Planar Reachability in Linear Space and Constant Time

Jacob Holm and Eva Rotenberg and Mikkel Thorup†

University of Copenhagen (DIKU),
jh@poplar.dk, roden@di.ku.dk, mthorup@di.ku.dk

November 24, 2014

Abstract

We show how to represent a planar digraph in linear space so that distance queries can be answered in constant time. The data structure can be constructed in linear time. This representation of reachability is thus optimal in both time and space, and has optimal construction time. The previous best solution used $O(n \log n)$ space for constant query time [Thorup FOCS’01].
1 Introduction

Representing reachability of a directed graph is a fundamental challenge. We want to represent a digraph $G = (V, E)$, $n = |V|$, $m = |E|$, so that we for any vertices $u$ and $w$ can tell if $u$ reaches $v$, that is, if there is a dipath from $u$ to $v$. There are two extreme solutions: one is to just store the graph, as is, using $O(m)$ space and answering reachability queries from scratch, e.g., using breadth-first-search, in $O(m)$ time. The other is to store a reachability matrix using $n^2$ bits and then answer reachability queries in constant time. Thorup and Zwick [16] proved that there are graphs classes such that any representation of reachability needs $\Omega(m)$ bits. Also, Pătraşcu [12] has proved that there are directed graphs with $O(n)$ edges where constant time reachability queries require $n^{1+\Omega(1)}$ space. Thus, for constant time reachability queries to a general digraph, all we know is that the worst-case space is somewhere between $\Omega(m + n^{1+\Omega(1)})$ and $n^2$ bits.

The situation is in stark contrast to the situation for undirected/symmetric graphs where we can trivially represent reachability queries on $O(n)$ space and constant time, simply by enumerating the connected components, and storing with each vertex the number of the component it belongs to. Then $u$ reaches $v$ if and only if the have the same component number.

In this paper we focus on the planar case, which feels particularly relevant when you live on a sphere. For planar digraphs it is already known that we can do much better than for general digraphs. Back in 2001, Thorup [15] presented a reachability oracle for planar digraphs using $O(n \log n)$ space for constant query time. In this paper, we present the first improvement; namely an $O(n)$ space reachability distance oracle that can answer reachability queries in constant time. This is optimal in both time and space. Our oracle is constructed in linear time.

Computational model The computational model for all upper bounds is the word RAM, modelling what we can program in a standard programming language such as C [9]. A word is a unit of space big enough to fit any vertex identifier, so a word has $w \geq \lg n$ bits. Here $\lg = \log_2$. We will only use the standard operations on words available in C and word operations take constant time. This includes indexing arrays as needed just to store a reachability matrix with constant time access. Thus, unless otherwise specified, we measure space as the number of words used and time as the number of word operations performed. In fact, our construction we will not use multiplication or division, or any other non-AC$^0$ operation.

The $\Omega(m + n^{1+\Omega(1)})$ space lower bound from [12] for general graphs is in the cell-probe model subsuming the word RAM with an arbitrary instruction set.

Other related work Before [15], the best reachability oracles for general planar digraphs were distance oracles, telling not just if $u$ reaches $w$, but if so, also the length of the shortest dipath from $u$ to $w$ [2–4]. For such planar distance oracles, the best current time-space trade-off is $\tilde{O}(n/\sqrt{\varepsilon})$ time for any $s \in [n, n^{2-\varepsilon}]$ [11].

The construction of [15] also yields approximate distance oracles for planar digraphs. With edge weights from $[N]$, $N \leq 2^w$, distance queries where answered within a factor $(1+\varepsilon)$ in $O((\log \log (Nn) + 1/\varepsilon)$ time using $O(n(\log n)(\log(Nn))/\varepsilon)$ space. These bounds have not been improved.

For the simpler case of undirected graphs, where reachability is trivial, [10] provides a more efficient $(1+\varepsilon)$-approximate distance queries for planar graphs in $O(1/\varepsilon)$ time and $O(n(\log n)/\varepsilon)$ space. In [7] it was shown that the space can be improved to linear if the query time is increased to $O((\log n)^2/\varepsilon^2)$. In [8] it was shown how to represent planar graphs with bounded weights using $O(n \log^2((\log n)/\varepsilon) \log^* (n) \log(1/\varepsilon))$ space and answering $(1+\varepsilon)$ approximate distance queries in $O((1/\varepsilon) \log(1/\varepsilon) \log(1/\varepsilon) \log^* (n) + \log \log \log n))$ time, Using $\tilde{O}$ to suppress factors of $O(\log \log n)$ and $O(\log(1/\varepsilon))$, these bounds reduce to $O(n)$ space and $O(1/\varepsilon)$ time. This improvement is similar in spirit to our improvement for reachability in planar digraphs. However, the techniques are entirely different.

There has also been work on special classes of planar digraphs. In particular, for a planar $s$-$t$-graph, where all vertices are on dipaths between $s$ and $t$, Tamassia and Tollis [13] have shown that we can represent
reachability in linear space, answering reachability queries in constant time. Also, [3][5][6] presents improved bounds for planar exact distance oracles when all the vertices are on the boundary of a small set of faces.

**Techniques** We will develop our linear space constant query time reachability oracles by considering more and more complex classes of planar digraphs. We make reductions from \(i + 1\) to \(i\) in the following list:

1. Planar \(s\)-\(t\)-graph; \(\exists (s, t)\) such that all vertices are reachable from \(s\) and all vertices may reach \(t\). [13]
2. Planar Single-source graph; \(\exists s\), such that all vertices may be reached from \(s\).
3. Planar In-Out graph; \(\exists s\) such that all vertices with out-degree 0 are reachable from \(s\).
4. Any planar graph. The reduction to planar In-Out graphs from general planar graphs is known. [15]

Note that the graph can be assumed to be acyclic. All strongly connected components may be found in linear time using a depth first search by an algorithm of Tarjan. [14]

Also note that this bound is asymptotically optimal; even to distinguish between the subclass of directed paths of length \(n\), we need \(O(n \log n)\) bits.

The most technically involved step is the reduction from single-source graph to \(s\)-\(t\)-graph. As in [15], we use separators to form a tree over the vertices of the graph. However, in [15], the alteration count: the number of directed segments in the frame that separates a child from its parent (see Section 2), needs only be a constant number. In order to obtain linear space, it is a crucial part of our construction that the alteration number, which must be even, is almost always 2. The alteration count may be 4, but this only happens when we have additional structure we can use. The low alteration count becomes very important to our data structure, as we use a level ancestor -like algorithm to calculate fastly the best \(\leq 4\) ”projections” of some vertex \(v\) to one of its ancestral components. Each component is an \(s\)-\(t\)-graph, and \(v\) can be reached by some \(x\) in the ancestral component if and only if \(x\) can reach at least one of the ”projections”.

## 2 Preliminaries

For a vertex \(v\) at depth \(d\) in a rooted forest \(T\) and an integer \(0 \leq i \leq d\), the \(i\)’th level ancestor of \(v\) in \(T\), denoted \(\text{la}(T, v, i)\), is the ancestor to \(v\) in \(T\) at depth \(i\).

We say a graph is plane, if it is embedded in the plane, and denote by \(\pi_v\) the permutation of edges around \(v\). Given a plane graph, \((G, \pi)\), we may introduce corners to describe the incidence of a vertex to a face. A vertex of degree \(n\) has \(n\) corners, where if \(\pi_v((v, u)) = (v, w)\), and the face \(f\) is incident to \((v, u)\) and \((v, w)\), then there is a corner of \(f\) incident to \(v\) between \((v, u)\) and \((v, w)\). We denote by \(V[X], E[X], \text{cor}[X]\) the vertices, edges, and corners of some (not necessarily induced) subgraph. Given a subgraph \(H\) of a planar embedded graph \(G\), the faces of \(H\) define superfaces of those of \(G\), and the faces of \(G\) are subfaces of those of \(H\). Similarly for corners. Note that the faces of \(H\) correspond to the connected components of \(G^* \setminus H\). The super-corners incident to \(v\) correspond to a set of consecutive corners in the ordering around \(v\).

In an oriented graph, we may consider the boundary of a face in some subgraph, \(H\). A corner of a face \(f\) of \(H\) is a target for \(f\) if it lies between ingoing edges \((u, v)\) and \((w, v)\), and source if it lies between outgoing edges \((v, u)\) and \((v, w)\). We say the face boundary has alteration number \(2a\) if it has a source and a target corners. When a face boundary has alteration number \(2a\), we say it consists of \(2a\) disegments (directed segments), associated with the directed paths from source to target. We associate to each disegment also the total ordering stemming from reachability of vertices on the path via the path, and by convention we set \(\text{succ}(t, S) = \bot\) for a target vertex \(t\) on the disegment, and similarly, the predecessor of the source is \(\bot\). With a clockwise turning disegment we also associate all corners to the right side of the path, and with a counter-clockwise, all corners to the left side of the path. Given a connected planar graph with a spanning tree \(T\), the edges \(T^* := E \setminus T\) form a spanning tree for the dual graph. We call this a tree-cotree decomposition of the graph, referring to \(T\) and \(T^*\) as tree and cotree.
When $u$ can reach $v$ we write $u \rightsquigarrow v$. An s-t-graph is a graph with special vertices $s, t$ such that $s \rightsquigarrow v$ and $v \rightsquigarrow t$ for all vertices $v$. We say a graph is a truncated s-t-graph if it is possible to add vertices $s, t$ to obtain an s-t-graph, without violating the embedding. In an s-t-graph, all faces has alteration number 2.

