Reachability Substitutes for Planar Digraphs

Irit Katriel[1], Martin Kutz[1], and Martin Skutella[2]

[1] Max-Planck-Institut für Informatik Saarbrücken, Germany
{irit,mkutz}@mpi-sb.mpg.de
[2] Universität Dortmund, Germany
martin.skutella@uni-dortmund.de

April 7, 2005

Abstract. Connectivity and reachability are fundamental concepts in the study of graphs and graph algorithms. For directed graphs, finding compressed representations of reachability relations among certain vertices constitutes an important basic problem. Given a digraph $G = (V, E)$ with a set U of vertices marked "interesting," we want to find a smaller digraph $\mathcal{H} = (V', E')$ with $V' \supseteq U$ in such a way that the reachabilities amongst those interesting vertices in G and $\mathcal{H}$ are the same. So with respect to the reachability relations within U, the digraph $\mathcal{H}$ is a substitute for G. This problem has applications, for example, in network design and also as an important building block in the context of dynamic reachability.

The complexity of finding such a *reachability substitute* of minimum size has so far been open. We prove that the problem is NP-hard. The main contribution of the paper is about planar digraphs: While almost all digraphs do not allow reachability substitutes smaller than $\Omega(|U|^2/\log|U|)$, every planar digraph has a reachability substitute of size $\mathcal{O}(|U|\log^2|U|)$. Our proof is constructive and offers an efficient algorithm to find a reachability substitute of this size.

Our results constitute the first progress on this fundamental problem in more than a decade. The upper bound for the planar case rests on two new structural results for planar dags, a separation procedure and a reachability theorem. We believe that these contributions are of independent interest and might be useful in the study of other related problems.

1 Introduction

We consider the following network design problem. Let $G = (V, E)$ be a directed graph with $n = |V|$ nodes and let $U \subseteq V$ be a set of κ nodes in G, which are designated *interesting*. A *reachability substitute (RS) for* (G, U) is a digraph $\mathcal{H} = (V', E')$ such that $U \subseteq V'$ and for any two interesting nodes $u, v \in U$, there is a path from u to v in $\mathcal{H}$ iff there is a path from u to v in G. Of course, G is an RS for (G, U) for any $U \subseteq V$. The problem we are interested in is that of finding a small RS, i.e., one that minimizes $|V'| + |E'|$. We will show that this problem is NP-hard.

Aside from the complexity of finding a small RS, we are mainly interested in the structural question of lower and upper bounds on its size. Trivially, every input has a solution of size $\mathcal{O}(\kappa^2)$, and we show by a counting argument that in general it is not possible to obtain a considerably smaller RS: Almost all inputs have only RSs of size $\Omega(\kappa^2/\log\kappa)$.

The main focus of this paper is on planar inputs—the output RS need not be planar, though. Subramanian [25] showed that if all nodes of U lie on a constant number of faces of a planar embedding, then there is a solution of size $\mathcal{O}(\kappa\log\kappa)$, which can be found in $\mathcal{O}(n\log n)$ time. This algorithm was designed as a component of an algorithm for dynamic reachability in planar graphs; the graph is recursively partitioned with small separators, where the separator nodes become the interesting nodes. Eppstein et al. [15] later generalized this approach and obtained a faster algorithm. In addition, Klein and Subramanian [20] showed that the algorithm of [25] can be modified to construct a substitute graph that not only represents the reachabilities among the interesting nodes but also approximates the lengths of shortest paths between them.

The main result of this paper is that planar inputs have small RSs, even if we remove the restriction that the interesting nodes lie on a constant number of faces. This is the first progress on this fundamental problem in more than a decade.

Theorem 1. *Any planar graph $G = (V, E)$ with a subset $U \subseteq V$ of κ "interesting" nodes has a reachability substitute of size $\mathcal{O}(\kappa \log^2 \kappa)$.*

Our proof is constructive and offers an efficient algorithm to find the RS. The result is particularly interesting in view of the lower bound $\Omega(\kappa^2/\log \kappa)$ on the size of RSs for (almost all) general digraphs. It substantiates the general intuition according to which the structure of reachability relations amongst (interesting) vertices in planar digraphs is considerably less complex than in general digraphs.

Besides its intrinsic value as a structural result, Theorem 1 has applications in network design and graph algorithms. For example, one consequence of our result is the following. For a given planar digraph with κ interesting nodes, there exists a reachability oracle that has access to a database of size $\mathcal{O}(\kappa \log^2 \kappa)$ only and can answer queries about directed paths between pairs of interesting vertices in $\mathcal{O}(\kappa \log^2 \kappa)$ time.

Further related work. The special case $U = V$, where all nodes of the input are considered interesting, has actually received more attention than the general RS setting studied in [15, 20, 25].

If we require $V' = V$ and $E' \subseteq E$, the problem is called *Minimum Equivalent Graph (MEG)*. It is NP-hard and can be approximated within a constant factor in polynomial time [19]. If only $V' = V$ is demanded while E' is allowed to contain arcs that are not in E, the problem can be solved in polynomial time [1, 18, 19]: Connect the nodes of each strongly connected component (SCC) by a simple cycle and compute the transitive reduction of the dag obtained by contracting each SCC into a single node. Note that the RS problem treated in this paper, besides allowing $U \subset V$, removes both requirements of MEG: V' does not need to be contained in V and E' does not need to be contained in E.

The case $U = V$ is strongly related to reachability, clearly one of the most fundamental algorithmic graph problems. The static problem is to find the transitive closure, while in the dynamic version we need to efficiently maintain a data structure that can answer reachability queries [16, 24, 26] or, in a widely studied special case, the explicit transitive closure matrix [3, 11, 22]. A substitute graph which not only represents the existence of paths but also approximates their lengths is called a *spanner*. Several spanner constructions, with different size and approximation guarantees, are known, though most of the existing work on spanners is about undirected graphs [2, 4, 5, 7, 8, 12–14, 21, 23, 29].

