Join-Reachability Problems in Directed Graphs*

Loukas Georgiadis[1], Stavros D. Nikolopoulos[2], and Leonidas Palios[2]

[1] Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece
lgeorg@uowm.gr
[2] Department of Computer Science, University of Ioannina, Greece
{stavros,palios}@cs.uoi.gr

Abstract. For a given collection $\mathcal{G}$ of directed graphs we define the *join-reachability graph* of $\mathcal{G}$, denoted by $\mathcal{J}(\mathcal{G})$, as the directed graph that, for any pair of vertices a and b, contains a path from a to b if and only if such a path exists in all graphs of $\mathcal{G}$. Our goal is to compute an efficient representation of $\mathcal{J}(\mathcal{G})$. In particular, we consider two versions of this problem. In the *explicit* version we wish to construct the smallest join-reachability graph for $\mathcal{G}$. In the *implicit* version we wish to build an efficient data structure (in terms of space and query time) such that we can report fast the set of vertices that reach a query vertex in all graphs of $\mathcal{G}$. This problem is related to the well-studied *reachability problem* and is motivated by emerging applications of graph-structured databases and graph algorithms. We consider the construction of join-reachability structures for two graphs and develop techniques that can be applied to both the explicit and the implicit problem. First we present optimal and near-optimal structures for paths and trees. Then, based on these results, we provide efficient structures for planar graphs and general directed graphs.

1 Introduction

In the *reachability problem* our goal is to preprocess a (directed or undirected) graph G into a data structure that can quickly answer queries that ask if a vertex b is reachable from a vertex a. This problem has numerous and diverse applications, including internet routing, geographical navigation, and knowledge-representation systems [13]. Recently, the interest in graph reachability problems has been rekindled by emerging applications of graph data structures in areas such as the semantic web, bio-informatics and social networks. These developments together with recent applications in graph algorithms [4,5,7] have motivated us to introduce the study of the *join-reachability problem* that we define as follows. We are given a collection $\mathcal{G}$ of λ directed graphs $G_i = (V_i, A_i)$, $1 \leq i \leq \lambda$, where each graph G_i represents a binary relation R_i over a set of elements $V \subseteq V_i$ in the following sense: For any $a, b \in V$, we have aR_ib if and only if b is reachable

* This research project has been funded by the John S. Latsis Public Benefit Foundation. The sole responsibility for the content of this paper lies with its authors.

A. Kulikov and N. Vereshchagin (Eds.): CSR 2011, LNCS 6651, pp. 195–208, 2011.

from a in G_i. Let $\mathcal{R} \equiv \mathcal{R}(\mathcal{G})$ be the binary relation over V defined by: $a\mathcal{R}b$ if and only if aR_ib for all $i \in \{1, \ldots, \lambda\}$ (i.e., b is reachable from a in all graphs in $\mathcal{G}$). We can view $\mathcal{R}$ as a type of JOIN operation on graph-structured databases. Our objective is to find an efficient representation of this relation. To the best of our knowledge, this problem has not been previously studied. We will restrict our attention to the case of two input graphs ($\lambda = 2$).

Contribution. In this paper we explore two versions of the join-reachability problem. In the *explicit* version we wish to represent $\mathcal{R}$ with a directed graph $\mathcal{J} \equiv \mathcal{J}(\mathcal{G})$, which we call the *join-reachability graph of* $\mathcal{G}$, i.e., for any $a, b \in V$, we have $a\mathcal{R}b$ if and only if b is reachable from a in $\mathcal{J}$. Our goal is to minimize the size (i.e., the number of vertices plus arcs) of $\mathcal{J}$. We consider this problem in Sections 2 and 3, and present results on the computational and combinatorial complexity of $\mathcal{J}$. In the *implicit* version we wish to represent $\mathcal{R}$ with an efficient data structure (in terms of space and query time) that can report fast all elements $a \in V$ satisfying $a\mathcal{R}b$ for any query element $b \in V$. We deal with the implicit problem in Section 4. First we describe efficient join-reachability structures for simple graph classes. Then, based on these results, we consider planar graphs and general directed graphs. Although we focus on the case of two directed graphs ($\lambda = 2$), we note that some of our results are easily extended for $\lambda \geq 3$ with the use of appropriate multidimensional geometric structures.

Applications. Instances of the join-reachability problem appear in various applications. For example, in the rank aggregation problem [3] we are given a collection of rankings of some elements and we may wish to report which (or how many) elements have the same ranking relative to a given element. This is a special version of join-reachability since the given collection of rankings can be represented by a collection of directed paths with the elements being the vertices of the paths. Similarly, in a graph-structured database with an associated ranking of its vertices we may wish to find the vertices that are related to a query vertex and have higher or lower ranking than this vertex. Instances of join-reachability also appear in graph algorithms arising from program optimization. Specifically, [4] uses a data structure that reports fast vertices that satisfy certain ancestor-descendant relations in a collection of rooted trees. Moreover, in [7] it is shown that any directed graph G with a distinguished source vertex s has two spanning trees rooted at s such that a vertex a is a dominator of a vertex b (meaning that all paths in G from s to b pass through a) if and only if a is an ancestor of b in both spanning trees. This generalizes the graph-theoretical concept of *independent spanning trees*. Two spanning trees of a graph G are independent if they are both rooted at the same vertex r and for each vertex v the paths from r to v in the two trees are internally vertex disjoint. Similarly, λ spanning trees of G are independent if they are pairwise independent. In this setting, we can apply a join-reachability structure to decide if λ given spanning trees are independent. Finally we note that a variant of the join-reachability problem we defined here appears in the context of a recent algorithm for computing two internally vertex-disjoint paths for any pair of query vertices in a 2-vertex connected directed graph [5].