3 Planar single-source digraph

Given a global source vertex $s$ for the planar digraph, we wish to make a data structure for reachability queries. We do this by reduction to the s-t-case. A tree-like structure with truncated s-t-graphs as nodes is obtained by recursively choosing a face $f$ wisely, and then letting vertices that can reach vertices on $f$ belong to this node, and let all other vertices belong to the descendents of this node. As we shall see in Section 3.1 this can be done in such a way that we obtain logarithmic height and such that the border between a node and its ancestors is a cycle of alteration number at most 4. We call this the frame of the node.

To use the tree-like decomposition to answer queries, we always choose the truncated s-t-graph maximally, such that once a path crosses a frame, it does not exit the frame again. Thus, for $u$ to reach $v$, $u$ has to lie in a component which is ancestral to that of $v$, and since the alteration number of any frame between those two component is at most 4, the path could always be chosen to use one of the at most 4 different "best" vertices for reaching $v$ on that frame. Thus, the idea is to do something inspired by level ancestry to find those "best" vertices in $u$'s component. $s$ We handle the case of frames with alteration number 2 in Section 3.2. Frames with alteration number 4 are similar but more involved, and the details are found in Appendix 3.3.

Definition 3.1. Given a graph $G = (V, E)$, a subgraph $G' = (V', E')$ is backward closed if $\forall (u, v) \in E : v \in V' \implies (u, v) \in E'$.

Definition 3.2. The backward closure of a face $f$, denoted $bc(f)$ is the unique smallest backward closed graph that contains all the vertices incident to $f$.

Definition 3.3. Let $G = (V, E)$ be an acyclic single-source plane digraph, and let $G^* = (V^*, E^*)$ be its dual. An s-t-decomposition of $G$ is a rooted tree where each node $x$ is associated with a face $f_x \in V^*$ and subgraphs $G_x^* \subseteq G^*$ and $C_x \subseteq S_x \subseteq G$ such that:

- $f_x$ is unique ($f_x \neq f_y$ for $x \neq y$).
- $S_x$ is $bc(f_x)$ if $x$ is the root, and $bc(f_x) \cup S_y$ if $x$ is a child of $y$.
- $C_x$ is $bc(f_x)$ if $x$ is the root, and $bc(f_x) \setminus S_y$ if $x$ is a child of $y$.
- $G_x^*$ is the subgraph of $G^*$ induced by $\{f_z \mid z$ is a descendent of $x\}$ and is $G^*$ if $x$ is the root, and is the connected component of $G^* \setminus E^*[S_y]$ containing $f_x$ if $x$ is a child of $y$.

If $x$ is a child of $y$, $x$ has a parent frame $F_x \subseteq S_y$ such that:

- $F_x$ is the face cycle in $S_y$ that corresponds to $G_x^*$.

An s-t-decomposition is good if the tree has height $O(\log n)$ and each frame has alteration number 2 or 4. The name s-t-decomposition is chosen based on the following

Observation 3.4. Each vertex of $G$ is in exactly one $C_x$, and each $C_x$ is a truncated s-t-graph.

Theorem 3.5. Any acyclic single-source plane digraph has a good s-t-decomposition.

We defer the proof to section 3.1. The reason for studying s-t-decompositions in the context of reachability is the following
Lemma 3.6. If \( u \sim v \) where \( u \in C_x \) and \( v \in C_y \) then either \( x = y \) or \( x \) has a child \( z \) that is ancestor to \( y \) such that any \( u \sim v \) path contains a vertex in \( F_z \).

Since (by theorem 3.5) we can assume the alternation number is at most 4, this reduces the reachability question to the problem of finding the at most 4 “last” vertices on \( F_z \cap C_x \) that can reach \( v \) and then checking in \( C_x \) if \( u \) can reach either of them. In section 3.2 we will show how to do this efficiently when \( F_z \) is a 2-frame, that is, has alternation number 2, and in section 3.3 we will extend this to the case when \( F_z \) is a 4-frame, that is, has alternation number 4.

3.1 Constructing an s-t-decomposition

The s-t-decomposition recursively chooses a maximal truncated s-t-subgraph \( H \) of the graph \( G \). Since \( G \) was embedded in the plane, the subgraph \( H \) is embedded in the plane, and all vertices of \( G \setminus H \) lie in a unique face of \( H \). We may choose a tree-cotree composition wisely, such that for each face of \( H \), the restriction of \( T^* \) to the subfaces of that face is again a dual spanning tree (Lemma 3.8).

We also have to choose \( H \) carefully to ensure logarithmic height, and a limited alternation number on the frames. There are two cases: 2-frame-nodes have only small children, while for 4-frame-nodes, we only need to ensure that the 4-frame children themselves are small.

Lemma 3.7. Let \( G = (V, E) \) be a plane graph, let \( G^* = (V^*, E^*) \) be its dual, let \( (T, T^*) \) be a tree/cotree decomposition of \( G \), and let \( S \) be a subgraph of \( G \) such that \( S \cap T \) is connected. Then the faces of \( S \) correspond to connected components of \( T^* \setminus E^*[S] \).

Proof. Let \( S^* \) be the dual of \( S \), then \( S^* = G^*/(G^* \setminus E^*[S]) \) and the claim is equivalent to saying that the components of \( G^* \setminus E^*[S] \) correspond to the components of \( T^* \setminus E^*[S] \). Consider a pair of faces \( f_1, f_2 \in V^* \). Clearly, if they are in separate components of \( G^* \setminus E^*[S] \), they are also in separate components in \( T^* \setminus E^*[S] \).

On the other hand, suppose \( f_1 \) and \( f_2 \) are in different components in \( T^* \setminus E^*[S] \). Then there exists an edge \( e^* \in E^*[S] \cap T^* \) separating them. The corresponding edge \( e \in E[S] \) induces a cycle in \( T \), which is also part of \( S \) since \( S \cap T \) is connected. The dual to that cycle is an edge cut in \( G^* \) that separates \( f_1 \) from \( f_2 \). □

Lemma 3.8. Let \( T \) be a spanning tree where all edges point away from the source \( s \) of \( G \), then for any node \( x \) in an st-decomposition of \( G \), the subgraph \( T_x \) of \( T^* \) induced by \( V^*[G_x] \) is a connected subtree of \( T^* \).

Proof. If \( x \) is the root, this trivially holds. If \( x \) has a parent \( y \), \( G_x^* \) corresponds to a face in \( S_y \). Now \( S_y \cap T \) is connected since \( S_y \) is the union of backward-closed graphs, and the result follows from Lemma 3.7. □

Lemma 3.9. Let \( x \) be a node in an st-decomposition whose parent frame \( F_x \) has alternation number 2, and let \( A^* \) be the set of faces in \( T_x^* \) incident to the target corner of \( F_x \). Then for any child \( y \) of \( x \):

\[
A^* \subseteq V^*[T_y^*] \quad \implies \quad F_y \text{ has alternation number 4.}
\]

\[
A^* \not\subseteq V^*[T_y^*] \quad \implies \quad F_y \text{ has alternation number 2.}
\]

Proof. Let \( t_x \) be the target corner of \( F_x \) and let \( A^* \) be the set of faces in \( T_x^* \) incident to \( t_x \). For any child \( y \) if \( x, F_y \) consists of a (possibly empty) segment of \( F_x \) and two directed paths that meet at a new target corner \( t_y \). Each target corner of \( F_y \) must therefore be at either \( t_x \) or \( t_y \). Now if \( A^* \subseteq V^*[T_y^*] \), then both \( t_x \) and \( t_y \) are target corners of \( F_y \), otherwise only \( t_y \) is. Either way the result follows. □

Lemma 3.10. Let \( x \) be a node in an st-decomposition whose parent frame \( F_x \) has alternation number 4, and let \( A^0 \) and \( A^1 \) be the sets of faces in \( T_x^* \) incident to the target corners of \( F_x \). Then for any child \( y \) of \( x \):

\[
A^0 \not\subseteq V^*[T_y^*] \lor A^1 \not\subseteq V^*[T_y^*] \quad \implies \quad F_y \text{ has alternation number at most 4.}
\]

\[
A^0 \not\subseteq V^*[T_y^*] \land A^1 \not\subseteq V^*[T_y^*] \quad \implies \quad F_y \text{ has alternation number 2.}
\]
Proof. Let \( t_{x}^0 \) and \( t_{x}^1 \) be the two target corners of \( F_x \) and for \( i \in \{0, 1\} \) let \( A^i \) be the set of faces in \( T_x^i \) incident to \( t_{x}^i \). For any child \( y \) of \( x \), \( F_y \) consists of a (possibly empty) segment of \( F_x \) and two directed paths that meet at a new target corner \( t_y \). Each target corner of \( F_y \) must therefore be at either \( t_{y}^0 \), \( t_{x}^0 \), or \( t_{x}^1 \). Now if \( A^i \not\subset V^i[T_y^i] \) for some \( i \in \{0, 1\} \), then \( t_{x}^i \) is not a target corner of \( F_y \). So the number of target corners in \( F_y \) is at least 1, and at most 3 minus the number of such \( i \), and the result follows. \( \square \)

**Proof of Theorem 3.5.** Let \( s \) be the source of \( G \) and let \((T, T^*)\) be a tree/cotree decomposition of \( G \) such that all edges in \( T \) point away from \( s \). The st-decomposition can be constructed recursively as follows. Start with the root. In each step we have a node \( x \) and by Lemma 3.8, the subgraph \( T^*_x \) induced in \( T^* \) by \( V^*[G_x^*] \) is a tree. The goal is to select a face \( f_x \) such that for each child \( y \):

- The alternation number of \( f_y \) is at most 4, and
- For each child \( z \) of \( y \), \( |T^*_z| \leq \frac{1}{2}|T^*_x| \).