Recently, Coppersmith and Elkin [10] introduced a new variant of spanners which is more closely related to the problem considered here. One special case investigated in [10] is the following: Given an (undirected) graph together with a subset of interesting vertices, the task is to find a small subgraph with the same distances between interesting vertices as in the original graph. It is shown in [10] that there always exists such a subgraph with $\mathcal{O}(n)$ edges if the number of interesting vertices is $\mathcal{O}(n^{1/4})$.

Another related problem is that of computing a *distance oracle* [6, 27, 28], i.e., a representation of the reachability relation of a given graph which supports (exact or approximate) distance queries efficiently. An example of such labeling schemes are so-called 2-hop labels, studied by Cohen et al. [9], where again the whole node set is considered interesting.

Our techniques. The main result stated in Theorem 1 builds upon three components. The first is a digraph construction we call an *interval structure*. Given a linearly ordered set S of ℓ nodes, an interval structure for S is a graph of size $\mathcal{O}(\ell \log \ell)$ containing S such that for every length-2^i subinterval I of S, there is a node in the graph that reaches exactly the nodes in I. This simple data structure has several nice properties on which we shall elaborate later.

The second component is a separation procedure for planar dags, which is of independent interest. We show that, given a plane dag (i.e., a planar dag with a planar embedding) with a weight for each node, it is always possible to find a balanced almost-directed cut. That is, a simple closed curve which does not go through any node and crosses each arc at most once; and which partitions the nodes into two sets A and B of balanced weight. Almost-directed means that almost all arcs which cross the curve are directed from A to B.

The third component is the algorithm that, given a plane dag G and a simple almost-directed cut, constructs a dag $\mathcal{H}$ that represents all paths from interesting nodes in A to interesting nodes in B. This construction uses the interval structure. $\mathcal{H}$ is the union of three parts: The first represents paths from interesting nodes in A to the separating curve, the second represents paths from the curve to interesting nodes in B, and the third links these structures in the right way. We prove an upper bound of $\mathcal{O}(\kappa \log \kappa)$ on the size of $\mathcal{H}$. One step of this proof establishes that the reachability from interesting nodes in A to the separating curve cannot be very complicated. More precisely, let $e = (a, b)$ be a cross-cut arc and let its *type* be the set of interesting nodes in A that can reach its tail a. We show that the number of types on the separating curve is $\mathcal{O}(\kappa)$ and, furthermore, that while walking along the curve the type can only change $\mathcal{O}(\kappa)$ times. This result is interesting in its own.

Once we have the substitute graph $\mathcal{H}$, we recursively compute an RS for A and an RS for B. Now, all paths between interesting nodes are represented by a graph with total size $\mathcal{O}(\kappa \log^2 \kappa)$.

Road Map. The rest of this paper is organized as follows. In Section 2 we derive several complexity results about the computational cost of computing small RSs and lower bounds on their sizes. Sections 3 to 5 form the bulk of the paper and describe the construction of RSs for planar acyclic graphs: In Section 3 we assume that we have a simple directed cut and show how to represent the reachabilities from each side to the cutting curve. In Section 4 we show how to find a simple almost-directed cut and in Section 5 we describe how to combine the different parts of the algorithm and show how to handle the fact that the cut may not be perfectly directed. Finally, in Section 6 we show how to extend the algorithm to handle cyclic graphs.

2 The Complexity of Reachability Substitutes

In order to prove that the optimization problem under consideration is NP-hard, we first consider a more general problem.

2.1 NP-hardness of a more general problem

Consider the following problem.
Input: A digraph $G = (V, E_s \dot\cup E_d)$ with solid arcs (E_s) and dashed arcs (E_d).
Output: A digraph $H = (V', E)$ on a node set $V' \supseteq V$ such that there is a path from u to v in H for each pair $(u, v) \in E_s$ and there is no path from u to v in H unless $(u, v) \in E_s \cup E_d$. In other words, solid arcs indicate mandatory reachabilities, while dashed arcs specify optional reachabilities.
Objective: Minimize $|V'| + |E|$. (Note that this problem may not have a solution, namely when the input already violates transitivity.)

Theorem 2. *The problem above is NP-hard.*

Proof. By reduction from Minimum Hitting Set, which is the following problem.
Input: A collection C of subsets of a finite set S.
Output: A hitting set for C, i.e., a subset S' of S such that S' contains at least one element from each subset in C.
Objective: Minimize $|S'|$.

The reduction is as follows. V contains a node for every item in S, for every subset in C, and a special node x, that is, $V = S \cup C \cup \{x\}$. The arcs are as follows:

- For each $u \in S$ and $v \in C$, let $(u, v) \in E_s$ if $u \in v$;
- let $(x, u) \in E_d$ for each $u \in S$; and
- let $(x, u) \in E_s$ for each $u \in C$.

Given a solution to this problem, we construct a solution to the minimum hitting set problem as follows: for each arc (x, u) leaving x, if $u \in S$ then add u to the hitting set, if $u \in C$ then add an arbitrary element from u to the hitting set. If u is a dummy node that was introduced by the algorithm, then it is reachable from some nodes in S (otherwise the solution is not optimal). Select one of them for the hitting set.

One can easily verify that every set in C is represented in the hitting set. To see that the hitting set is optimal, note that the cardinality of the hitting set is equal to the degree of x in the RS. If there is a smaller hitting set, then we can obtain a smaller RS by removing all arcs incident to x and connecting x to the node representing each element of the optimal hitting set, contradicting our assumption that the RS is optimal.

2.2 NP-hardness of our problem

Multiply each node representing an item by four. That is, for every item $u \in S$, U contains four nodes, $u_1, \ldots, u_4$. The arcs are now as follows:

- For each $u \in S$ and $v \in C$, if $u \in v$ then $(u_i, v) \in E_s$ for all $1 \leq i \leq 4$.
- For each $u \in C$, $(x, u) \in E_s$.

Proposition 1. *An optimal $\mathcal{H}$ contains a node $\overline{u}$ for each item $u \in S$ such that $\overline{u}$ is reachable from each of $u_1, \ldots, u_4$ and reaches every node that represents a set in C that contains u.*

It is easy to see that otherwise, $\mathcal{H}$ is not optimal. Hence, in an optimal $\mathcal{H}$ there will be an arc from x to $\overline{u}$ if in the previous reduction there was an arc from x to u. This implies that finding an optimal RS is NP-hard.