Preliminaries and Related Work. The reachability problem is easy in the undirected case since it suffices to compute the connected components of the input graph. Similarly, the undirected version of the join-reachability problem is also easy, as given the connected components of two undirected graphs G_1 and G_2 with n vertices, we can compute the connected components of $\mathcal{J}(\{G_1, G_2\})$ in $O(n)$ time. On the other hand, no reachability data structure is currently known to simultaneously achieve $o(n^2)$ space and $o(n)$ query time for a general directed graph with n vertices [13]. Nevertheless, efficient reachability structures do exist for several important cases. First, asymptotically optimal structures exist for rooted trees [1] and planar directed graphs with one source and one sink [8,11]. For general planar graphs Thorup [12] gives an $O(n \log n)$-space structure with constant query time. Talamo and Vocca [10] achieve constant query time for lattice partial orders with an $O(n\sqrt{n})$-space structure.

Notation. In the description of our results we use the following notation and terminology. We denote the vertex set and the arc set of a directed graph (digraph) G by $V(G)$ and $A(G)$, respectively. Without loss of generality we assume that $V(G) = V$ for all $G \in \mathcal{G}$. The size of G, denoted by $|G|$, is equal to the number of arcs plus vertices, i.e., $|G| = |V| + |A|$. We use the notation $a \rightsquigarrow_G b$ to denote that b is reachable from a in G. (By definition $a \rightsquigarrow_G a$ for any $a \in V$.) The *predecessors* of a vertex b are the vertices that reach b, and the *successors* of a vertex b are the vertices that are reached from b. Let P be a directed path (dipath); the *rank* of $a \in P$, $r_P(a)$, is equal to the number of predecessors of a in P minus one, and the *height* of $a \in P$, $h_P(a)$, is equal to the number of successors of a in P minus one. For a rooted tree T, we let $T(a)$ denote the subtree rooted at a. We will deal with two special types of directed rooted trees: In an *in-tree*, each vertex has exactly one outgoing arc except for the root which has none; in an *out-tree*, each vertex has exactly one incoming arc except for the root which has none. We use the term *unoriented tree* for a directed tree with no restriction on the orientation of its arcs. Similarly, we use the term *unoriented dipath* to refer to a path in the undirected sense, where the arcs can have any orientation. In our constructions we map the vertices of V to objects in a d-dimensional space and use the notation $x_i(a)$ to refer to the ith coordinate that vertex a receives. Finally, for any two vectors $\xi = (\xi_1, \ldots, \xi_d)$ and $\zeta = (\zeta_1, \ldots, \zeta_d)$, the notation $\xi \leq \zeta$ means that $\xi_i \leq \zeta_i$ for $i = 1, \ldots, d$.

1.1 Preprocessing: Computing Layers and Removing Cycles

Thorup's Layer Decomposition. In [12] Thorup shows how to reduce the reachability problem for any digraph G to reachability in some digraphs with special properties, called *2-layered digraphs.* A *t-layered spanning tree* T of G is a rooted directed tree such that any path in T from the root (ignoring arc directions) is the concatenation of at most t dipaths in G. A digraph G is *t-layered* if it has such a spanning tree. Now we provide an overview of Thorup's reduction. The vertices of G are partitioned into layers $L_0, L_1, \ldots, L_{\mu-1}$ that define a sequence of digraphs $G^0, G^1, \ldots, G^{\mu-1}$ as follows. An arbitrary vertex $v_0 \in V(G)$ is chosen as a root. Then, layer L_0 contains v_0 and the vertices that are reachable from v_0.

For odd i, layer L_i contains the vertices that reach the previous layers L_j, $j < i$. For even i, layer L_i contains the vertices that are reachable from the previous layers L_j, $j < i$. To form G^i for $i > 0$ we contract the vertices in layers L_j for $j \leq i-1$ to a single root vertex r_0; for $i = 0$ we set $r_0 = v_0$. Then G^i is induced by L_i, L_{i+1} and r_0. It follows that each G^i is a 2-layered digraph. Let $\iota(v)$ denote the index of the layer containing v, that is, $\iota(v) = i$ if and only if $v \in L_i$. The key properties of the decomposition are: (i) all the predecessors of v in G are contained in $G^{\iota(v)-1}$ and $G^{\iota(v)}$, and (ii) $\sum_i |G^i| = O(|G|)$.

Removing Cycles. In the standard reachability problem, a useful preprocessing step that can reduce the size of the input digraph is to contract its strongly connected components (strong components) and consider the resulting acyclic graph. When we apply the same idea to join-reachability we have to deal with the complication that the strong components in the two digraphs may differ. Still, we can construct two acyclic digraphs $\hat{G}_1$ and $\hat{G}_2$ such that, for any $a, b \in V$, $a \rightsquigarrow_{\mathcal{J}(G_1,G_2)} b$ if and only if $a \rightsquigarrow_{\mathcal{J}(\hat{G}_1,\hat{G}_2)} b$, and $|\hat{G}_i| \leq |G_i|$, $i = 1, 2$. This is accomplished as follows. First, we compute the strong components of G_1 and G_2 and order them topologically. Let G'_i, $i = 1, 2$, denote the digraph produced after contracting the strong components of G_i. (We remove loops and duplicate arcs so that each G'_i is a simple digraph.) Also, let C_i^j denote the jth strong component of G_i. We partition each component C_i^j into subcomponents such that two vertices are in the same subcomponent if and only if they are in the same strong component in both G_1 and G_2. The subcomponents are the vertices of $\hat{G}_1$ and $\hat{G}_2$. Next we describe how to add the appropriate arcs. The process is similar for the two digraphs so we consider only $\hat{G}_1$.

Let $C_1^{j,1}, C_1^{j,2}, \ldots, C_1^{j,l_j}$ be the subcomponents of C_1^j, which are ordered with respect to the topological order of G'_2. That is, if $x \in C_1^{j,i}$ and $y \in C_1^{j,i'}$, where $i < i'$, then in the topological order of G'_2 the component of x precedes the component of y. We connect the subcomponents by adding the arcs $(C_1^{j,i}, C_1^{j,i+1})$ for $1 \leq i < l_j$. Moreover, for each arc (C_1^i, C_1^j) in $A(G'_1)$ we add the arc $(C_1^{i,l_i}, C_1^{j,1})$ to $A(\hat{G}_1)$, where C_1^{i,l_i} is the last subcomponent of C_1^i. It is straightforward to verify that $a \rightsquigarrow_{\mathcal{J}} b$ if and only if a and b are in the same subcomponent or the subcomponent of a is a predecessor of the subcomponent of b in both $\hat{G}_1$ and $\hat{G}_2$.