If we can do this for all \( x \), we are done. There are 3 cases:

**\( x \) is the root** Let \( f_x \) be the median of \( T^*_x = T^* \). Since \( S_x = bc(f_x) \) is a truncated s-t-graph with a single source, all faces other than \( f_x \) have alternation number at most 2. So for each child \( y \), \( |T^*_y| \leq \frac{1}{2}|T^*_x| \).

**\( F_x \) has alternation number 2** Let \( f_x \) be the median of \( T^*_x \). By Lemma 3.9, all faces other than \( f_x \) have alternation number at most 4. So for each child \( y \), since \( f_x \) is the median, \( |T^*_y| \leq \frac{1}{2}|T^*_x| \).

**\( F_x \) has alternation number 4** Let \( t_0 \) and \( t_1 \) be the local targets of \( F_x \) and let \( f_0, f_1 \in V^*[T^*_x] \) be (not necessarily distinct) faces incident to \( t_0 \) and \( t_1 \) respectively. Now choose \( f_x \) as the projection of the median \( m \) of \( T^*_x \) on the path \( f_0, \ldots, f_1 \) in \( T^*_x \). By Lemma 3.10 this means that for any child \( y \) of \( x \), the alternation number of the parent frame \( F_y \) is at most 4.

- If \( f_x = m \) then \( |T^*_y| \leq \frac{1}{2}|T^*_x| \).
- If \( f_x \neq m \) and \( T^*_x \) is not the component of \( m \) in \( T^*_x \backslash E^*[bc(f_x)] \), then \( |T^*_y| \leq \frac{1}{2}|T^*_x| \).
- If \( f_x \neq m \) and \( T^*_x \) is the component of \( m \) in \( T^*_x \backslash E^*[bc(f_x)] \), then \( T^*_y \) contains neither \( f_0 \) nor \( f_1 \) by Lemma 3.10, the parent frame \( F_y \) has alternation number at most 2 and we have just shown this means any child \( z \) of \( y \) has \( |T^*_z| \leq \frac{1}{2}|T^*_y| \leq \frac{1}{2}|T^*_x| \). \( \square \)

Note that this construction can be implemented in linear time by using ideas similar to \([1]\).

### 3.2 2-frames

**Definition 3.11.** Let \( x \) be a node in an st-decomposition whose parent frame \( F_x \) is a 2-frame, let \( s_x \) be the source corner and \( t_x \) the target corner of \( F_x \). Let \( L_x \) and \( R_x \) be the two directed segments of \( F_x \) such that:

- \( L_x \) is the counterclockwise-pointing segment of \( F_x \) that starts at \( s_x \) and ends at \( t_x \).
- \( R_x \) is the clockwise-pointing segment of \( F_x \) that starts at \( s_x \) and ends at \( t_x \).

Let \( y \) be an ancestor of \( x \) in an st-decomposition of \( G = (V, E) \), let \( v \in V[C_y] \), and suppose \( F_y \) is a 2-frame. We will show how to ”project” \( v \) efficiently, to vertices of \( F_y \), in a way that preserves reachability.

**Definition 3.12.** Let \( T \) be an st-decomposition of \( G = (V, E) \). For any vertex \( v \in V \) define:

\[
X_2(v) := \left\{ x \mid x \text{ is an ancestor to } v, \text{ and either} \right\}
\]

\[
d_2(v) := |X_2(v)| - 1
\]
We may enumerate \(X_2(v) = \{x_0, x_1, \ldots, x_{d_2[v]}\}\) such that \(d_2[x_i] = i\). For \(0 \leq i < d_2[v]\) let

\[
L_i(v) := L_{x_{i+1}} \\
R_i(v) := R_{x_{i+1}} \\
\hat{L}_i(v) := \left\{ w \in L_i(v) \mid \exists c \in \text{cor}(L_{x_{i+1}}), (w, w') \in E \setminus \{(w, \text{succ}(L_i(v), w))\} : (w, w') \text{ is incident to } c \wedge w' \sim v \right\} \\
\hat{R}_i(v) := \left\{ w \in \text{cor}(R_i(v)) \mid \exists c \in R_{x_{i+1}}, (w, w') \in E \setminus \{(w, \text{succ}(R_i(v), w))\} : (w, w') \text{ is incident to } c \wedge w' \sim v \right\} \\
l_i(v) := \text{the last vertex in } \hat{L}_i(v) \\
r_i(v) := \text{the last vertex in } \hat{R}_i(v)
\]

**Lemma 3.13.** For any vertex \(v \in V\) and \(0 \leq i < d_2[v]\), for any \(d_2[v] \leq j \leq i\), \(l_i(v) \in L_j(v) \implies l_i(v) \in \hat{L}_j(v)\) and \(r_i(v) \in R_j(v) \implies r_i(v) \in \hat{R}_j(v)\).

**Proof.** Note that \(G^*_{x_{i+1}}\) is a subface of \(G^*_{x_{j+1}}\) for all such \(j\). Thus, if an edge \((w, w')\) is incident to a corner \(c \in L_{i+1}\), it is also incident to a supercorner \(c' \in L_{j+1}\) of \(c\). Furthermore, if \(\text{succ}(L_j(v), w)\) is incident to \(c \in L_{i+1}\), then \(\text{succ}(L_j(v), w) = \text{succ}(L_i(v), w)\). Thus, \(l_i(v) \in L_j(v) \implies l_i(v) \in \hat{L}_j(v)\).

We can now rephrase the question as the problem of finding an efficient data structure for computing the functions \(l_i(v)\) and \(r_i(v)\). The main idea that doesn’t quite work is to represent each function with a suitable rooted forest and use a level ancestor structure on that forest to answer queries.

**Definition 3.14.** For any vertex \(v \in V\) let

\[
p_l[v] := \begin{cases} 
\bot & \text{if } d_2[v] = 0 \\
L_{d_2[v] - 1}(v) & \text{if } d_2[v] > 0
\end{cases} \quad p_r[v] := \begin{cases} 
\bot & \text{if } d_2[v] = 0 \\
R_{d_2[v] - 1}(v) & \text{if } d_2[v] > 0
\end{cases}
\]

and let \(T_l\) and \(T_r\) denote the rooted forests over \(V\) whose parent pointers are \(p_l\) and \(p_r\) respectively.

Ideally we would now find \(l_i(v)\) as the ancestor to \(v\) of maximum depth not exceeding \(i\) in \(T_l\).

**Definition 3.15.** For any \(v \in V\) and any \(i \geq 0\) let

\[
l'_i(v) := \begin{cases} 
v & \text{if } d_2[v] \leq i \\
l'_i(p_l[v]) & \text{otherwise}
\end{cases} \quad r'_i(v) := \begin{cases} 
v & \text{if } d_2[v] \leq i \\
r'_i(p_r[v]) & \text{otherwise}
\end{cases}
\]

**Observation 3.16.** Let \(v \in V\) and \(i \geq 0\) be given, then

\[
\begin{align*}
& i = d_2[v] - 1 \quad \implies \quad l'_i(v) = l_i(v) \quad \wedge \quad r'_i(v) = r_i(v) \\
& i > d_2[v] - 1 \quad \implies \quad l'_i(v) = v \quad \wedge \quad r'_i(v) = v
\end{align*}
\]

**Observation 3.17.** Let \(v \in V\) and \(0 \leq i \leq j\) then it follows trivially from the recursion, that

\[
l'_i(l'_j(v)) = l'_i(v) \quad \wedge \quad r'_i(r'_j(v)) = r'_i(v)
\]

Unfortunately the idea of level ancestry alone does not always work. Fortunately, when this happens we have some more structure we can use. Namely, when we have a crossing.

**Definition 3.18.** For any level, \(i\), let the define \(W_i(v)\) as the “important” subset of \(\{l_i(v), r_i(v)\}\):

\[
W_i(v) := \begin{cases} 
\{l_i(v)\} & \text{if } r_i(v) = s_i(v) \\
\{r_i(v)\} & \text{if } l_i(v) = s_i(v) \\
\{l_i(v), r_i(v)\} & \text{otherwise}
\end{cases}
\]
Note that if \( d_2[x] \leq i \), then \( x \) can reach \( v \) if and only if \( x \) can reach \( W_i(v) \).

**Lemma 3.19.** Let \( v \in V \) and let \( 0 \leq i < d_2[v] - 1 \), then

\[
\begin{align*}
\ell_i(v) &\neq \ell_i'(l_{i+1}(v)) & \implies W_i \subseteq \{ \ell_i'(m), r_i'(m) \}, \text{ where } m = r_{i+1}(v) \\
r_i(v) &\neq r_i'(r_{i+1}(v)) & \implies W_i \subseteq \{ \ell_i'(m), r_i'(m) \}, \text{ where } m = l_{i+1}(v)
\end{align*}
\]

**Proof.** Suppose \( \ell_i(v) \neq \ell_i'(l_{i+1}(v)) \) (the case \( r_i(v) \neq r_i'(r_{i+1}(v)) \) is symmetrical). Consider any path \( P \) from \( l_i(v) \) to \( v \). It must at some point cross the \( i + 1 \)st frame \( F_{i+1}(v) \) of \( v \). Let \( w \) be the last vertex in \( P \cap (F_{i+1}(v)) \). If \( w \in L_{i+1}(v) \) but is not the source, then \( w \in L_{i+1}(v) \) since it is the last such vertex on \( P \). But then \( l_i(v) \sim l_i'(l_{i+1}(v)) \) and either \( l_i(v) \in L_i(v) \) implying \( l_i(v) = l_i(v) = l_i'(l_{i+1}(v)) \) (by acyclicity and Lemma 3.13), or \( d_2(l_{i+1}(v)) = i + 1 \) again implying \( l_i(v) = l_i'(l_{i+1}(v)) \), since \( L_i(l_{i+1}(v)) = L_i(v) \).