2.3 The counting argument

We use a counting argument in order to show that there are relations that cannot be represented by a graph of size less than $\Omega(\kappa^2/\log\kappa)$.

Let $\kappa = 2k$ for some integer k and consider as possible inputs all labeled bipartite graphs with k (interesting) nodes on each side of the bipartition, with all arcs going in the same direction. There are exactly 2^{k^2} such input graphs and no two of them induce the same reachability relation. We use this as a lower bound on the total number of inputs.

On the other hand, we upper bound the number $N(\ell)$ of different reachability substitutes of size at most ℓ. Obviously, $N(\ell)$ is smaller than the number of different digraphs on at most ℓ nodes with at most ℓ arcs. The latter quantity can be bounded as follows: For any arc, there are less than ℓ^2 possibilities of how to place it in (or omit it from) the digraph. As a consequence, the number of digraphs, and thus $N(\ell)$, is bounded by $\ell^{2\ell} = 2^{2\ell \log \ell}$. Therefore, only a fraction of $2^{2\ell \log \ell - k^2}$ of all inputs can have an RS of size at most ℓ. This fraction can only be constant if $\ell \in \Omega(k^2/\log k)$.

Lemma 1. *Almost all digraphs on κ interesting nodes only have RSs of size $\Omega(\kappa^2/\log \kappa)$.*

2.4 A lower bound for planar outputs

Throughout this exposition, we assume that a planar instance $(G = (V, E), U \subseteq V)$ of the RS problem comes with a fixed embedding of G into the plane. Nodes correspond to points and arcs are embodied as simple curves who may only intersect at nodes. If not given, such a representation can be constructed in linear time [17].

The almost linear-size RS for planar graphs that we construct for Theorem 1 will, however, be far from planar. To see that this cannot be avoided, we briefly argue why in general, planarity must be sacrificed if one wants small RSs.

Consider the plane dag in Figure 1. The paths through the black uninteresting nodes are set up to make a lower interesting node v_j reachable from some upper interesting node u_i if and only if $i \leq j < i + r$. We claim that any planar RS for this dag with the white nodes marked interesting, must contain essentially all those black intermediate nodes, too. In other words, this dag is incompressible if planarity is to be maintained. With $r \approx \ell$ this gives a quadratic lower bound on the representation size in terms of κ.

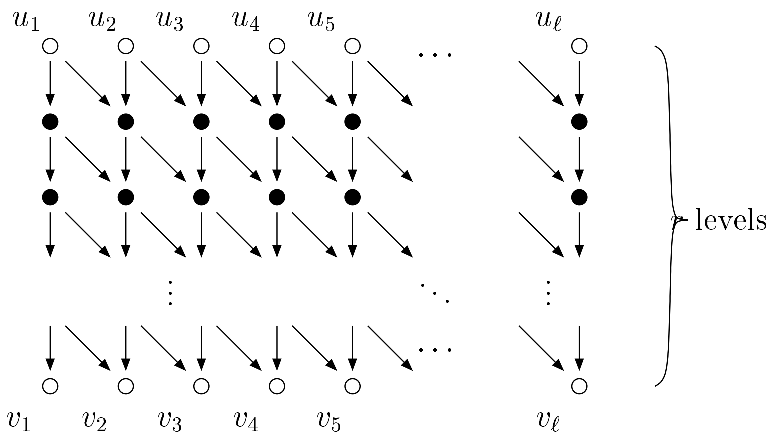

Fig. 1. A planar dag with planar RSs of quadratic size only.

To prove this claim, consider a pair $u_i, u_{i'}$ with $i \leq i' < i+r$. The set of bottom nodes reachable from both, forms an interval $\{v_{i'}, \ldots, v_{i+r-1}\}$ of length $r + i - i'$. In an RS for this dag, there must be a path P from u_i to v_{i+r-1} and a path P' from $u_{i'}$ to $v_{i'}$. By some additional framework nodes and arcs, which we omitted for simplicity, we can guarantee that also in the output, the u_i and the v_i come in ordered lines and that any connecting paths run in the area between them. Hence, P and P' will have to intersect at some node x. Under these side conditions, it is easy to see that the set of v_j's reachable from this x is exactly the above length-$(r + i - i')$ interval. Hence, we must have a different node for each such interval, which yields our claim.

3 Crossing a Line

The rest of the paper contains the proof of Theorem 1. Initially, we assume that the input is a dag. In Section 6 we show how to handle cycles.

Assume we can split the plane along a simple closed curve $\mathcal{L}$ such that the two resulting areas A and B contain about the same number of interesting nodes and such that cross-cut arcs of G go only in one direction, from A to B. In this section we show how to represent all cross connections from interesting nodes in A to such in B with a graph of size $\mathcal{O}(\kappa \log \kappa)$. Let's make the concept of "splitting" more precise.

Definition 1. *A* simple cut *of a plane digraph (or graph) $G = (V, E)$ is a partition $(A, B = V \setminus A)$ of the nodes such that there exists a Jordan curve $\mathcal{L}$ (a simple closed curve, that is) separating the plane into two areas such that one contains all nodes in A and the other all in B, and such that $\mathcal{L}$ contains no node and crosses no arc of G more than once. A simple cut is* directed *(from A to B) if no arc goes from B to A.*

We call the involved sets A and B simple sets *and call A* out-directed *and B* in-directed *in case of a directed cut.*

The condition on arc-curve crossings in Definition 1 is needed to ensure that the cut conforms with the embedding. If it was not there, the separating curve could cut out just any set we like.

It will turn out that obtaining a directed simple cut that is also reasonably balanced is difficult and sometimes not even possible. However, we shall deal with those problems later. For now, let us assume we have such a cut. The construction behind the following theorem forms the main ingredient for Theorem 1.