2 Computational Complexity

We explore the computational complexity of computing the smallest $\mathcal{J}(\{G_1, G_2\})$: Given two digraphs $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ we wish to compute a digraph $\mathcal{J} \equiv \mathcal{J}(\{G_1, G_2\})$ of minimum size such that for any $a, b \in V$, $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $a \rightsquigarrow_{G_1} b$ and $a \rightsquigarrow_{G_2} b$. We consider two versions of this problem, depending on whether $\mathcal{J}$ is allowed to have Steiner vertices (i.e., vertices not in V) or not: In the *unrestricted* version $V(\mathcal{J}) \supseteq V$, while in the *restricted* version $V(\mathcal{J}) = V$. Computing $\mathcal{J}$ is NP-hard in the unrestricted case. This is implied by a straightforward reduction from the *reachability substitute problem*, which

was shown to be NP-hard by Katriel et al. [9]. In this problem we are given a digraph H and a subset $U \subseteq V(H)$, and ask for the smallest digraph H^* such that for any $a, b \in U$, $a \rightsquigarrow_{H^*} b$ if and only if $a \rightsquigarrow_H b$. For the reduction, we let $G_1 = H$ and let G_2 contain all the arcs connecting vertices in U only, that is, $A(G_2) = U \times U$. Clearly, for any $a, b \in U$ we have $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $a \rightsquigarrow_H b$. Therefore computing the smallest join-reachability graph is equivalent to computing H^*. In the restricted case, on the other hand, we can compute $\mathcal{J}$ using transitive closure and transitive reduction computations, which can be done in polynomial time [2].

Theorem 1. *Let $\mathcal{J}$ be the smallest join-reachability graph of a collection of digraphs. The computation of $\mathcal{J}$ is feasible in polynomial time if Steiner vertices are not allowed, and NP-hard otherwise.*

Note that allowing Steiner vertices can reduce the size of $\mathcal{J}$ significantly. In Section 3 we explore the combinatorial complexity of the unrestricted join-reachability graph and provide bounds for $|\mathcal{J}|$ in several cases.

3 Combinatorial Complexity

In this section we provide bounds on the size of $\mathcal{J}(\{G_1, G_2\})$ for several types of graphs. These are summarized in the next theorem.

Theorem 2. *Given two digraphs G_1 and G_2 with n vertices, the following bounds on the size of the join-reachability graph $\mathcal{J}(\{G_1, G_2\})$ hold:*

(a) $\Theta(n \log n)$ *in the worst case when G_1 is an unoriented tree and G_2 is an unoriented dipath.*
(b) $O(n \log^2 n)$ *when both G_1 and G_2 are unoriented trees.*
(c) $O(n \log^2 n)$ *when G_1 is a planar digraph and G_2 is an unoriented dipath.*
(d) $O(n \log^3 n)$ *when both G_1 and G_2 are planar digraphs.*
(e) $O(\kappa_1 n \log n)$ *when G_1 is a digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is an unoriented dipath.*
(f) $O(\kappa_1 n \log^2 n)$ *when G_1 is a digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is a planar graph.*
(g) $O(\kappa_1 \kappa_2 n \log n)$ *when each G_i, $i = 1, 2$, is a digraph that can be covered with κ_i vertex-disjoint dipaths.*

In the following sections we prove Theorem 2. In each case we provide a construction of the corresponding join-reachability graph that achieves the claimed bound. In Section 4 we provide improved space bounds for the implicit representation of $\mathcal{J}(\{G_1, G_2\})$, i.e., data structures that answer join-reachability reporting queries fast. Still, a process that computes an explicit representation of $\mathcal{J}(\{G_1, G_2\})$ can be useful, as it provides a natural way to handle collections of more than two digraphs (i.e., it allows us to combine the digraphs one pair at a time).

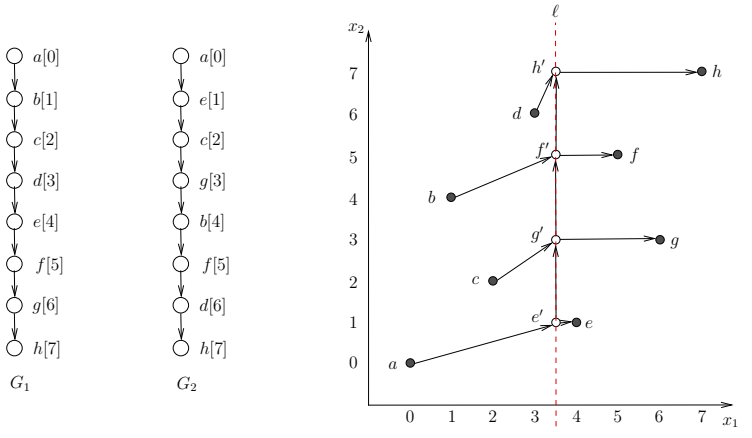

Fig. 1. The mapping of the vertices of two dipaths to a 2d rank space and the construction of $\mathcal{J}_\ell$; Steiner vertices in $\mathcal{J}_\ell$ are shown white

3.1 Two Paths

We start with the simplest case where G_1 and G_2 are dipaths with n vertices. First we show that we can construct a join-reachability graph of size $O(n \log n)$. Given this result we can provide bounds for trees, planar and general digraphs. Then we show this bound is tight, i.e., there are instances for which $\Omega(n \log n)$ size is needed. We begin by mapping the vertices of V to a two-dimensional rank space: Each vertex a receives coordinates $(x_1(a), x_2(a))$ where $x_1(a) = r_{G_1}(a)$ and $x_2(a) = r_{G_2}(a)$. Note that these ranks are integers in the range $[0, n-1]$. Now we can view these vertices as lying on an $n \times n$ grid, such that each row and each column of the grid contains exactly one vertex. Clearly, $a \mathcal{R} b$ if and only if $(x_1(a), x_2(a)) \leq (x_1(b), x_2(b))$.