Thus, \( w \in R_{i+1}(v) \) and by a similar argument \( l_i(v) \) can reach \( m = r_{i+1}(v) \). Now any path from \( r_i(v) \) to \( v \) must either cross \( P \) or \( R_{i+1}(v) \) or contain a vertex that can reach the source of \( R_{i+1}(v) \). But then, \( l_i(v) \) and \( r_i(v) \) can both reach \( m \). Now if \( d_2[m] \leq i \), then \( W_i(v) = \{ m \} \) and by Lemma 3.16 \( m = l_i'(m) = r_i'(m) \).

On the other hand, if \( d_2[m] > i \) then \( d_2[m] = i + 1 \). But then \( F_i(m) = F_i(v) \), and thus, \( l_i(v) = l_i'(m) \) and \( r_i(v) = r_i'(m) \), implying \( W_i(v) \subseteq \{ l_i'(m), r_i'(m) \} \), where we have \( \subseteq \) if one can reach the other. \( \square \)

**Definition 3.20.** Let \( v \in V \) and let \( 0 \leq i < d_2[v] \).

\[
m_i(v) := \begin{cases} 
  v & \text{if } i + 1 = d_2[v] \\
  l_{i+1}(v) & \text{if } i + 1 < d_2[v] \land r_i(v) \neq r_i'(r_{i+1}(v)) \\
  r_{i+1}(v) & \text{if } i + 1 < d_2[v] \land l_i(v) \neq l_i'(l_{i+1}(v)) \\
  m_{i+1}(v) & \text{otherwise}
\end{cases}
\]

**Corollary 3.21.** Let \( v \in V \) and let \( 0 \leq i < d_2[v] - 1 \). If \( l_i(v) \neq l_i'(l_{i+1}(v)) \) or \( r_i(v) \neq r_i'(r_{i+1}(v)) \) then

\[
W_i(v) \subseteq \{ l_i'(m_i(v)), r_i'(m_i(v)) \}
\]

**Proof.** This is just a reformulation of Lemma 3.19 in terms of \( m_i(v) \). \( \square \)

**Lemma 3.22.** For any vertex \( v \in V \) and any \( 0 \leq i < d_2[v] \)

\[
W_i(v) \subseteq \{ l_i'(m_i(v)), r_i'(m_i(v)) \}
\]

**Proof.** The proof is by induction on \( j \), the number of times the “otherwise” case is used before reaching one of the other cases when expanding the recursive definition of \( m_i(v) \).

For \( j = 0 \), either \( i + 1 = d_2[v] \) and the result follows from Lemma 3.16, or \( i + 1 < d_2[v] \) and \( l_i(v) \neq l_i'(l_{i+1}(v)) \) or \( r_i(v) \neq r_i'(r_{i+1}(v)) \). In either case, by Corollary 3.21 \( W_i(v) \subseteq \{ l_i'(m_i(v)), r_i'(m_i(v)) \} \).

For \( j > 0 \) we have \( i + 1 < d_2[v] \) and \( l_i(v) = l_i'(l_{i+1}(v)) \) and \( r_i(v) = r_i'(r_{i+1}(v)) \) and \( m_i(v) = m_{i+1}(v) \). By induction we can assume that \( W_{i+1}(v) \subseteq \{ l_{i+1}(m_{i+1}(v)), r_{i+1}(m_{i+1}(v)) \} = \{ l_{i+1}(m_i(v)), r_{i+1}(m_i(v)) \} \).
If $|W_{i+1}(v)| = 2$ then it follows that $W_{i+1}(v) = \{l_{i+1}(v), r_{i+1}(v)\} = \{l'_i(m_i(v)), r'_i(m_i(v))\}$ implying $l_{i+1}(v) = l'_i(m_i(v))$ and $r_{i+1}(v) = r'_i(m_i(v))$. Then, $l_i(v) = l'_i(l_{i+1}(m_i(v))) = l'_i(m_i(v))$ and $r_i(v) = r'_i(r_{i+1}(m_i(v))) = r'_i(m_i(v))$ and we are done.

Otherwise $|W_{i+1}(v)| = 1$ and we can assume wolog. that $W_{i+1}(v) = \{r_{i+1}(v)\}$. Then, $\{l_{i+1}(v), l'_i(m_i(v)), r'_i(m_i(v)), r_{i+1}(v)\} \subseteq R_{i+1}(v)$, and $l_{i+1}(v)$ is the source of $L_{i+1}(v) \cup R_{i+1}(v)$ so $l_{i+1}(v) \rightsquigarrow l'_i(m_i(v)) \rightsquigarrow r'_i(m_i(v)) = r_{i+1}(v)$. The last equality immediately gives $r_i(v) = r'_i(r_{i+1}(v)) = r'_i(r'_i(m_i(v))) = r'_i(m_i(v))$. Finally, either $l_i(v)$ is the source of $L_i(v) \cup R_i(v)$ so $W_i(v) = \{r_i(v)\} = \{r'_i(m_i(v))\} \subseteq \{l'_i(m_i(v)), r'_i(m_i(v))\}$, or since $l_{i+1}(v) \rightsquigarrow l'_i(m_i(v))$ we must have $l_i(v) = l'_i(l_{i+1}(v)) = l'_i(l'_i(m_i(v))) = l'_{i+1}(m_i(v))$, and thus $W_i(v) \subseteq \{l_i(v), r_i(v)\} = \{l'_i(m_i(v)), r'_i(m_i(v))\}$.

We thus desire to compute $l', r'$, and $m$ in an efficient way. This can be done because the $m$-nodes form a tree. Once we found the last crossing before a given level, we only need to use a chain of $l'$s or $r'$s.

**Observation 3.23.** Let $u$ be a vertex at level at most $i$. Then $u$ can reach $v$ iff $u$ can reach $m_i(v)$. Furthermore, if $u \in F_k(v) \cap F_k(m_i(v))$, then $u \in \overline{F}_k(v)$ iff $u \in \overline{F}_k(m_i(v))$.

**Proof.** If $u$ can reach $m_i(v)$, then since $m_i(v)$ can reach $v$, clearly $u$ can reach $v$. Assume $u$ can reach $v$, assume $m_i(v) = r_{i+1}(v)$, and consider a path from $u$ to $v$. Since $u$ is at level at most $i$, this path must have a last crossing point with $F_k(v)$. Call this vertex $x$. From Corollary 3.21 $W_i(v) \subseteq \{l'_i(m_i(v)), r'_i(m_i(v))\}$. As noted, since $x$ can reach $v$, $x$ can reach some point of $W_i(v)$. Thus, $x$ can reach at least one of $\{l'_i(m_i(v)), r'_i(m_i(v))\}$, and via that vertex, reach $m_i(v)$.

Restrictions of this to paths incident to certain corners yield the second claim. □

**Lemma 3.24.** Let $v \in V$ and let $0 \leq i < j < d[v]$, then $m_i(v) = m_i(m_j(v))$

**Proof.** We may assume $m_i(v) \neq m_j(v)$, as it otherwise trivially holds. Note that $d_2[m_j(v)] \leq j$ need not be $j$. Consider the last level, $k \geq i$, such where $m_k(v) \neq m_{k+1}(v)$. Assume wolog that $l'_{k}(l_{k+1}(v)) \neq l_{k}(v)$. But we noted that any vertex at level at most $j$ may reach $v$ iff it may reach $m_j(v)$. Thus, $l_{k+1}(v)$ can reach $m_j(v)$. Since $m_i(v) = m_k(v) \neq m_j(v)$, we may conclude $d_2[m_j(v)] = k + 1$. If $m_j(v) \in L_{k+1}(v)$, then $x \in L_k(v)$ may reach $v$ iff it can reach $m_j(v) = l_{k+1}(v)$, and then $l'_{k}(l_{k+1}(v)) = l_{k}(v)$ and we have no crossing. If $m_j(v) \in R_{k+1}(v)$, then if we have a crossing, $m_k(v) = m_j(v)$.

But if $d_2[m_j(v)] > k+1$, then $L_k(m_j(v)) = L_k(v)$. Combining this with Obs. 3.23, we now get $l_k(v) = l_k(m_j(v))$ and $l'_{k}(l_{k+1}(m_j(v))) = l'_{k}(l_{k+1}(v))$. Thus, $m_k(m_j(v)) = m_k(m_j(v))$. But $r_{k+1}(m_j(v)) = r_{k+1}(v)$ since $R_{k+1}(m_j(v)) = R_{k+1}(v)$, and thus $m_k(m_j(v)) = m_{k+1}(m_j(v)) = m_{k+1}(v) = m_k(v)$.

Since there are no more crossings for $v$ at levels $k \ldots i$, this also holds for $m_j(v)$, and thus $m_i(m_j(v)) = m_k(m_j(v)) = m_k(v) = m_i(v)$ as desired. □

This means we can represent $m$ with a tree as follows

**Definition 3.25.** For any vertex $v \in V$ let

$$M[v] := \{i \mid 0 < i < d_2[v] \land m_{i-1}(v) \neq m_i(v)\}$$

$$p_m[v] := \begin{cases} 
\perp & \text{if } M[v] = \emptyset \\
\max_{M[v] \neq \emptyset} M[v] - 1 & \text{otherwise} 
\end{cases}$$

And define $T_m$ as the rooted forest over $V$ whose parent pointers are $p_m$.