Theorem 3. *Consider a plane dag with a directed simple cut (A, B) and with a total of κ nodes marked interesting. Then there exists a digraph of size $\mathcal{O}(\kappa \log \kappa)$ containing all those interesting nodes, who represents all reachabilities from interesting nodes in A to interesting nodes in B and no further connections amongst interesting nodes.*

3.1 The type bound

The proof of Theorem 3 treats the two parts A and B independently. We first account for all interesting reachabilities from A to the separating curve $\mathcal{L}$, then for the reachabilities from $\mathcal{L}$ into B, and afterwards merge the two structures into one RS for the whole problem.

For technical convenience, we introduce an *interface node* x_e for each arc $e = (u, w)$ that crosses $\mathcal{L}$ and replace e by the two "halves" (u, x_e) and (x_e, w). After that, $\mathcal{L}$ meets the graph only at these interface nodes (none of which are interesting) and cuts no arcs any more. Denote the set of interface nodes by X.

Beginning our analysis in A, we ignore the other side, B, completely for a while. Hence, the outdegree of all interface nodes is (effectively) zero for the time being. For convenience, we split the cutting curve $\mathcal{L}$ at an arbitrary point that is not a node and stretch it out into a straight horizontal line, so that the A-portion of G now sits completely above this *baseline*.

All interface nodes come in a natural order now so we can label them x_1 through x_t (from left to right) and as they sit on the baseline, we also like to call them *ground nodes* in this section. Each of them carries a *type*: the set of interesting nodes in A that reach this ground node. We shall see that neighboring nodes of identical type can be treated like a single node in our representation, so we are interested in the number of *type changes* along $\mathcal{L}$: pairs $(i-1, i)$ where the types of x_{i-1} and x_i differ. (We consider $(0, 1)$ a type change, too.)

The following observation gives a linear bound on the complexity of the baseline.

Proposition 2. *With ℓ interesting nodes above the baseline, there are at most $4\ell - 3$ type changes on the baseline.*

Our proof of Proposition 2 rests on an inductive reconstruction procedure for the dag, subject to some special rules. We define the *net* of an interesting node a of the plane dag to be the set of all arcs and nodes which are reachable from a and reach the baseline. The *shadow* of a is the region of all points in the halfplane that are completely enclosed by arcs from a's net and the baseline, together with all points of the net itself (see Figure 2). Here are the insertion rules.

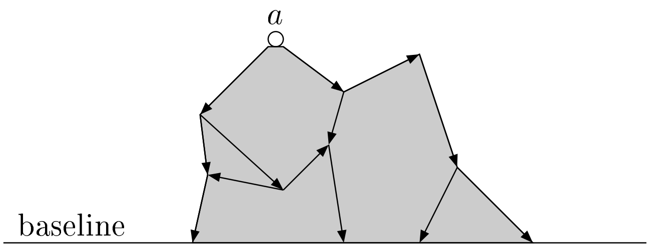

Fig. 2. The net and shadow of a node.

Rule 1. Exactly one interesting node is inserted at a time and with it all arcs and nodes from its net but no more.
Rule 2. An interesting node must be inserted before all other interesting nodes in its shadow.

This process starts with an empty dag above an empty baseline. The first rule guarantees that after the insertion of an interesting node a, its influence on the baseline will never again change. New ground nodes may be created afterwards but none of them will carry an a in its type. The motivation for the second rule is more complicated and shall become clear throughout the proofs of the two subsequent lemmas.

Lemma 2. *Immediately after a new interesting node a has been inserted, the ground nodes that carry a in their type form a contiguous interval on the baseline.*

Proof. Immediately after insertion of a, all nodes in the shadow of a must carry a in their type because if some node x inside the shadow didn't, it would by Rule 1 have to be reachable by some other node, b, say. Such a b must, by Rule 2, lie outside the shadow of a. But then any path from b to x would have to cross a's net and would thus, by planarity, also carry the label a into the type of x. A contradiction. See Figure 3. □

Lemma 3. *Immediately after the insertion of a new interesting node a, the ground nodes that have been created through this insertion are of type $\{a\}$ and they form a contiguous interval on the baseline.*

Proof. The first part is an immediate implication of Rule 1. If a node carries any further label b, it must have existed already at the creation time of node b. To see that the ground nodes with type $\{a\}$ form an interval, assume for contradiction that there was some node of type $\{c, \ldots\}$ between two $\{a\}$-nodes. Then the interesting node c would have to lie in the shadow of a; a contradiction to Rule 2. □

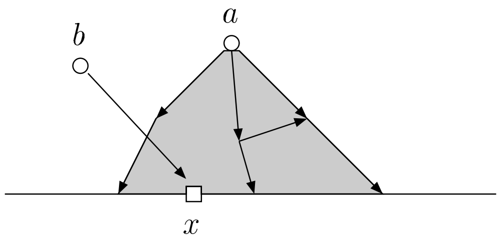

Fig. 3. All types in the shadow of a new a receive that a.

Proof (of Proposition 2). By induction. We keep count of the number of type changes along the baseline while new interesting nodes are inserted according to the two rules.

After the first interesting node, there is exactly one type on the baseline which means just one type change, the initial one. When a new interesting node a is added, new type changes may appear in at most four places: at the two ends of the interval of a (by Lemma 2) and at the ends of the interval of new type-$\{a\}$ ground nodes (by Lemma 3). Note that we do not claim that only four new types are created—that would not be true—but that there are no more than four new positions on the baseline where the type changes. □

3.2 Representing intervals

Lemmas 2 and 3 tell us that if we follow Rules 1 and 2, changes along the baseline occur along contiguous intervals. Our RS construction rests on the following structure for representing reachabilities of intervals.

Consider a digraph $\mathcal{H}$ (which will be our RS) with a designated subset X of nodes (corresponding to the ground nodes). We say that a node set $Z \subseteq X$ is *represented* by $\mathcal{H}$ (w.r.t. X) if there exists a node u_Z in $\mathcal{H}$ such that $\{$nodes reachable from $u_Z\} \cap X = Z$. Assume further that the set Z is endowed with a linear order, i.e., Z is an "interval." We say that $\mathcal{H}$ has an *interval structure for* Z (w.r.t. X) if every contiguous subinterval of length 2^i of Z is represented by $\mathcal{H}$ for all integer $i \geq 0$. The proof of the following two lemmas should make the concept clear.