Upper bound. We use a simple divide-and-conquer method. Let ℓ be the vertical line with x_1-coordinate equal to $n/2$. A vertex z is *to the right of* ℓ if $x_1(z) \geq n/2$ and *to the left of* ℓ otherwise. The first step is to construct a subgraph $\mathcal{J}_\ell$ of $\mathcal{J}$ that connects the vertices to the left of ℓ to the vertices to the right of ℓ. For each vertex b to the right of ℓ we create a Steiner vertex b' and add the arc (b', b). Also, we assign to b' the coordinates $(n/2, x_2(b))$. We connect these Steiner vertices in a dipath starting from the vertex with the lowest x_2-coordinate. Next, for each vertex a to the left of ℓ we locate the Steiner vertex b' with the smallest x_2-coordinate such that $x_2(a) \leq x_2(b')$. If b' exists we add the arc (a, b'). See Figure 1. Finally we recurse for the vertices to the left of ℓ and for the vertices to the right of ℓ. It is easy to see that $\mathcal{J}$ contains a path from a to b if and only if $(x_1(a), x_2(a)) \leq (x_1(b), x_2(b))$. To bound $|\mathcal{J}|$ note that we have $O(\log n)$ levels of recursion, and at each level the number of added Steiner vertices and arcs is $O(n)$. Hence, the $O(n \log n)$ bound for two dipaths follows.

The case of two unoriented dipaths G_1 and G_2 can be reduced to that of dipaths, yielding the same $O(n \log n)$ bound. This is accomplished by splitting

G_1 and G_2 to maximal subpaths that consist of arcs with the same orientation. Then $\mathcal{J}$ is formed from the union of separate join-reachability graphs for each pair of subpaths of G_1 and G_2. The $O(n \log n)$ bound follows from the fact that each vertex appears in at most two subpaths of each unoriented dipath, so in at most four subgraphs. We remark that our construction can be generalized to handle more dipaths, with an $O(\log n)$ factor blowup per additional dipath.

Lower bound. Let G_1 be any dipath, and let $x_1(a) = r_{G_1}(a)$. Also let $x_1^i(a)$ denote the ith bit in the binary representation of $x_1(a)$ and let $\beta = \lceil \log_2 n \rceil$ be the number of bits in this representation. We use similar notation for $x_2(a)$. We define G_2 such that the rank of a in G_2 is $x_2(a) = x_1(a)^R$, where $x_1(a)^R$ is the integer formed by the bit-reversal in the binary representation of $x_1(a)$, i.e., $x_2^i(a) = x_1^{\beta-1-i}(a)$ for $0 \leq i \leq \beta - 1$. Let $\mathcal{P}$ be the set that contains all pairs of vertices (a, b) that satisfy $x_1^i(a) = 0$, $x_1^i(b) = 1$ and $x_1^j(a) = x_1^j(b), j \neq i$, for $0 \leq i \leq \beta - 1$. Notice that for a pair $(a, b) \in \mathcal{P}$, $x_1(a) < x_1(b)$ and $x_1(a)^R < x_1(b)^R$. Hence $(x_1(a), x_2(a)) < (x_1(b), x_2(b))$, which implies $a \rightsquigarrow_{\mathcal{J}} b$. Now let G be the digraph that is formed by the arcs $(a, b) \in \mathcal{P}$. Then $a \rightsquigarrow_G b$ only if $a \rightsquigarrow_{\mathcal{J}} b$. Moreover, the transitive reduction of G is itself and has size $\Omega(n \log n)$. We also observe that any two vertices in G share at most one immediate successor. Therefore the size of G cannot be reduced by introducing Steiner vertices. This implies that size of $\mathcal{J}$ is also $\Omega(n \log n)$.

3.2 Tree and Path

Let G_1 be a rooted (in- or out-)tree and G_2 a dipath. First we note that the ancestor-descendant relations in a rooted tree can be described by two linear orders (corresponding to a preorder and a postorder traversal of the tree) and therefore we can get an $O(n \log^2 n)$ bound on the size of $\mathcal{J}$ using the result of Section 3.1. Here we provide an $O(n \log n)$ bound, which also holds when G_1 is unoriented. This upper bound together with the $\Omega(n \log n)$ lower bound of Section 3.1 implies Theorem 2(a).

Let T be the rooted tree that results from G_1 after removing arc directions. We associate each vertex $x \in T$ with a label $h(x) = h_{G_2}(x)$, the height of x in G_2. If G_1 is an out-tree then any vertex b must be reachable from all its ancestors a in T with $h(a) > h(b)$. Similarly, if G_1 is an in-tree then any vertex b must be reachable from all its descendants a in T with $h(a) > h(b)$. We begin by assigning a depth-first search interval to each vertex in T. Let $I(a) = [s(a), t(a)]$ be the interval of a vertex $a \in T$; $s(a)$ is the time of the first visit to a (during the depth-first search) and $t(a)$ is the time of the last visit to a. These times are computed by incrementing a counter after visiting or leaving a vertex during the search. This way all the $s()$ and $t()$ values that are assigned are distinct and for any vertex a we have $1 \leq s(a) < t(a) \leq 2n$. Moreover, by well-known properties of depth-first search, we have that a is an ancestor of b in T if and only if $I(b) \subseteq I(a)$; if a and b are unrelated in T then $I(a)$ and $I(b)$ do not intersect. Now we map each vertex a to the x_1-axis-parallel segment $S(a) = I(a) \times h(a)$.

As in Section 3.1 we use a divide-and-conquer method to build $\mathcal{J}$. We will consider G_1 to be an out-tree; the in-tree case is handled similarly and yields the

same asymptotic bound. Let ℓ be the horizontal line with x_2-coordinate equal to $n/2$. A vertex x is *above* ℓ if $h(x) \geq n/2$; otherwise ($h(x) < n/2$), x is *below* ℓ. We create a subgraph $\mathcal{J}_\ell$ of $\mathcal{J}$ that connects the vertices above ℓ to the vertices below ℓ. To that end, for each vertex u above ℓ we create a Steiner vertex u' together with the arc (u, u'). Let z be the nearest ancestor of u in T that is above ℓ. If z exists then we add the arc (z', u'). Then, for each vertex y below ℓ we locate the nearest ancestor u of y in T that is above ℓ. If u exists then we add the arc (u', y). Finally, we recurse for the vertices above ℓ and for the vertices below ℓ.

