maintain sequences of false meridians (but still maintain the prime meridian). The false meridians appear in the data structure only implicitly in the value from (5.1) stored at each edge. Let us very briefly sketch how to handle the updates.

5.3.1. *Insertions.* Whenever a new edge is inserted into a face that appears on some (possibly false) meridian, we have to update the value from (5.1). The modified balanced search trees with each face allows us to compute the number of meridians that enter and leave the two new faces. From these values, we can derive the value stored with the new edge consistently with some legal rearrangement of the false meridians.

5.3.2. *Deletions.* Whenever an edge deletion induces a cycle in the prime meridian, we remove that cycle from the corresponding list in the data structure as before and make the removed cycle a new false meridian. Figure 8 gives an example.

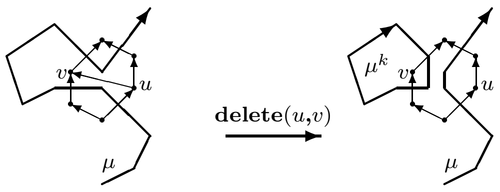


Figure 8. Edge deletion, worst case

**5.4. Global rebuilding.** We are almost finished. The only problem is that the number of false meridians is unbounded and hence the values stored with each edge may at some point become exponential.

To avoid this, we use the standard trick of *global rebuilding* (see Chapter 5 of [12]): Construct a new data structure in the background, based on only the prime meridian. After a linear number of operations, the construction has finished and we switch to this new structure (which may already have some false meridians but nevertheless cannot be too large). Now all calculation takes place using the new data structure and we refresh the old data structure in the background. This process of switching data structures is repeated *ad infinitum*. We leave the details with the reader.

**Theorem 5.1.** *The Dynamic Transitive Closure Problem on spherical st-graphs can be solved in time* $\mathrm{O}(\log n)$*. The data structure uses linear space and can be initialised in linear time.*

## 6. Conclusion

We have characterised the complexity of the dynamic transitive closure problem on planar embedded graphs with one source and one sink within a factor $\log\log n$.

Note that it is easy to extend the data structure to cope with a *report* operation that outputs a path from $u$ to $v$ if it exists in time $\mathrm{O}(\log n)+r$, where $r$ denotes the length of the path. We leave the detail to the reader.

**6.1. Open questions.** It would be aesthetically pleasing to close the gap between the upper and the lower bound. Dietz [4] has removed the $\log\log n$ factor in other dynamic problems, maybe similar techniques apply. However, the necessary overhead supposedly dwarfs the asymptotic improvement for all realistic input sizes and the result would be of theoretical interest only.

Upper bounds on the Dynamic Transitive Closure Problem in the general case are still weak. Maybe slight extensions of the class of spherical $st$-graphs can be handled by techniques similar to this papers'. For example, the class of graphs that admit an upward planar drawing could be the next target. Along another path, one could try to remove the acyclicity condition. More ambitiously, we could look for improved lower bounds for the general problem.

**6.2. Acknowledgements.** I am indebted to Gerth Stølting Brodal for several discussions that led to this work and for his insightful comments on an earlier version of this paper. I also thank Lars Arge, Peter Binderup, Peter Bro Miltersen, and Gudmund Frandsen for comments that considerably improved he presentation.

## References

1. Giuseppe Di Battista and Roberto Tamassia, *Algorithms for plane representations of acyclic digraphs*, Theoretical Computer Science **61** (1988), 175–198.
2. Robert F. Cohen and Roberto Tamassia, *Dynamic expression trees and their applications*, Proc. 1st Ann. Symp. on Discrete Algorithms (SODA), ACM-SIAM, 1990, pp. 52–61.
3. ———, *Combine and conquer: a general technique for dynamic algorithms*, Proc. 1st Ann. European Symp. on Algorithms (ESA) (Thomas Lengauer, ed.), Lecture Notes in Computer Science, vol. 726, Springer Verlag, Berlin, 1993, pp. 97–108.
4. Paul F. Dietz, *Optimal algorithms for list indexing and subset rank*, Proc. First Workshop on Algorithms and Data Structures (WADS) (F. Dehne, J.-R. Sack, and N. Santoro, eds.), Lecture Notes in Computer Science, vol. 382, Springer Verlag, Berlin, 1989, pp. 39–46.
5. David Eppstein, Giuseppe Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery Westbrook, and Moti Yung, *Maintenance of a minimum spanning forest in a dyamic planar graph*, Journal of Algorithms **13** (1992), 33–54.
6. Greg E. Frederickson, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM Journal of Computing **14** (1985), no. 4, 781–798.
7. Giuseppe F. Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni, *Dynamic data structures for series parallel digraphs*, Proc. First Workshop on Algorithms and Data Structures (WADS) (F. Dehne, J.-R. Sack, and N. Santoro, eds.), Lecture Notes in Computer Science, vol. 382, Springer Verlag, Berlin, 1989, pp. 352–373.
8. David Kelly, *On the dimension of partially ordered sets*, Discrete Mathematics **35** (1981), 135–156.
9. K. Mehlhorn, R. Sundar, and C. Uhrig, *Maintaining dynamic sequences under equality-tests in polylogarithmic time*, Proc. 4th Ann. Symp. on Discrete Algorithms (SODA), ACM-SIAM, 1994, pp. 213–222.
10. P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia, *Complexity models for incremental computation*, Theoretical Computer Science **130** (1994), 203–236.
11. Peter Bro Miltersen, *On-line reevaluation of functions*, Tech. Report DAIMI PB–380, Computer Science Department, Aarhus University, 1992.
12. Mark H. Overmars, *The design of dynamic data structures*, Lecture Notes in Computer Science, vol. 156, Springer Verlag, Berlin, 1983.
13. Monika Rauch, *Improved data structures for fully dynamic biconnectivity*, 26th Ann. Symp. on Theory of Computing (STOC), ACM, 1994, pp. 686–695.

14. Daniel D. Sleator and Robert Endre Tarjan, *A data structure for dynamic trees*, Journal of Computer and Systems Sciences **26** (1983), 362–391.
15. Sairam Subramanian, *A fully dynamic data structure for reachability in planar digraphs*, Proc. 1st Ann. European Symp. on Algorithms (ESA) (Thomas Lengauer, ed.), Lecture Notes in Computer Science, vol. 726, Springer Verlag, Berlin, 1993, pp. 372–383.
16. Roberto Tamassia, *A dynamic data structure for planar graph embedding*, Proc. 15th International Colloquium on Automata, Languages, and Programming (ICALP) (T. Lepisto and A. Salomaa, eds.), Lecture Notes in Computer Science, vol. 317, Springer Verlag, Berlin, 1988, pp. 576–590.
17. Roberto Tamassia and Franco P. Preparata, *Dynamic maintenance of planar digraphs, with applications*, Algorithmica **5** (1990), 509–527.
18. Roberto Tamassia and Ioannis G. Tollis, *Dynamic reachability in planar digraphs with one source and one sink*, Theoretical Computer Science **119** (1993), 331–343.
19. J. van Leeuwen, *Graph algorithms*, Algorithms and complexity (J. van Leeuwen, ed.), Handbook of theoretical computer science, vol. A, Elsevier, Amsterdam, 1990, pp. 525–631.
20. Andrew Chi-Chih Yao, *Should tables be sorted?*, Journal of the ACM **28** (1981), no. 3, 615–628.