Lemma 4. *For an interval of length ℓ there exists an interval structure of size $\mathcal{O}(\ell \log \ell)$.*

Proof. We need $\lfloor \log \ell \rfloor$ levels, each responsible for the subintervals of one common length 2^i. There are less than ℓ intervals on each level, and each length-2^i interval can be realized by one node and two arcs pointing to two level-2^{i-1} intervals. □

Assume that $\mathcal{H}$ contains an interval structure for an interval Z, and that we wish to represent some subinterval Z' of Z whose length ℓ' is not a power of 2. Since Z' is the union of two intervals of length $2^{\lfloor \log \ell' \rfloor}$, we can do this by adding just one node and two arcs to $\mathcal{H}$.

Lemma 5. *If a digraph R already contains interval structures for two intervals $\{x_1, \ldots, x_t\}$ and $\{x'_1, \ldots, x'_{t'}\}$, then $\mathcal{O}(\min\{t, t'\} \log(t + t'))$ additional nodes and arcs suffice to extend them to an interval structure of the concatenation $\{x_1, \ldots, x_t, x'_1, \ldots, x'_{t'}\}$. The original interval structures are not destroyed by this extension.*

Proof. We only have to represent subintervals that extend over both domains. The others are already covered by the interval structures for the given sets. Any such subinterval begins on some x_i and ends at some x'_j. It can be represented as the union of two intervals on the left and two further intervals on the right, which can be realized by one new node and four arcs. Charging each new interval to its endpoint in the smaller side, we see that there are no more than $\min\{t, t'\}$ new intervals on each level $i \leq \lfloor \log(t + t') \rfloor$, which gives the claimed bound. □

3.3 Constructing the RS for one halfplane

We now construct an RS $\mathcal{H}$ that encodes all reachabilities from interesting nodes in the upper halfplane to the ground nodes. Since we shall soon see that adjacent nodes of identical type can be treated jointly, we delete adjacent duplicates so that by Proposition 2, we are left with a sequence $x_1, \ldots, x_t$ of ground nodes whose length t is linear in the number of interesting nodes.

We follow the reconstruction from the proof of Proposition 2 again, starting with an empty dag and extending it one interesting node after another by Rules 1 and 2. (Only with duplicate ground nodes removed from the process now.) The RS $\mathcal{H}$ for our dag is constructed simultaneously, also starting as an empty dag.

Case a. If a new interesting node a does not introduce new ground nodes (i.e., no new singleton type $\{a\}$ is created) then the set of ground nodes with a in their type (which forms a contiguous interval by Lemma 2) can be covered by four intervals from the interval structures in $\mathcal{H}$. (This will be shown in Lemma 7 below.) We create a new node in $\mathcal{H}$ and link it to these four intervals. This node will be only there for the current interesting node a and not be reused in the construction of $\mathcal{H}$ later.

Case b. If a new interesting node a does introduce new ground nodes, the set N of type-$\{a\}$ ground nodes forms a contiguous interval by Lemma 3. Using Lemma 4 we create a new interval structure of size $\mathcal{O}(|N| \log |N|)$ for N. By Lemma 2, the set of all ground nodes with a in their type forms a super interval of N. Denote the portion to the left of N by L and that to the right by R. (See the dashed regions in the left drawing of Figure 4.) We shall see (in Lemma 6 below) that the dag $\mathcal{H}$ already contains interval structures for L and R. With the help of Lemma 5 we can thus create merged interval structures for the unions $L \cup N$ and $N \cup R$, which costs $\mathcal{O}(|N| \log \kappa)$ (using κ as a generous bound on $|L| + |N|$ and $|N| + |R|$). Afterwards, we represent the whole interval $L \cup N \cup R$ with one new node and four arcs, just like in Case 1.

Note that in Case b we do not create an interval structure for the whole section $L \cup N \cup R$. That would turn out too costly. Instead we provide two independent structures that overlap on N. We now show that the interval structures required in both cases do always exist as claimed. Afterwards we upper bound the total size of $\mathcal{H}$. We begin with the more difficult case.

Lemma 6. *Before a Case b insertion, $\mathcal{H}$ contains an interval structure for each of L and R.*

Proof. Formally, the proof is by induction, assuming that at each step the current $\mathcal{H}$ has been constructed according to the above rules. The induction base is trivial with empty L and R for the first interesting vertex to be inserted. We do the induction step only for L, concluding for R by symmetry.

Consider the rightmost ground node y in L and let its type at creation time be $\{b\}$. We claim that y was amongst the last ground nodes created in L. By Rule 2, b must lie outside the shadow of the new node a. Take any path P_b from b to y and a path P_a from a to the leftmost ground node x in L. These paths must cross in some node r, as shown in Figure 4. (Here we use that a creates new ground nodes which lie to the right of y.)

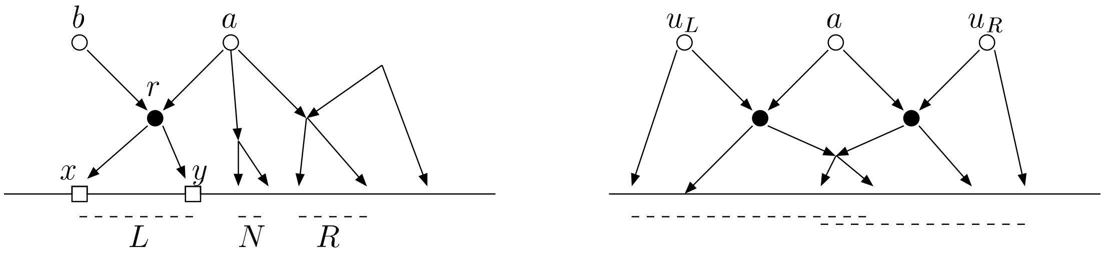

Fig. 4. In Case a (left), the interval L already has an interval structure; and in Case b (right), we get a subinterval of the union of two interval structures.

The shadow of r covers all of L and hence, the interesting node b reaches all of L. This means that, by Rule 1, no nodes in L were introduced after b and y were inserted, which is what we claimed.