It is not hard to verify the correctness of the above construction. The size of the resulting graph can be bounded by $O(n \log n)$ as in Section 3.1. Furthermore, we can generalize this construction for an unoriented tree and an unoriented path, and accomplish the same $O(n \log n)$ bound as required by Theorem 2(a). We omit the details which are similar to the more complicated construction of Section 3.4.

3.3 Two Trees

The construction of Section 3.2 can be extended to handle more than one dipath. We show how to apply this extension in order to get an $O(n \log^2 n)$ bound for the join-reachability graph of two rooted trees. We consider the case where G_1 is an out-tree and G_2 is an in-tree; the other two cases (two out-trees and two in-trees) are handled similarly.

Let T_1 and T_2 be the corresponding undirected trees. We assign to each vertex a two depth-first search intervals $I_1(a) = [s_1(a), t_1(a)]$ and $I_2(a) = [s_2(a), t_2(a)]$, where $I_j(a)$ corresponds to T_j, $j = 1, 2$. We create two linear orders (i.e., dipaths), P_1 and P_2, from the I_2-intervals as follows: In P_1 the vertices are ordered by decreasing s_2-value and in P_2 by increasing t_2-value. Each vertex a is mapped to an x_1-axis-parallel segment $I_1(a) \times x_2(a) \times x_3(a)$ (in three dimensions), where $x_2(a) = h_{P_1}(a)$ and $x_3(a) = h_{P_2}(a)$. Then b is reachable from a in $\mathcal{J}$ if and only if $I_1(b) \subseteq I_1(a)$ and $(x_2(b), x_3(b)) \leq (x_2(a), x_3(a))$. See Figure 2.

Again we employ a divide-and-conquer approach and use the method of Section 3.2 as a subroutine. The details are as follows. Let p be the plane with x_3-coordinate equal to $n/2$. We construct a subgraph $\mathcal{J}_p$ of $\mathcal{J}$ that connects the vertices above p (i.e., vertices z with $x_3(z) \geq n/2$) to the vertices below p (i.e., vertices z with $x_3(z) < n/2$). Then we use recursion for the vertices above p and the vertices below p.

We construct $\mathcal{J}_p$ using the method of Section 3.2 with some modifications. Let ℓ be the horizontal line with x_2-coordinate equal to $n/2$. We create a subgraph $\mathcal{J}_{p,\ell}$ of $\mathcal{J}_p$ that connects the vertices above p and ℓ to the vertices below p and ℓ. To that end, for each vertex z with $(x_2(z), x_3(z)) \geq (n/2, n/2)$ we create a Steiner vertex z' together with the arc (z, z'). Let u be the nearest ancestor of z in T_1 such that $(x_2(u), x_3(u)) \geq (n/2, n/2)$. If u exists then we add the arc (u', z'). Finally, for each vertex y with $(x_2(y), x_3(y)) < (n/2, n/2)$ we locate the nearest ancestor z of y in T_1 such that $(x_2(z), x_3(z)) \geq (n/2, n/2)$. If z exists then we add the arc (z', y). Finally, we recurse for the vertices above ℓ and for

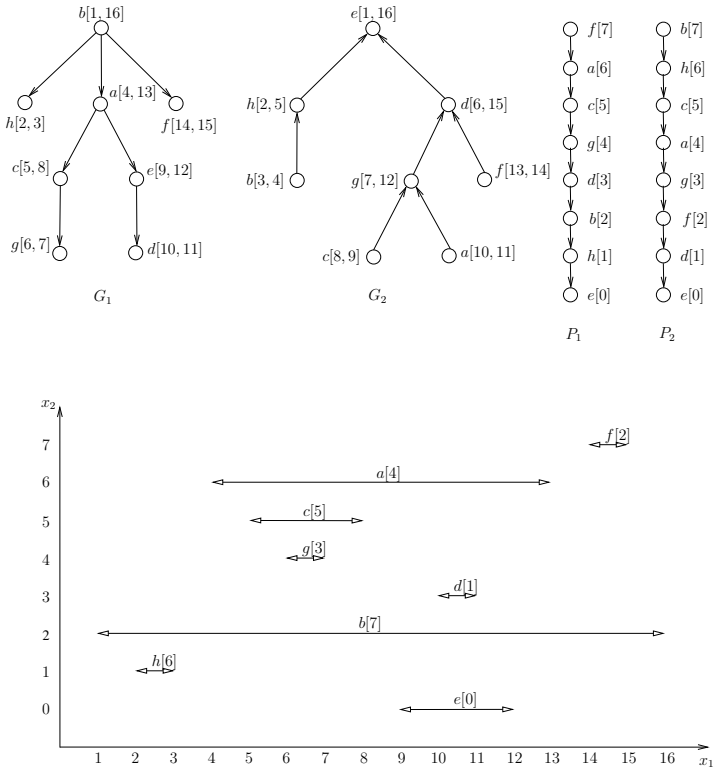

Fig. 2. The mapping of the vertices of two rooted trees to horizontal segments in 3d. The value in brackets above the segments correspond to the x_3-coordinates.

the vertices below ℓ. The above construction implies that $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $I_1(b) \subseteq I_1(a)$ and $(x_2(b), x_3(b)) \leq (x_2(a), x_3(a))$, as required.

Now we bound the size of our construction. From Section 3.2 we have that the size of each substructure $\mathcal{J}_p$ is $O(n \log n)$. Since each vertex participates in $O(\log n)$ such substructures, the total size is bounded by $O(n \log^2 n)$.