**Theorem 3.26.** There exists a practical RAM data structure that for any good st-decomposition of a graph with $n$ vertices uses $O(n)$ words of $O(\log n)$ bits and can return a superset of $W_i(v)$ in constant time.
Proof. For any vertex \( v \in V \) let

\[
D_l[v] := \{ i \mid v \text{ has a proper ancestor } w \text{ in } T_l \text{ with } d_2[w] = i \}
\]

\[
D_r[v] := \{ i \mid v \text{ has a proper ancestor } w \text{ in } T_r \text{ with } d_2[w] = i \}
\]

Now, store levelancestor structures for each of \( T_l, T_r, \) and \( T_m \), together with \( d_2[v], D_l[v], D_r[v], \) and \( M[v] \) for each vertex. Since the height of the st-decomposition is \( O(\log n) \) each of \( D_l[v], D_r[v], \) and \( M[v] \) can be represented in a single \( O(\log n) \)-bit word. This representation allows us to find \( d_2(m_i(v)) = \text{succ}(M[v] \cup \{d_2[v]\}, i) \) in constant time, as well as computing the depth in \( T_m \) of \( m_i(v) \). Then using the levelancestor structure for \( T_m \) we can compute \( m_i(v) \) in constant time.

Similarly, this representation of the \( D_l[v] \) set lets us compute the depth in \( T_l \) of \( l_i'(v) \) in constant time, and with the levelancestor structure that lets us compute \( l_i'(v) \) in constant time. A symmetric argument shows that we can compute \( r_i'(v) \) in constant time. Finally, lemma [3.22](3.22) says we can compute a superset of \( W_i(v) \) in constant time given constant-time functions for \( l', r', \) and \( m \). \( \square \)

### 3.3 4-frames

**Definition 3.27.** Let \( x \) be a non-root node in an st-decomposition, and let \( y \) be its parent. We will name 4 (not necessarily distinct) corners \( s^0_x, s^1_x, t^0_x, \) and \( t^1_x \) on the \( F_x \) cycle as follows:

- If \( F_x \) is a 2-frame let \( s^0_x = s^1_x \) be the source corner of \( F_x \) and let \( t^0_x = t^1_x \) be the target corner of \( F_x \).
- If \( F_x \) is a 4-frame \( y \) is not the root. Let \( s^0_x \) and \( s^1_x \) be the source corners on \( F_x \) and let \( t^0_x \) and \( t^1_x \) be the target corners on \( F_x \), numbered such that their clockwise cyclic order on \( F_x \) is \( s^0_x, t^0_x, s^1_x, t^1_x \), and that \( t^0_x = l^0_y \) if possible, else \( t^1_x = l^1_y \). In particular, if \( F_y \) is a 2-frame \( t^0_x = t^0_y \).

**Definition 3.28.** Let \( x \) be a non-root node in an st-decomposition. Let \( E_x \) denote the set of edges \((w, w')\) such that \( w \in V[F_x] \) and there exists a descendent \( y \) of \( x \) with \( w' \in V[C_y] \). Let \( E_x \) have the natural cyclic order given by the embedding.

**Definition 3.29.** Let \( x \) be a non-root node in an st-decomposition such that \( F_x \) is a 4-frame. Define sets \( L^0_x, R^0_x, L^1_x, R^1_x \subseteq E_x \) forming a partition of \( E_x \) into 4 disjoint sets where for \( \alpha \in \{0, 1\} \):

- \( L^0_x \) and \( R^\alpha_x \) are contiguous subsets of \( E_x \) in the cyclic order.
- \( L^\alpha_x \) is totally ordered by the counterclockwise order on \( E_x \).
- \( R^\alpha_x \) is totally ordered by the clockwise order on \( E_x \).
- The corners in \( F_x \) incident to \( L^\alpha_x \) are contained in the dissegment of \( F_x \) between \( s^{1-\alpha}_x \) and \( t^\alpha_x \).
- The corners in \( F_x \) incident to \( R^\alpha_x \) are contained in the dissegment of \( F_x \) between \( s^\alpha_x \) and \( t^\alpha_x \).
- If \( y \) is the parent to \( x \) and is not the root, and \( F_y \) is a 2-frame then \( L^\alpha_y \cap E_x = L^\alpha_x \cap E_y \) and \( R^\alpha_y \cap E_x = R^\alpha_x \cap E_y \).

The importance of the order is that if e.g. \((u, u') \in L^\alpha_x \) comes before \((v, v') \in L^\alpha_x \) then \( u \prec v \). Thus we only need to find at most one edge from each set to determine reachability across the edge cut defined by \( E_x \).

**Definition 3.30.** Let \( T \) be an st-decomposition of \( G = (V, E) \). For any vertex \( v \in V \) define:

\[
c[v] := \text{The node } x \text{ in } T \text{ such that } v \in V[C_x]
\]

\[
d[v] := \text{The depth of } c[v] \text{ in } T
\]

\[
J_2[v] := \{\text{depth}(x) \mid x \text{ is a non-root ancestor to } c[v] \text{ in } T \text{ and } F_x \text{ is a 2-frame}\}
\]

\[
j_2[v] := \max(J_2[v])
\]
The number \( j_2[v] \) is especially useful for 4-frame nodes. On the path from the root to the component of \( v \) in the s-t-decomposition tree, there will be a last component whose frame is a 2-frame. We call the depth of the next component on the path \( j_2[v] \). If \( C[v] \) has a 4-frame, then for the rest of the path, that is, depth \( i \) with \( j_2[v] \leq i < d[v] \), we will have 4-frames nested in 4-frames, which gives a lot of useful structure.

**Definition 3.31.** For any \( j_2[v] \leq i < d[v] \) and \( \alpha \in \{0, 1\} \), let \( x \) be the ancestor of \( c[v] \) at depth \( i + 1 \) and define:

\[
F_\alpha(v) := F_x,
E_\alpha(v) := E_x,
L_\alpha^\alpha(v) := L_x^\alpha,
R_\alpha^\alpha(v) := R_x^\alpha,
\]

\[
\hat{L}_\alpha^\alpha(v) := \{(w, w') \in L_\alpha^\alpha(v) \mid w' \sim v\} \quad \text{with the total order inherited from } L_\alpha^\alpha(v)
\]

\[
\hat{R}_\alpha^\alpha(v) := \{(w, w') \in R_\alpha^\alpha(v) \mid w' \sim v\} \quad \text{with the total order inherited from } R_\alpha^\alpha(v)
\]

\[
\hat{F}_\alpha(v) := \hat{L}_\alpha^0(v) \cup \hat{R}_\alpha^1(v) \cup \hat{L}_\alpha^1(v) \cup \hat{R}_\alpha^1(v)
\]

\[
l_\alpha^\alpha(v) := \begin{cases} 
\bot & \text{if } \hat{L}_\alpha^\alpha(v) = \emptyset \\
\text{the initial vertex of the last edge in } \hat{L}_\alpha^\alpha(v) & \text{otherwise}
\end{cases}
\]

\[
r_\alpha^\alpha(v) := \begin{cases} 
\bot & \text{if } \hat{R}_\alpha^\alpha(v) = \emptyset \\
\text{the initial vertex of the last edge in } \hat{R}_\alpha^\alpha(v) & \text{otherwise}
\end{cases}
\]

\[
s_\alpha^\alpha(v) := \text{The vertex associated with } s_x^0
\]

\[
t_\alpha^\alpha(v) := \text{The vertex associated with } t_x^1
\]

We know from Section 3.2 that we can find the relevant vertices on each 2-frame surrounding \( v \). The goal in this section is a data structure for efficiently computing \( l_\alpha^\alpha(v) \) and \( r_\alpha^\alpha(v) \) for \( j_2[v] \leq i < d[v] \).

**Lemma 3.32.** For any vertex \( v \in V \) and \( j_2[v] \leq i < d[v] \): \( \hat{F}_\alpha(v) \neq \emptyset \)

**Proof.** Let \( x \) be the ancestor of \( c[v] \) at depth \( i + 1 \). Since \( G \) is a single-source graph, there is a path from \( s \) to \( v \). This path must contain a vertex in \( V[F_x] \), which is reachable from \( s_x^0 \) or \( s_x^1 \) (or both). But then the edge following the last such vertex on the path must be in \( \hat{L}_\alpha^0(v) \cup \hat{R}_\alpha^1(v) \cup \hat{L}_\alpha^1(v) \cup \hat{R}_\alpha^1(v) \) which is therefore nonempty. \( \square \)

**Lemma 3.33.** Given any vertex \( v \in V \), \( j_2[v] \leq i < d[v], \alpha \in \{0, 1\} \), and \( (w, w') \in E_\alpha(v) \). Let \( j = \max \{d[w], j_2[v]\} \) and \( k = \min \{d[w'], d[v]\} \) then:

\[
(w, w') \in \hat{L}_\alpha^\alpha(v) \implies (w, w') \in \bigcup_{j \leq i' < k} \hat{L}_\alpha^\alpha(v)
\]

\[
(w, w') \in \hat{R}_\alpha^\alpha(v) \implies (w, w') \in \bigcup_{j \leq i' < k} \hat{R}_\alpha^\alpha(v)
\]

**Proof.** Clearly \( (w, w') \in E_\alpha \) for all \( j \leq i' < k \). Suppose \( (w, w') \in \hat{L}_\alpha^\alpha(v) \subseteq \hat{L}_\alpha^\alpha(v) \), then since \( j \leq i < k \) the definition gives us \( (w, w') \in \hat{L}_\alpha^\alpha(v) \) for all \( j \leq i' < k \). And since \( w' \sim v \) this implies \( (w, w') \in \hat{L}_\alpha^\alpha(v) \) for all \( j \leq i' < k \) and the result follows. The case for \( R \) is symmetric. \( \square \)
Definition 3.34. For any vertex $v \in V$ and $\alpha \in \{0, 1\}$ let

$$p_i^\alpha[v] := \begin{cases} \bot & \text{if } d[v] = 0 \lor F_{d[v]-1}(v) \text{ is a 2-frame} \\ l_{d[v]-1}^\alpha(v) & \text{otherwise} \end{cases}$$

$$p_r^\alpha[v] := \begin{cases} \bot & \text{if } d[v] = 0 \lor F_{d[v]-1}(v) \text{ is a 2-frame} \\ r_{d[v]-1}^\alpha(v) & \text{otherwise} \end{cases}$$

and let $T_i^\alpha$ and $T_r^\alpha$ denote the rooted forests over $V$ whose parent pointers are $p_i^\alpha$ and $p_r^\alpha$ respectively.