Denote the three ground intervals involved in the appearance of the interesting node b by L_b, N_b, and R_b. We know by induction that we have created an interval structure for the unions $L_b \cup N_b$ and $N_b \cup R_b$ then. Since by definition, y lies in N_b, the above claim implies that $L \subseteq L_b \cup N_b$. Hence, we already have an interval structure for L in $\mathcal{H}$, as was to be shown. □

Lemma 7. *In Case a, the set of ground nodes with the new node a in their type can be covered with no more than four intervals represented in* $\mathcal{H}$.

Proof. Traverse the leftmost path in the net of a until we meet the first node that was created in an iteration of type b and denote the respective interesting node by u_L. Likewise, follow the right flank to the first such node there, created by an interesting node u_R. (Note that $u_L = u_R$ is well possible.)

Clearly, all ground nodes reachable from a are reachable from u_L or u_R, too, and since they form an interval by Lemma 2, this interval can be written as the union of two subintervals from the structures for u_L and two from that of u_R. □

3.4 Merging all interval structures

We have everything in place to conclude the proof of Theorem 3 about reachabilities across a directed simple cut.

Proof (of Theorem 3). First of all, we need to verify that the previous construction is of size $\mathcal{O}(\kappa \log \kappa)$. This is not difficult. An application of Case a produces only a constant number of arcs and nodes which accumulates to $\mathcal{O}(\kappa)$ for all interesting nodes. In Case b, we charge the cost for the new interval structure and for the two subsequent merges to the newly created ground nodes, which sums to $\mathcal{O}(\kappa \log \kappa)$ as desired.

We repeat the whole construction for the other side B, with everything directed from the baseline into B this time, so that we have two digraphs $\mathcal{H}_A$ and $\mathcal{H}_B$ for the two parts. It only remains to merge them at the baseline. Therefore, recall that we have deleted type multiplicities on the baselines for A and B. Ground nodes of A might not have pendants in B and vice-versa. This has to be taken into account again now—in the obvious way: add an arc from a ground node a of A to a ground node b of B iff the intersection of the equivalence classes represented by them is not empty. The number of such connections is easily seen to be linear in κ by a straight-forward type-change counting argument, again. □

4 Balanced Cuts

In order to apply Theorem 3 efficiently in a recursive procedure, we would like to find directed simple cuts in plane dags (as in Definition 1) that yield balanced partitions with respect to the number of interesting nodes on each side. Unfortunately, this is not always possible.

Figure 5 shows a plane dag whose 4 interesting nodes form a star around a central node with some further in-between arcs which separate the interesting nodes. With this dag, we face the problem that as soon as we put two different interesting nodes into A, the outward arcs force also the center into A, but then A must contain *all* interesting nodes. Therefore, this dag does not have a balanced directed simple cut. (It is not hard to see how to extend the dag to ensure that this property is maintained in any possible plane embedding.) The obvious n-petal generalization of this figure does not have directed simple cuts better balanced than $1 : n-1$. So the degree of imbalance may get arbitrarily bad.

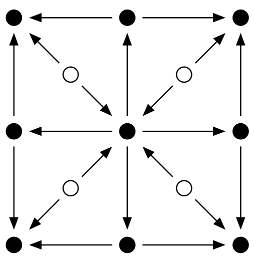

Fig. 5. A plane dag without a balanced directed cut (filled dots representing non-interesting nodes).

Our remedy to this problem is to settle for cuts with slightly weaker properties. The key is that annoying center node in Figure 5.

Definition 2. *A simple cut (A, B) of a plane digraph G is* almost directed *if the deletion of at most one node of G together with all incident arcs turns it into a directed simple cut.*

Assume the dag is endowed with an additive weight function $\mu: 2^V \to \mathbb{R}$. Then a simple cut is α-balanced, *$0 \le \alpha \le 1/2$, if $\alpha \le \mu(A)/\mu(V) \le 1-\alpha$.*

Note that the cut itself is required to be simple, i.e., it must be realizable by a Jordan curve with the properties from Definition 1. Only for directedness we are then allowed to remove one bothersome node to obtain a clean directed cut.

For our purpose, the weight function μ is induced by letting $\mu(v) = 1$ for every interesting node v and $\mu(v) = 0$ otherwise.

With this relaxed notion of directedness, we are able to obtain balanced cuts. In the next section, we will show how the following result can be combined with the reachability structure from Theorem 3, which requires perfectly directed cuts.

Theorem 4. *Any plane dag with $\kappa \ge 2$ interesting nodes has an almost-directed $1/3$-balanced cut.*

In Figure 5, any two interesting nodes together with the center (playing the role of the exceptional node) form such a cut.

Proof. Choose a linear extension of the given dag $G = (V, E)$. Processing the nodes in this order, one by one, we maintain a collection of simple out-directed node sets A_i. These sets may grow and get merged, they never shrink. Our goal is to create a set that reaches a mass between $\kappa/3$ and $2\kappa/3$. We make sure that in an individual step, weights do not increase by more than $\kappa/3$; so as soon as some A_i gets heavier than $\kappa/3$ we will be done.

Figuratively, we like to think of the chosen order as assigning heights to the nodes such that higher nodes come first in the order and thus all arcs of the dag lead downwards. Envision this height field as located in a big sea of water. Processing nodes in the chosen order can be seen as lowering the waterline step by step, raising node after node out of the water. The sets A_i we maintain, will be islands in this world. They will grow and merge as the sea level sinks. Initially, there are no sets A_i; everything is under water.

When a new node v rises out of the water, we distinguish different cases:

Case 1. If (v) has in-degree zero, we create a new island $A_{\text{new}} = \{v\}$.

Case 2. In case of positive in-degree, each in-neighbor belongs to some island A_i already. Let $I = \{i \mid \text{some node in } A_i \text{ has } v \text{ as out-neighbor}\}$. All these islands, together with v and all the connecting arcs, partition the remaining nodes $V \setminus (\bigcup_I A_i \cup \{v\})$ into disjoint bays B_j, as shown in Figure 6.