3.4 Unoriented Trees

We can reduce the case of unoriented trees to that of rooted trees by applying Thorup's layer decomposition (see Section 1.1). We apply this decomposition to both G_1 and G_2. Let $G_i^0, G_i^2, \ldots, G_i^{\mu_i - 1}$ be the sequence of rooted trees produced from G_i, $i = 1, 2$, where each G_i^j is a 2-layered tree. See Figure 3. For even j, G_i^j consists of a *core* out-tree, formed by the arcs directed away from the root, and a collection of *fringe* in-trees. The situation is reversed for odd j, where the core tree is an in-tree and the fringe trees are out-trees. We call a vertex of the core tree a *core vertex*; we call a vertex of a fringe tree (excluding its root) a *fringe vertex*.

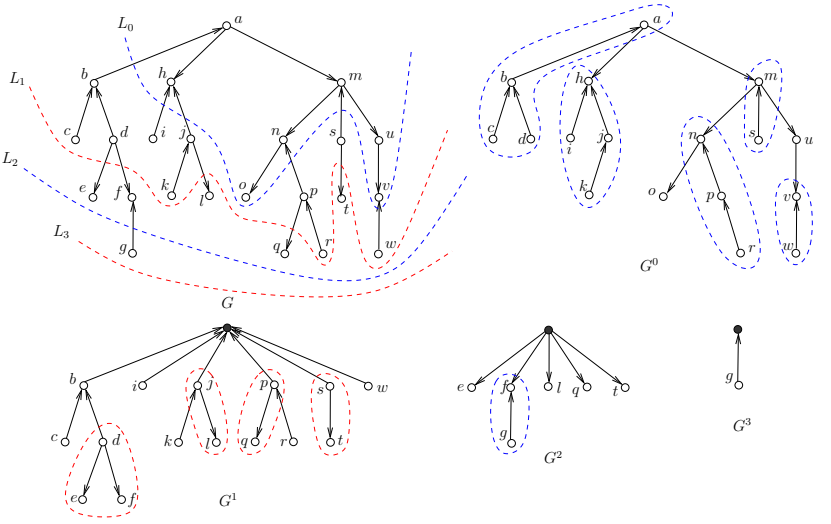

Fig. 3. An unoriented tree and its sequence of 2-layered trees. Fringe trees are encircled.

We build $\mathcal{J}$ as the union of join-reachability graphs $\mathcal{J}_{i,j}$ for each pair (G_1^i, G_2^j). Each graph $\mathcal{J}_{i,j}$ is constructed similarly to Section 3.3, with the exception that we have to take special care for the fringe vertices. (We also remark that in general $\mathcal{J}_{i,j} \neq \mathcal{J}(G_1^i, G_2^j)$.) A vertex $z \in V(G_1^i) \cap V(G_2^j)$ is included in $\mathcal{J}_{i,j}$ if one of the following cases hold: (i) z is a core vertex in at least one of G_1^i and G_2^j, or (ii) z is a fringe vertex in both G_1^i and G_2^j and the corresponding fringe trees containing z are either both in-trees or both out-trees. Let $V_{i,j}$ be the vertices in $V(G_1^i) \cap V(G_2^j)$ that satisfy the above condition.

If $V_{i,j} = \emptyset$ then $\mathcal{J}_{i,j}$ is empty. Now suppose $V_{i,j} \neq \emptyset$. First consider the case where the core of G_1^i is an out-tree. We contract each fringe in-tree to its root and let the new core supervertex correspond to the vertices of the contracted fringe tree. Let $\hat{G}_1^i$ be the out-tree produced from this process. Equivalently, if the core of G_1^i is an in-tree then the contraction of the fringe out-trees produces an in-tree $\hat{G}_1^i$. We repeat the same process for G_2^j. Next, we assign a depth-first search interval $I_1(z)$ to each vertex z in $\hat{G}_1^i$ and a depth-first search interval $I_2(z)$ to each vertex z in $\hat{G}_2^j$, as in Section 3.3. The vertices in $V_{i,j}$ are assigned a depth-first search interval in both trees, and therefore can be mapped to horizontal segments in a 3d space, as in Section 3.3. Hence, we can employ the method of Section 3.3 with some necessary changes that involve the fringe vertices. Let $z \in V_{i,j}$ be a fringe vertex in at least one of G_1^i and G_2^j. If the fringe tree containing z is an in-tree then we only include in $\mathcal{J}_{i,j}$ arcs leaving z; otherwise we only include arcs entering z.

Finally we need to show that the size of the resulting graph is $O(n \log^2 n)$. This follows from the fact that each subgraph $\mathcal{J}_{i,j}$ has size $O(n \log^2 n)$ and that each vertex can appear in at most four such subgraphs. Theorem 2(b) follows.

3.5 Planar Digraphs

Now we turn to planar digraphs and combine our previous constructions with Thorup's reachability oracle [12]. From this combination we derive the bounds stated in Theorem 2(c) and (d). First we need to provide some details for the reachability oracle of [12].

Let G be a planar digraph, and let $G^0, G^1, \ldots, G^{\mu-1}$ be the sequence of 2-layered digraphs produced from G as described in Section 1.1. Consider one of these digraphs G^i. The next step is to obtain a separator decomposition of G^i. To that end, we treat G^i as an undirected graph and compute a separator S whose removal separates G^i into components, each with at most half the vertices. The separator S consists of three root paths of a spanning tree of G^i rooted at r_0. Because G^i is 2-layered, each root path in S corresponds to at most two dipaths in G^i. The key idea now is to process each separator dipath Q and find the connections between $V(G^i)$ and Q. For each $v \in V(G^i)$ two quantities are computed: (i) $\text{from}_v[Q]$ which is equal to $r_Q(u)$, where $u \in Q$ is the vertex with the highest rank in Q such that $u \rightsquigarrow_{G^i} v$, and (ii) $\text{to}_v[Q]$ which is equal to $r_Q(u)$, where $u \in Q$ is the vertex with the lowest rank in Q such that $v \rightsquigarrow_{G^i} u$. Clearly there is a path from a to b that passes though Q if and only if $\text{to}_a[Q] \leq \text{from}_b[Q]$. The same process is carried out recursively for each component of $G^i \setminus V(S)$. The depth of this recursion is $O(\log n)$, so each vertex is connected to $O(\log n)$ separator dipaths. The space and construction time for this structure is $O(n \log n)$.