Definition 3.35. For any $v \in V \cup \{\bot\}$, $\alpha \in \{0, 1\}$, and $i \geq j_2[v]$ let

$$l_i^\alpha(v) := \begin{cases} v & \text{if } v = \bot \lor d[v] \leq i \\ l_i^\alpha(p_i^\alpha[v]) & \text{otherwise} \end{cases}$$

$$r_i^\alpha(v) := \begin{cases} v & \text{if } v = \bot \lor d[v] \leq i \\ r_i^\alpha(p_r^\alpha[v]) & \text{otherwise} \end{cases}$$

Lemma 3.36. Let $v \in V$, $\alpha \in \{0, 1\}$, and $i \geq j_2[v]$ be given, then

$$i = d[v] - 1 \implies l_i^\alpha(v) = l_i^\alpha(v) \quad \land \quad r_i^\alpha(v) = r_i^\alpha(v)$$

$$i \leq d[v] - 1 \implies l_i^\alpha(v) \in \text{init}(\hat{L}_i^\alpha(v)) \cup \{\bot\} \quad \land \quad r_i^\alpha(v) \in \text{init}(\hat{R}_i^\alpha(v)) \cup \{\bot\}$$

$$i > d[v] - 1 \implies l_i^\alpha(v) = v \quad \land \quad r_i^\alpha(v) = v$$

Where by init(X), for a set of edges, X, we denote the initial vertices \{a \mid (a, b) \in X\}.

Proof. We will show this for $l'$ only, as $r'$ is completely symmetrical. If $i > d[v] - 1$ then $d[v] \leq i$ and we get $l_i^\alpha(v) = v$ directly from the definition of $l'$. Similarly if $i = d[v] - 1$ then $l_i^\alpha(v) = l_i^\alpha(p_i^\alpha[v]) = l_i^\alpha(l_{d[v]-1}^\alpha(v)) = l_i^\alpha(l_i^\alpha(v)) = l_i^\alpha(v) \in \text{init}(\hat{L}_i^\alpha(v)) \cup \{\bot\}$. Finally suppose $i < d[v] - 1$. If $l_i^\alpha(v) = \bot$ we are done, so suppose that is not the case. Let $v$ be the child of $l_i^\alpha(v)$ in $T_i$ that is ancestor to $v$. Then $l_i^\alpha(v) = l_i^\alpha(u) = p_i^\alpha[u] = l_i^\alpha(l_{d[u]-1}^\alpha(u))$. By definition of $l_i^\alpha(u)$ there exists an edge $(w, w') \in \hat{L}_{d[u]-1}(u)$ where $w = l_i^\alpha(l_{d[u]-1}^\alpha(u))$ and $w[v] \leq i < d[w'] \leq d[u]$ and by setting $(v, i, (w, w')) = (u, d[u] - 1, (w, w'))$ in lemma 3.33 we get $(w, w') \in \hat{L}_i^\alpha(u)$, and therefore $l_i^\alpha(v) \in \text{init}(\hat{L}_i^\alpha(u))$. But since $u \sim v$ we have $\hat{L}_i^\alpha(u) \subseteq \hat{L}_i^\alpha(v)$ and we are done.

Lemma 3.37. Let $v \in V$, $\alpha \in \{0, 1\}$, and $j_2[v] \leq i \leq j$ then

$$l_i^\alpha(l_j^\alpha(v)) = l_i^\alpha(v) \quad \land \quad r_i^\alpha(r_j^\alpha(v)) = r_i^\alpha(v)$$

Proof. $l_j^\alpha(v)$ is on the path from $v$ to $l_i^\alpha(v)$ in $T_i$, so this follows trivially from the recursion. The case for $r'$ is symmetric.

Lemma 3.38. Let $v \in V$, $\alpha \in \{0, 1\}$, and $j_2[v] \leq i < d[v] - 1$, then

$$l_i^\alpha(v) = \bot \implies l_i^\alpha(l_{i+1}^\alpha(v)) = \bot \quad \land \quad r_i^\alpha(v) = \bot \implies r_i^\alpha(r_{i+1}^\alpha(v)) = \bot$$

Proof. If $l_i^\alpha(v) = \bot$ then $\hat{L}_i^\alpha(v) = \emptyset$, so either $l_{i+1}^\alpha(v) = \bot$ implying $l_i^\alpha(l_{i+1}^\alpha(v)) = \bot$ by the definition of $l'$, or $l_{i+1}^\alpha(v) \notin \text{init}(\hat{L}_i^\alpha(v))$ so $d[l_{i+1}^\alpha(v)] = i + 1$ and by lemma 3.36, $l_i^\alpha(l_{i+1}^\alpha(v)) \in \text{init}(\hat{L}_i^\alpha(l_{i+1}^\alpha(v))) \cup \{\bot\} \subseteq \text{init}(\hat{L}_i^\alpha(v)) \cup \{\bot\} = \{\bot\}$ so again $l_i^\alpha(l_{i+1}^\alpha(v)) = \bot$. The case for $r$ is symmetric.
Lemma 3.39 (Crossing lemma). Let $v \in V$, $\alpha \in \{0, 1\}$, and $j_2[v] \leq i < d[v] - 1$.

$$l_i^\alpha(v) \neq l_i^\alpha(l_{i+1}^\alpha(v)) \quad \implies \quad l_i^\alpha(v) = l_i^\alpha(m) \land r_i^\alpha(v) = r_i^\alpha(m) \land d[m] = i + 1$$

where $m = r_{i+1}^\alpha(v) \neq \bot$

$$r_i^\alpha(v) \neq r_i^\alpha(r_{i+1}^\alpha(v)) \quad \implies \quad l_i^\alpha(v) = l_i^\alpha(m) \land r_i^\alpha(v) = r_i^\alpha(m) \land d[m] = i + 1$$

where $m = l_{i+1}^\alpha(v) \neq \bot$

Proof: Suppose $l_i^\alpha(v) \neq l_i^\alpha(l_{i+1}^\alpha(v))$ (the case $r_i^\alpha(v) \neq r_i^\alpha(r_{i+1}^\alpha(v))$ is symmetrical). Then $l_i^\alpha(v) \neq \bot$ by lemma 3.38. Thus there is a last edge $(w, w') \in L_i^\alpha(v)$ with $w = l_i^\alpha(v)$ and $d[w] \leq i < d[w']$ and a path $P = w \rightsquigarrow v$.

Now $(w, w') \notin E_{i+1}(v)$ since otherwise by Definition 3.29 $(w, w') \in L_{i+1}^\alpha(v)$ and since $w' \rightsquigarrow v$ even $(w, w') \in L_{i+1}^\alpha(v)$ implying $l_i^\alpha(v) = l_{i+1}^\alpha(v)$ and thus $l_i^\alpha(v) = l_i^\alpha(l_{i+1}^\alpha(v))$ by lemma 3.36, contradicting our assumption.

Since $(w, w') \notin E_{i+1}(v)$, the path $P$ must cross $L_{i+1}^\alpha(v)$. Let $(u, u')$ be the last edge in $P \cap L_{i+1}^\alpha(v)$. Then $w' \rightsquigarrow u$ so $d[u] \geq i + 1$ and $(u, u') \notin L_{i+1}^\alpha(v)$ since otherwise $d[l_{i+1}^\alpha(v)] = i + 1$ and hence by Lemma 3.36 $l_i^\alpha(v) = l_i^\alpha(l_{i+1}^\alpha(v))$, again contradicting our assumption.

Also, $l_i^\alpha(v) \neq l_{i+1}^\alpha(v)$ because $t_i^\alpha(v) = t_{i+1}^\alpha(v)$ would imply $(w, w') \in L_{i+1}^\alpha(v) \cup \{\bot\}$ which we have just shown is not the case.

Since $t_i^\alpha(v) \neq t_{i+1}^\alpha(v)$, then by definition $t_i^{1-\alpha}(v) = t_{i+1}^{1-\alpha}(v)$ and hence $L_{i+1}^{1-\alpha}(v) \subseteq L_i^{1-\alpha}(v)$ and $R_{i+1}^{1-\alpha}(v) \subseteq R_i^{1-\alpha}(v)$, implying $d[u'] \leq i$ for all $u' \in L_{i+1}^{1-\alpha}(v) \cup R_{i+1}^{1-\alpha}(v)$. Thus, $(u, u') \notin L_i^{1-\alpha}(v) \cup R_{i+1}^{1-\alpha}(v)$ since $d[u] > i$, and we can conclude that $(u, u') \notin \tilde{R}_{i+1}^\alpha(v)$.