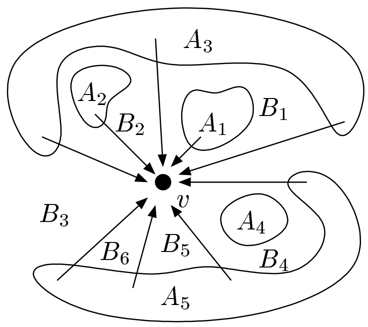

Fig. 6. An arrangement of islands and bays.

Note that islands that are not listed in I are not used for partitioning but instead become part of their surrounding bays. So bays may actually contain little enclosed islands. For example, in the figure we have $A_4 \subseteq B_4$. This guarantees that each bay B_j is an in-directed simple set.

Case 2(a). We now measure the weight of the bays. If any of them, B_j, say, weighs at least $\kappa/3$, we group everything else, $V \setminus B_j$, that is, in one big island A_{new}, destroying all islands contained therein. Note that further sets A_i may survive, namely if they were located within B_j. Also, the new island may contain underwater nodes which have to be removed from the global list. Such enclosed lakes, however, pose no problem. By construction, the resulting island A_{new} is a simple out-directed set as required. It also won't be heavier than than $2\kappa/3$. We may proceed with the next node v' in our total order. Note that in the special case $\delta^-(v) = 1$, the above procedure simply adds the node v to the one island of its only in-neighbor.

Case 2(b). We are left with a situation where the mass of each single island A_i and also of each bay B_j lies strictly below $\kappa/3$. In this case, we will directly construct the desired cut, thus terminating the big loop over all nodes.

We define *levels* for islands and bays: On level 0 lie all A_i and B_j for which $A_i \cup \{v\}$ respectively $B_j \cup \{v\}$ is a simple set. Take away all these sets and let level 1 consist of all islands and bays that now become simply connected if extended by v. Repeat this process, defining levels iteratively until the whole plane is exhausted. In the example of Figure 6, level 0 contains the sets A_1, A_2, B_4, B_5, and B_6; on level 1 there are B_1, B_2, and A_5; and the last level is made up of A_3 and B_3.

We now successively merge islands and bays in order to create some area of land or sea that has the desired mass. Therefore, consider the total mass of all level-0 islands. If it exceeds $\kappa/3$, we can join a suitable subset of these islands with the node v to form an almost out-directed set of the desired mass. Likewise, a large total mass of level-0 bays would give an almost in-directed set.

So assume that there is less than $\kappa/3$ island mass on level-0 and likewise for the bays. We then merge the levels 0 and 1. Each level-1 island absorbs all level-0 bays it encloses and each level-1 bay unites with its contained level-0 islands. In our example, A_5 would become a big island that now includes B_4, B_5, and B_6, while the lakes B_1 and B_2 would swallow A_1 and A_2, respectively. There are two possibilities now. Either one of the new sets grows beyond $\kappa/3$; then this set, extended by v, has the desired properties. Or, all new sets remain below the $\kappa/3$ threshold; then we step to the next level. That means, we measure the total island mass and total bay mass on level 1 (also counting enclosed level-0 sets) and, in case they don't reach the $\kappa/3$ bound, join all level-1 sets to their surrounding level-2 sets.

This way we grow larger and larger simple sets, who always have the property that with the possible exception of the central node v, they are either completely out- or in-directed. Eventually, by the time we reach the highest level, we must find a set of the desired mass.

This finishes the case distinctions. As long as we don't happen to find a good set with case 2(b), we keep lowering the waterline node by node, growing and merging islands until one of them reaches the desired size. For a connected dag this must happen at some point because in the end we would have just one big island of weight κ. If, for a disconnected dag, the process ends with many islands of insufficient weight, we may simply merge a suitable set of islands to achieve a total weight in the interval $[\kappa/3, 2\kappa/3]$. With no arcs outside of the islands left, the result will obviously be a simple cut. $\square$

5 Putting Things Together

On the large scale, our algorithm for dags works as follows. It uses Proposition 4 to find a balanced almost-directed simple cut (A, B). Almost-directedness guarantees that any path from a node in A to a node in B either crosses the cut exactly once or passes through an exceptional node v that might violate perfect directedness. The other direction is even simpler: Any path from B into A must pass through that v since all other cross-cut arcs point in the opposite direction.

Theorem 3 allows us to represent all paths from interesting nodes in A to interesting nodes in B that do not contain a B-A-arc with a graph of size $\mathcal{O}(\kappa \log \kappa)$. All remaining cross-cut paths go through v and are easily represented by at most a linear number of arcs: Include v as a node of $\mathcal{H}$ and express each reachability from an interesting node to v or from v to an interesting node by a single arc in the respective direction.

We now take care of reachabilities amongst interesting nodes in A and within B by recursion. Therefore observe that for connections between A-nodes, we don't have to consider B anymore. A directed path between two nodes in A cannot cross the cut—unless it passes through v, but all those paths have already been taken care of in the cross-cut step above, as a side effect, actually. Likewise for reachabilities within B. Hence, we really face two smaller problems of the same type as the original one. The balancedness provided by Proposition 4 ensures that we have to recurse by $\Theta(\log \kappa)$ levels. Altogether, this gives a representation of size $\mathcal{O}(\kappa \log^2 \kappa)$.

This proves the statement of Theorem 1 for dags. The full claim requires the extensions of the subsequent section.

6 Handling Cycles

A simple cycle in a plane graph divides the plane into two areas A and B, such that any path from one area to the other passes through a node on the cycle. Let U_A be the set of interesting nodes in A that are reachable from the cycle. Then for every interesting node b in B, there are to options: (1) There is a path from b to a node on the cycle and b reaches all of the nodes in U_A, or (2) b does not reach the cycle and does not reach any node in A.

A strongly connected component (SCC) that is not a simple cycle, divides the plane into more than two areas, which again must communicate through the SCC. One of these areas is "outside" and the rest are "inside" the SCC. Note that for any two distinct SCCs, either one is enclosed in an inside-area of the other or they are outside each other. (Otherwise, they would share at least one node and thus be the same SCC.) This determines a hierarchy of the SCCs, where there are no SCCs inside a level-0 SCC and a level-i SCC contains only SCCs of lower levels.