Now we consider how to construct a join-reachability graph when G_1 is a planar digraph. We begin with the case where G_2 is a dipath. First we perform the layer decomposition of G_1 and construct the corresponding graph sequence $G_1^0, G_1^1, \ldots, G_1^{\mu-1}$. Then we form pairs of digraphs $P_i = \{G_1^i, G_2^i\}$ where G_2^i is a dipath containing only the vertices in $V(G_1^i)$ in the order they appear in G_2. Clearly $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $a \rightsquigarrow_{\mathcal{J}_{\iota(b)-1}} b$ or $a \rightsquigarrow_{\mathcal{J}_{\iota(b)}} b$, where $\mathcal{J}_i$ is the join-reachability graph of P_i. Then $\mathcal{J}$ is formed from the union of $\mathcal{J}_0, \ldots, \mathcal{J}_{\mu-1}$.

To construct $\mathcal{J}_i$ we perform the separator decomposition of G_1^i, so that each vertex is associated with $O(\log n)$ separator dipaths. Let Q be such a separator dipath. Also, let V_Q be the set of vertices that have a successor or a predecessor in Q. We build a subgraph $\mathcal{J}_{i,Q}$ of $\mathcal{J}_i$ for the vertices in V_Q; $\mathcal{J}_i$ is formed from the union of the subgraphs $\mathcal{J}_{i,Q}$ for all the separator dipaths of G_1^i. The construction of $\mathcal{J}_{i,Q}$ is carried out as follows. Let $z \in V_Q$. If z has a predecessor in Q then we create a vertex z^- which is assigned coordinates $x_1(z^-) = \text{from}_z[Q]$ and $x_2(z^-) = r_{G_2}(z)$, and add the arc (z, z^-). Similarly, if z has a successor in Q then we create a vertex z^+ which is assigned coordinates $x_1(z^+) = \text{to}_z[Q]$ and $x_2(z^+) = r_{G_2}(z)$, and add the arc (z^+, z).

Now we can use the method of Section 3.1 to build the rest of $\mathcal{J}_{i,Q}$, so that $a \rightsquigarrow_{\mathcal{J}_{i,Q}} b$ if and only if $(x_1(a^+), x_2(a^+)) \leq (x_1(b^-), x_2(b^-))$. Let ℓ be the vertical line with x_1-coordinate equal to $n/2$. The first step is to construct the subgraph of $\mathcal{J}_{i,Q}$ that connects the vertices a^+ with $x_1(a^+) \leq n/2$ to the vertices b^- with $x_1(b^-) \geq n/2$. For each such b^- we create a Steiner vertex b' and add the arc (b', b^-). Also, we assign to b' the coordinates $(n/2, x_2(b^-))$. We connect

these Steiner vertices in a dipath starting from the vertex with the lowest x_2-coordinate. Next, for each vertex a^+ with $x_1(a^+) \leq n/2$ we locate the Steiner vertex b' with the smallest x_2-coordinate such that $x_2(a^+) \leq x_2(b')$. If b' exists we add the arc (a^+, b'). Finally we recurse for the vertices with x_1-coordinate in $[1, n/2)$ and for the vertices with x_1-coordinate in $(n/2, n]$.

It remains to bound the size of $\mathcal{J}$. From Section 3.1, we have $|\mathcal{J}_{i,Q}| = O(|V_Q| \log |V_Q|)$. Moreover, the bound $\sum_Q |V_Q| = O(|V(G_1^i)| \log |V(G_1^i)|)$, where the sum is taken over all separator paths of G_1^i, implies $|\mathcal{J}_i| \leq \sum_Q |\mathcal{J}_{i,Q}| = O(|V(G_1^i)| \log^2 |V(G_1^i)|)$. Finally, since $\sum_i |V(G_1^i)| = O(n)$ we obtain $|\mathcal{J}| \leq \sum_i |\mathcal{J}_i| = O(n \log^2 n)$.

We handle the case where G_2 is an unordered dipath as noted in Section 3.1, which implies Theorem 2(c). The methods we developed here in combination with the structures of Section 3.4 result to a join-reachability graph of size $O(n \log^3 n)$ for a planar digraph and an unoriented tree. The same bound of $O(n \log^3 n)$ is achieved for two planar digraphs, as stated in Theorem 2(d).

3.6 General Digraphs

A technique that is used to speed up transitive closure and reachability computations is to cover a digraph with simple structures such as dipaths, chains, or trees (e.g., see [1]). Such techniques are well-suited to our framework as they can be combined with the structures we developed earlier. We also remark that the use of the preprocessing steps of Section 1.1 reduces the problem from general digraphs to acyclic and 2-layered digraphs. In this section we describe how to obtain join-reachability graphs with the use of dipath covers. This gives the bounds stated in Theorem 2(e)-(g); similar results can be derived with the use of tree covers. Again for simplicity, we first consider the case where G_1 is a general digraph and G_2 is a dipath.

A *dipath cover* is a decomposition of a digraph into vertex-disjoint dipaths. Let $P_1^1, P_1^2, \ldots P_1^{\kappa_1}$ be a dipath cover of G_1. For each vertex v and each path P_1^i we compute $\text{from}_v[P_1^i]$, i.e., $r_{P_1^i}(z)$ where $z \in P_1^i$ is the vertex with the highest rank in P_1^i such that $z \rightsquigarrow_{G_1} v$. Let P_2^i be the dipath that consists of the vertices in P_1^i ordered by increasing rank in G_2. Also, set $\text{from}_v[P_2^i] = r_{P_2^i}(z)$ where $z \in P_2^i$ is the vertex with the largest rank such that $r_{G_2}(z) \leq r_{G_2}(v)$. Let $V_{P_1^i}$ be set of vertices that have a predecessor in P_1^i. We build a subgraph $\mathcal{J}_i$ of $\mathcal{J}$ that connects the vertices of P_1^i to $V_{P_1^i}$. Then $\mathcal{J}$ is formed from the union of the subgraphs $\mathcal{J}_i$. For each $z \in V_{P_1^i}$ we create a vertex z^- which is assigned coordinates $x_1(z^-) = \text{from}_z[P_1^i]$ and $x_2(z^-) = \text{from}_z[P_2^i]$, and add the arc (z^-, z). Also, for each $z \in P_1^i$ we create a vertex z^+ which is assigned coordinates $x_1(z^+) = r_{P_1^i}(z)$ and $x_2(z^+) = r_{P_2^i}(z)$, and add the arc (z, z^+). Now we can build a join-reachability graph, so that $a \rightsquigarrow_{\mathcal{J}_i} b$ if and only if $(x_1(a^+), x_2(a^+)) \leq (x_1(b^-), x_2(b^-))$, as in Section 3.5.