But then we can choose $P$ so it goes through $(m, m')$ where $m = r_{i+1}^\alpha(v) \neq \bot$. Now $i + 1 \leq d[w'] \leq d[t_{i+1}^{1-\alpha}(v)] \leq i + 1$ so $d[m] = i + 1$.

Let $e$ be the last edge in $\tilde{R}_{i}^\alpha(v)$ then any path $r_i^\alpha(v) \rightsquigarrow v$ that starts with $e$ crosses $P \cup \tilde{R}_{i+1}^\alpha(v)$, implying that there exists such a path that contains $(m, m')$ and thus $r_i^{1-\alpha}(v) = r_i^{1-\alpha}(m)$. Since $d[m] = i + 1,$
then \( l_i^\alpha(v) = l_i^\alpha(m_i(v)) \) and \( r_i^\alpha(v) = r_i^\alpha(m_i(v)) \) follows from lemma 3.36.

**Definition 3.40.** Let \( v \in V, \alpha \in \{0, 1\} \), and \( 0 \leq i < d[v] \).

\[
m_i^\alpha(v) := \begin{cases} 
v & \text{if } i + 1 = d[v] \\
l_{i+1}^\alpha(v) & \text{if } i + 1 < d[v] \land r_i^\alpha(v) \neq r_i^\alpha(r_{i+1}^\alpha(v)) \\
r_{i+1}^\alpha(v) & \text{if } i + 1 < d[v] \land l_i^\alpha(v) \neq l_i^\alpha(l_{i+1}^\alpha(v)) \\
m_{i+1}^\alpha(v) & \text{otherwise} \\
\end{cases}
\]

**Corollary 3.41.** Let \( v \in V, \alpha \in \{0, 1\} \), and \( j_2[v] \leq i < d[v] - 1 \). If \( l_i^\alpha(v) \neq l_i^\alpha(l_{i+1}^\alpha(v)) \) or \( r_i^\alpha(v) \neq r_i^\alpha(r_{i+1}^\alpha(v)) \) then

\[
l_{i+1}^\alpha(v) = l_{i+1}^\alpha(m_{i+1}^\alpha(v)) \quad \land \quad r_{i+1}^\alpha(v) = r_{i+1}^\alpha(m_{i+1}^\alpha(v)) \quad \land \quad d[m_{i+1}^\alpha(v)] = i + 1
\]

**Proof.** This is just a reformulation of lemma 3.39 in terms of \( m_i^\alpha(v) \).

**Lemma 3.42.** For any vertex \( v \in V, \alpha \in \{0, 1\} \), and \( j_2[v] \leq i < d[v] \)

\[
l_i^\alpha(v) = l_i^\alpha(m_i^\alpha(v)) \quad \land \quad r_i^\alpha(v) = r_i^\alpha(m_i^\alpha(v))
\]

**Proof.** The proof is by induction on \( j \), the number of times the “otherwise” case is used before reaching one of the other cases when expanding the recursive definition of \( m_i(v) \).

For \( j = 0 \), either \( i + 1 = d[v] \) and the result follows from Lemma 3.36 or \( i + 1 < d[v] \) and \( l_i(v) \neq l_{i+1}(l_{i+1}(v)) \) or \( r_i(v) \neq r_{i}(r_{i+1}(v)) \). In either case we have by Corollary 3.41 that \( l_i^\alpha(v) = l_i^\alpha(m_i^\alpha(v)) \) and \( r_i^\alpha(v) = r_i^\alpha(m_i^\alpha(v)) \).

For \( j > 0 \) we have \( i + 1 < d[v] \) and \( l_i(v) = l_{i+1}(l_{i+1}(v)) \) and \( r_i(v) = r_{i}(r_{i+1}(v)) \) and \( m_i(v) = m_{i+1}(v) \). By induction we can assume that \( l_{i+1}^\alpha(v) = l_{i+1}^\alpha(m_{i+1}^\alpha(v)) \) and \( r_{i+1}^\alpha(v) = r_{i+1}^\alpha(m_{i+1}^\alpha(v)) \).

Then by Lemma 3.37 \( l_i^\alpha(l_{i+1}(v)) = l_i^\alpha(l_{i+1}(m_{i+1}(v))) = l_i^\alpha(m_{i+1}(v)) = l_i^\alpha(m_i^\alpha(v)) \), showing that \( l_i^\alpha(v) = l_i^\alpha(m_i^\alpha(v)) \) as desired. The case for \( r \) is symmetric.

**Lemma 3.43.** Let \( v \in V, \alpha \in \{0, 1\} \), and \( j_2[v] \leq i < j < d[v] \), then

\[
l_i^\alpha(v) = l_i^\alpha(m_j^\alpha(v)) \quad \land \quad r_i^\alpha(v) = r_i^\alpha(m_j^\alpha(v)) \quad \land \quad m_i^\alpha(v) = m_j^\alpha(m_j^\alpha(v))
\]

**Proof.** If \( m_j^\alpha(v) = v \) it is trivially true, so assume \( m_j^\alpha(v) \neq v \). Then \( j + 1 < d[v] \) and there is a \( k \), \( j \leq k < d[v] - 1 \) such that \( m_k^\alpha(v) = m_k^\alpha(v) \neq m_{k+1}^\alpha(v) \). Since \( k < d[v] - 1 \), \( k + 1 < d[v] \) and by the definition of \( m \) we must have either \( l_k^\alpha(v) \neq l_k^\alpha(l_{k+1}^\alpha(v)) \) or \( r_k^\alpha(v) \neq r_k^\alpha(r_{k+1}^\alpha(v)) \). Assume without loss of generality that \( l_k^\alpha(v) \neq l_k^\alpha(l_{k+1}^\alpha(v)) \). Then by lemma 3.39 and lemma 3.36 \( d[m_k^\alpha(v)] = k + 1 \) and \( l_k^\alpha(v) = l_k^\alpha(m_k^\alpha(v)) = l_k^\alpha(m_k^\alpha(v)) \) and \( r_k^\alpha(v) = r_k^\alpha(m_k^\alpha(v)) = r_k^\alpha(m_k^\alpha(v)) \). But then for any \( k' \) with \( j_2[v] \leq k' \leq k \), we have \( l_k^\alpha(v) = l_k^\alpha(m_k^\alpha(v)) = l_k^\alpha(m_k^\alpha(v)) \) and \( r_k^\alpha(v) = r_k^\alpha(m_k^\alpha(v)) = r_k^\alpha(m_k^\alpha(v)) \). From the definition of \( m \) we then get that for any \( k' \leq k \), \( m_k^\alpha(v) = m_{k'}^\alpha(m_{k'}^\alpha(v)) = m_{k'}^\alpha(m_{k'}^\alpha(v)) \), and since \( i \leq j \leq k \) we are done.

**Definition 3.44.** For any vertex \( v \in V, \alpha \in \{0, 1\} \) let

\[
M^\alpha[v] := \{ i | j_2[v] < i < d[v] \land m_{i-1}^\alpha(v) \neq m_i^\alpha(v) \}
\]

\[
p_m^\alpha[v] := \begin{cases} 
\bot & \text{if } M^\alpha[v] = \emptyset \\
\max M^\alpha[v] - 1 & \text{otherwise}
\end{cases}
\]

And define \( T_m^\alpha \) as the rooted forest over \( V \) whose parent pointers are \( p_m^\alpha \).
Theorem 3.45. There exists a practical RAM data structure that for any good st-decomposition of a graph with \( n \) vertices uses \( O(n) \) words of \( O(\log n) \) bits and can answer \( l_i^\alpha(v) \) and \( r_i^\alpha(v) \) queries in constant time.

Proof. For any vertex \( v \in V \), and \( \alpha \in \{0, 1\} \) let

\[
D_i^\alpha[v] := \{ i | v \text{ has a proper ancestor } w \text{ in } T_i^\alpha \text{ with } d[w] = i \}
\]

\[
D_r^\alpha[v] := \{ i | v \text{ has a proper ancestor } w \text{ in } T_r^\alpha \text{ with } d[w] = i \}
\]

Now, store levelancestors structures for each of \( T_i^\alpha \), \( T_r^\alpha \), and \( T_m^\alpha \), together with \( d[v], j_2[v], J_2[v], D_i^\alpha[v], D_r^\alpha[v], \) and \( M^\alpha[v] \) for each vertex. Since the height of the st-decomposition is \( O(\log n) \) each of \( J_2[v], D_i^\alpha[v], D_r^\alpha[v], \) and \( M^\alpha[v] \) can be represented in a single \( O(\log n) \)-bit word.

This representation allows us to find \( d[m_i^\alpha(v)] = \text{succ}(M^\alpha[v] \cup \{d[v]\}, i) \) in constant time, as well as computing the depth in \( T_m^\alpha \) of \( m_i^\alpha(v) \). Then using the levelancestors structure for \( T_m^\alpha \) we can compute \( m_i^\alpha(v) \) in constant time.

Similarly, this representation of the \( D_i^\alpha[v] \) set lets us compute the depth in \( T_i^\alpha \) of \( l_i^\alpha(v) \) in constant time, and with the levelancestors structure that lets us compute \( l_i^\alpha(v) \) in constant time. A symmetric argument shows that we can compute \( r_i^\alpha(v) \) in constant time.