Conceptually, our algorithm represents every SCC C by a single node x_C. The communication between the different areas defined by C is realized by paths through the node x_C. The algorithm proceeds in a top-down fashion. For each maximal SCC C (i.e., an SCC that is not inside any other SCC) it first recursively generates an RS $\mathcal{H}_A$ for each inside-area A of C. By definition, this $\mathcal{H}_A$ contains every interesting node in A. Then, each maximal SCC C along with everything inside of it is contracted into a single node x_C, so we are left with a dag. The representative x_C of C is designated interesting iff there was at least one interesting node on or inside C. We then construct an RS $\overline{\mathcal{H}}$ for this dag, using the algorithm of the previous sections (*dag computation*). Finally, for each contracted SCC C and for every interesting node v on or inside C, we add the arc (v, x_C) if v reaches C and the arc (x_C, v) if C reaches v (*cross link*).

This procedure yields a correct RS because whenever there is a path between two interesting nodes from different areas of an SCC C, the respective representative x_C is made interesting and thus provides the connection between these nodes.

In order to bound the size of the resulting RS, consider the tree of interesting nodes induced by the SCC hierarchy: The interesting nodes of the input digraph form the leaves, interesting representatives of SCCs make up the interior nodes, and the parent of an internal node is the representative of its enclosing SCC and the parent of a leaf is the representative of the SCC to which this leaf belongs. (In case that there is no unique top-level SCC, we add a root to make the forest a tree.)

If this tree is fully branching then the total number of interesting nodes is linear in κ and we can bound the size of the constructed RS $\mathcal{H}$ as follows. Each interesting node appears in exactly one dag computation and adds at most one cross link to $\mathcal{H}$. So all dag computations together contribute $\mathcal{O}(\kappa \log^2 \kappa)$ nodes and arcs to the RS and the cross links add $\mathcal{O}(\kappa)$ more arcs.

Finally, we show how to eliminate all but $\mathcal{O}(\kappa)$ non-branching internal nodes from the tree. Consider a non-branching internal child u of a non-branching node v. The dag computation in which u participated was trivial, i.e., u was the only interesting node there, and thus it did not contribute anything to $\mathcal{H}$. So in $\mathcal{H}$, u has only the two neighbors it has obtained by cross links: its parent v and its only child. We can therefore remove u from $\mathcal{H}$ and connect its neighbors directly by a single arc if u lay on a length-2 path between them. Afterwards, every interior non-branching node has a branching or leaf child so that their total number will be $\mathcal{O}(\kappa)$.

This concludes the proof of Theorem 1.

7 Conclusion and Future Directions

There remains a gap between the upper bound of $\mathcal{O}(\kappa \log^2 \kappa)$ and the trivial lower bound of $\Omega(\kappa)$ on the size of an RS for a planar input. So far, we have not found a planar instance that does not have a linear-size reachability substitute.

We are currently working to extend our reachability substitutes to approximate the lengths of shortest paths between the interesting nodes, as Klein and Subramanian did for the case in which all interesting nodes lie on a constant number of faces [20].

References

1. A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.
2. I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.
3. S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 117–123. ACM Press, 2002.
4. S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. New constructions of (α, β)-spanners and purely additive spanners. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 672–681, 2005.
5. S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *30th Ann. Intl. Colloq. on Automata, Languages and Programming (ICALP)*, 2003.
6. S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $\tilde{O}(n^2)$ time. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 271–280. Society for Industrial and Applied Mathematics, 2004.
7. B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 414–423, 2003.
8. E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1998.
9. Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.
10. Don Coppersmith and Michael Elkin. Sparse source-wise and pair-wise distance preservers. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 660–669. Society for Industrial and Applied Mathematics, 2005.
11. C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: breaking through the o(n/sup 2/) barrier. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 381. IEEE Computer Society, 2000.
12. M. Elkin and D. Peleg. Strong inapproximability of the basic k-spanner problem. In *27th Int'l Symposium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Comput. Sci.*, pages 636–647, 2000.
13. M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-Spanner constructions for general graphs. In *ACM Symposium on Theory of Computing (STOC)*, 2001.
14. M. Elkin and J. Zhang. Efficient algorithms for constructing $(1+\epsilon, \beta)$-spanners in the distributed and streaming models. In *Proc. 23rd Ann. Symp. on Principles of Distributed Computing*, pages 160–168, 2004.
15. D. Eppstein, Z. Galil, G. F. Italiano, and T. H. Spencer. Separator based sparsification for dynamic planar graph algorithms. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 208–217. ACM Press, 1993.
16. J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
17. J. Hopcroft and R. Tarjan. Efficient planarity testing. *JACM*, 21:449–568, 1974.
18. H. T. Hsu. An algorithm for finding a minimal equivalent graph of a digraph. *J. ACM*, 22(1):11–16, 1975.
19. S. Khuller, B. Raghavachari, and N. E. Young. Approximating the minimum equivalent digraph. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 177–186. Society for Industrial and Applied Mathematics, 1994.

20. P. N. Klein and S. Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22:235–249, 1998.
21. D. Peleg and A. A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.
22. L. Roditty. A faster and simpler fully dynamic transitive closure. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 404–412. Society for Industrial and Applied Mathematics, 2003.
23. L. Roditty, M. Thorup, and U. Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In *Proc. 13th Ann. ACM-SIAM Symp. On Discrete Algorithms*, pages 844–851, 2002.
24. L. Roditty and U. Zwick. A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 184–191. ACM Press, 2004.
25. S. Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Proceedings of the First Annual European Symposium on Algorithms*, pages 372–383. Springer-Verlag, 1993.
26. M. Thorup. Near-optimal fully-dynamic graph connectivity. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 343–350. ACM Press, 2000.
27. M. Thorup. Compact oracles for reachability and approximate distances in planar digraphs. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 242. IEEE Computer Society, 2001.
28. M. Thorup and U. Zwick. Approximate distance oracles. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 183–192. ACM Press, 2001.
29. G. Venkatesan, U. Rotics, M. S. Madanlal, J. A. Makowsky, and C. P. Rangan. Restrictions of minimum spanner problems. *Inf. Comput.*, 136(2):143–164, 1997.