The size of this graph is bounded by $\sum_i |V_{P_1^i}| \log |V_{P_1^i}| = O(\kappa_1 n \log n)$, which implies the result of Theorem 2(e). We can extend this method to handle two general digraphs and obtain the bound of Theorem 2(g). The case where G_2 is

planar digraph is handled by combining the above method with the techniques of Section 3.5, resulting to Theorem 2(f).

4 Data Structures

Now we deal with the data structure version of the join-reachability problem. Our goal is to construct an efficient data structure for $\mathcal{J} = \mathcal{J}(G_1, G_2)$ such that given a query vertex b it can report all vertices a satisfying $a \rightsquigarrow_{\mathcal{J}} b$. We state the efficiency of a structure using the notation $\langle s(n), q(n,k)\rangle$ which refers to a data structure with $O(s(n))$ space and $O(q(n,k))$ query time for reporting k elements. In order to design efficient join-reachability data structures we apply the techniques we developed in Section 3. The bounds that we achieve this way are summarized in the following theorem. The proof is given in the full version of the paper [6].

Theorem 3. *Given two digraphs G_1 and G_2 with n vertices we can construct join-reachability data structures with the following efficiency:*

(a) $\langle n, k\rangle$ *when G_1 is an unoriented tree and G_2 is an unoriented dipath.*
(b) $\langle n, \log n + k\rangle$ *when G_1 is an out-tree and G_2 is an unoriented tree.*
(c) $\langle n \log^{\varepsilon} n, \log\log n + k\rangle$ *(for any constant $\varepsilon > 0$), when G_1 and G_2 are unoriented trees.*
(d) $\langle n \log n, k \log n\rangle$ *when G_1 is planar digraph and G_2 is an unoriented tree.*
(e) $\langle n \log^2 n, k \log^2 n\rangle$ *when both G_1 and G_2 are planar digraphs.*
(f) $\langle n\kappa_1, k\rangle$ *when G_1 is a general digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is an unoriented tree.*
(g) $\langle n(\kappa_1 + \log n), k\kappa_1 \log n\rangle$ *or* $\langle n\kappa_1 \log n, k \log n\rangle$ *when G_1 is a general digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is planar digraph.*
(h) $\langle n(\kappa_1 + \kappa_2), \kappa_1\kappa_2 + k\rangle$ *or* $\langle n\kappa_1\kappa_2, k\rangle$ *when each G_i, $i = 1, 2$, is a digraph that can be covered with κ_i vertex-disjoint dipaths.*

5 Conclusions and Open Problems

We considered the computational and combinatorial complexity of the join-reachability graph, and the design of efficient join-reachability data structures for a variety of graph classes. We believe that several open problems deserve further investigation. For instance, from the combinatorial complexity aspect, it would be interesting to prove or disprove that an $O(m \cdot \text{polylog}(n))$ bound on the size of the join-reachability graph $\mathcal{J}(\{G_1, G_2\})$ is attainable when G_1 is a general digraph with n vertices and m arcs and G_2 is a dipath. Another direction is to consider the problem of approximating the smallest join-reachability graph for specific graph classes. From the data structures side, one can investigate how to support the following type of counting queries: Given a pair of query vertices compute the number of their common predecessors in $\mathcal{J}$. While some of our structures can be easily extended in order to support such counting queries, there are several cases (e.g., for planar digraphs) where we need to overcome various technical difficulties.

Acknowledgements. We would like to thank Li Zhang for several useful discussions.

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: SIGMOD 1989: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, pp. 253–262 (1989)
2. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. 1(2), 131–137 (1972)
3. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW 2001: Proceedings of the 10th International Conference on World Wide Web, pp. 613–622 (2001)
4. Georgiadis, L.: Computing frequency dominators and related problems. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 704–715. Springer, Heidelberg (2008)
5. Georgiadis, L.: Testing 2-vertex connectivity and computing pairs of vertex-disjoint s-t paths in digraphs. In: Proc. 37th Int'l. Coll. on Automata, Languages, and Programming, pp. 738–749 (2010)
6. Georgiadis, L., Nikolopoulos, S.D., Palios, L.: Join-reachability problems in directed graphs. Technical Report arXiv:1012.4938v1 [cs.DS] (2010)
7. Georgiadis, L., Tarjan, R.E.: Dominator tree verification and vertex-disjoint paths. In: Proc. 16th ACM-SIAM Symp. on Discrete Algorithms, pp. 433–442 (2005)
8. Kameda, T.: On the vector representation of the reachability in planar directed graphs. Information Processing Letters 3(3), 75–77 (1975)
9. Katriel, I., Kutz, M., Skutella, M.: Reachability substitutes for planar digraphs. Technical Report MPI-I-2005-1-002, Max-Planck-Institut Für Informatik (2005)
10. Talamo, M., Vocca, P.: An efficient data structure for lattice operations. SIAM J. Comput. 28(5), 1783–1805 (1999)
11. Tamassia, R., Tollis, I.G.: Dynamic reachability in planar digraphs with one source and one sink. Theoretical Computer Science 119(2), 331–343 (1993)
12. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004)
13. Wang, H., He, H., Yang, J., Yu, P.S., Yu, J.X.: Dual labeling: Answering graph reachability queries in constant time. In: ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering, p. 75 (2006)