Finally, lemma 3.42 says we can compute \( l_i^\alpha(v) \) and \( r_i^\alpha(v) \) in constant time given constant-time functions for \( l', r', \) and \( m \).

4 In-out graphs

For an in-out graph \( G \) we have a source, \( s \), that can reach all vertices of outdegree 0. Given such a source, \( s \), we may assign all vertices a colour: A vertex is green if it can be reached from \( s \), and red otherwise. We may also colour the directed edges: \((u, v)\) has the same colour as its endpoints, or is a blue edge in the special case where \( u \) is red and \( v \) is green. Our idea is to keep the colouring and flip all non-green edges, thus obtaining a single source graph \( H \) with source \( s \). (Any vertex was either green and thus already reachable from \( s \), or could reach some target \( t \), and is reachable from \( s \) in \( H \) via the first green vertex on its path to \( t \).)

Consider the single source reachability data structure for the red-green graph, \( H \). This alone does not suffice to determine reachability in \( G \), but it does when endowed with a few extra words per vertex:

M1 A red vertex must remember the additional information of the best green vertices on its own parent frame it can reach. There are at most 4 such vertices.

M2 Information about paths from a red to a green vertex in the same component.

M3 Information about paths from a red vertex in some component \( C \) to a green vertex in an ancestor component of \( C \).

Given a green vertex \( v \), we know for each ancestral frame segment the best vertex that can reach \( v \). For a red vertex \( u \), given a segment \( p \) on an ancestral frame to \( u \), we have information about the best vertex on \( p \) that may reach \( u \) in \( H \). If that best vertex is green, then there exists no path in \( G \) going through \( p \) from \( u \) to any other vertex. If that vertex is red, then it is the best vertex on \( p \) that \( u \) can reach.

We may now case reachability based on the colour of nodes:

- For green \( u \) and red \( v \), \( \text{reach}_{G}(u, v) = \text{No} \).
- For green vertices \( u, v \), \( \text{reach}_{G}(u, v) = \text{reach}_{H}(u, v) \)
- For red vertices \( u, v \), \( \text{reach}_{G}(u, v) = \text{reach}_{H}(v, u) \)
- When \( u \) is red and \( v \) is green, to determine \( \text{reach}_{G}(u, v) \) we need more work. It will depend on where in the hierarchy of components, \( u \) and \( v \) reside. Let \( C(x) \) denote the component of \( x \). Let \( C_1 \preceq C_2 \) denote that the component \( C_1 \) is an ancestor of \( C_2 \).
When \( u \) is red and \( v \) is green, there are the following cases:

1. \( C(u) = C(v) \):
   - Via a green vertex \( w \) in the parent frame of \( u \), reach\(_H(w,v)\). (See M1).
   - Staying within the frame, that is, reach\(_C(u,v)\). To handle this case we need to store more information, see Section 4.1.

2. \( C(u) \sim C(v) \):
   - Via a green vertex \( w \) in the parent frame of \( u \), reach\(_H(w,v)\). (See M1).
   - Via a green vertex \( w \), where \( C(w) = C(u) \), then reach\(_C(u,w)\) is in case 1 above. \( v \) knows at most 4 such \( ws \) from the single source structure.

3. \( C(u) \succ C(v) \):
   - Via a red edge \((w',w)\) with \( C(w') \preceq C(v) \prec C(w) \prec C(u) \), then reach\(_C(w',v)\) is in case 1 or 2 above. (When \( u \)'s best vertex on a disegment of \( C(v) \)'s frame is red.)
   - Via a blue edge \((w',w)\) with \( C(w') \preceq C(v) \prec C(w') \prec C(u) \). We handle this case in Section 4.1.

4. \( C(u), C(v) \succ N \), where \( N = nca(C(u), C(v)) \).
   - Via \( w \), \( C(w) \preceq N \), then reach\(_C(u,w)\) is in case 3 above. \( v \) computes at most 4 such \( ws \) from the single source structure, and note that all the vertices that \( v \) computes must be green.

### 4.1 Intracomponental blue edges

Consider the set of "blue" edges \((a,b)\) from \( G \) where both the red vertex \( a \) and green \( b \) reside in some given component in the s-t-decomposition of \( H \).

**Lemma 4.1.** We may assign to each vertex \( \leq 2 \) numbers, such that if red \( u \) remembers \( i, j \in \mathbb{N} \) and green \( v \) remembers \( l, r \in \mathbb{N} \), then \( u \) can reach \( v \) if and only if \( i \leq l \leq j \), or \( i \leq r \leq j \).

**Proof.** The key observation is that we may enumerate all blue edges \( b_0 = (u_0, v_0), \ldots b_i = (u_m, v_m) \) such that any red vertex can reach a segment of their endpoints, \( v_i, \ldots, v_j \). Namely, the blue edges form a minimal cut in the planar graph which separates the red from the green vertices, and this cut induces a cyclic order. In this order, each red vertex may reach a segment of blue edges, and each green vertex may reach a segment of blue edge endpoints. Thus, the blue edge endpoints reachable from a given red vertex (through any path) is a union of overlapping segments, which is again a segment.

Now each red vertex remembers the indices of the first \( v_i \) and last \( v_j \) blue edge endpoint it may reach. For a green vertex \( v \), the s-t-subgraph with \( v \) as target has a delimiting face consisting of two paths, \( p_l \) and \( p_r \). \( v \) remembers the indices of the last blue edge endpoints on \( p_l \) and \( p_r \), number \( b_l \) and \( b_r \), respectively, if they exist.

Clearly, if \( b_l \) or \( b_r \) is within range, \( u \) may reach \( v \). Contrarily, if \( u \) may reach \( v \), it must do so via some vertex \( v' \) on \( p_l \cup p_r \). But \( v' \) must be able to reach \( v_l \) or \( v_r \), and thus, \( l \) or \( r \) is within range. \( \square \)

### 4.2 Intercomponental blue edges

For any red vertex \( u \), for any level, for any of the at most four directed frame segments, there is a best green vertex reachable from \( u \) by a path ending in a blue edge. We denote this vertex \( \lambda^0_i(u) = L^0_i(u) \) or \( \rho^0_i(u) = R^0_i(u) \), with \( \alpha \in \{0,1\} \). For any level, \( i \), if \( \lambda^0_i(u) \neq \lambda^0_{i-1}(u) \), then there exists a red vertex \( u' \in F_{x_{i+1}}(u) \) which can reach \( \lambda^0_{i-1}(u) \). But if some \( u \)-reachable red vertex on the frame \( F \) can reach \( \lambda \), then at least one of the \( \leq 4 \) best \( u \)-reachable red vertices on \( F \) can reach \( \lambda \).
Lemma 4.2. Using constant space and query time, for each red vertex \( u \) at level \( k \), and given \( j < k \), we may find the best four green vertices at level \( \leq j \) reachable from \( u \) via a path ending in a blue edge.

Proof. For each red vertex, associate four bitmaps, \( \Lambda^\alpha, P^\alpha \), where the \( i \)'th bit answers whether \( \lambda_i^\alpha = \lambda_{i-1}^\alpha \). For each bitmap, say, \( \Lambda^0 \), find the last bit before \( j \) set to 1, corresponding to some level \( l \) with \( l \leq j \). Then we know that \( \lambda_0^0(u) \) is reachable from one of the best \( \leq 4 \) red vertices on \( F_{l+1}(u) \), say, \( u^\dagger \). This happens in such a way that \( \lambda_0^0(u) \) belongs to the parent frame of \( u^\dagger \). But then, we may simply let each red vertex remember the \( \leq 4 \) best green vertices on their parent frame that they can reach.

References
[1] Stephen Alstrup, Jens Peter Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *IPL*, 64:64–4, 1997.
[2] S. Arikati, D.Z. Chen, L.P. Chew, G. Das, M. Smid, and C.D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *ESA ’96*, pages 514–528, 1996.
[3] D.Z. Chen and J. Xu. Shortest path queries in planar graphs. In *STOC ’00*, pages 469–478, 2000.
[4] H. Djidjev. Efficient algorithms for shortest path queries in planar digraphs. In *WG ’96*, pages 151–165, 1996.
[5] H. Djidjev, G. Panziou, and C. Zaroliagis. Computing shortest paths and distances in planar graphs. In *ICALP ’91*, pages 327–339, 1991.
[6] H. Djidjev, G. Panziou, and C. Zaroliagis. Fast algorithms for maintaining shortest paths in outerplanar and planar digraphs. In *FCT ’95*, pages 191–200, 1995.
[7] K. Kawarabayashi, P.N. Klein, and C. Sommer. Linear-space approximate distance oracles for planar, bounded-genus, and minor-free graphs. In *ALP ’11*, pages 135–146, 2011.
[8] K. Kawarabayashi, C. Sommer, and M. Thorup. More compact oracles for approximate distances in undirected planar graphs. In *SODA ’13*, pages 550–563, 2013.
[9] B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice Hall, 2nd edition, 1988.
[10] P. Klein. Preprocessing an undirected planar network to enable fast approximate distance queries. In *SODA ’02*, pages 820–827, 2002.
[11] S. Mozes and C. Sommer. Exact distance oracles for planar graphs. In *SODA ’12*, pages 209–222, 2012.
[12] M. Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40(3):827–847, 2011. Announced at FOCS’08. See also arXiv:1010.3783.
[13] R. Tamassia and I.G. Tollis. Dynamic reachability in planar digraphs with one source and one sink. *Theor. Comput. Sci.*, 119(2):331–343, 1993.
[14] R. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1972.
[15] M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, 2004.
[16] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005. Announced at STOC’01.