T U M

I N S T I T U T F Ü R I N F O R M A T I K

All-Pairs Common-Ancestor Problems in Weighted Dags

Matthias Baumgart Stefan Eckhardt Jan Griebsch
Sven Kosub Johannes Nowak

TUM-I0606
April 06

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-04-I0606-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

All-Pairs Common-Ancestor Problems in Weighted Dags

Matthias Baumgart *Stefan Eckhardt* *Jan Griebsch*
Sven Kosub *Johannes Nowak*
Fakultät für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching, Germany
{baumgart,eckhardt,griebsch,kosub,nowakj}@in.tum.de

Abstract

This work considers the (lowest) common ancestor problem in weighted directed acyclic graphs. The minimum-weight (lowest) common ancestor of two vertices is the vertex among the set of (lowest) common ancestors with the smallest ancestral distance. For the all-pairs minimum-weight common ancestor problem we present an $O(nm)$ algorithm for arbitrary edge weights which is optimal for sparse graphs and an $O(n^{2.575})$ algorithm for dense graphs with moderately bounded edge weights based on matrix multiplication. The presented solutions to the all-pairs minimum-weight lowest common ancestor problem are based upon solutions of the all-pairs all lowest common ancestors problem in unweighted graphs, which represents an upper bound. For the all-pairs all lowest common ancestors problem we give an $O(nmk^2)$ algorithm, with k the bound on the maximum number of lowest common ancestors for pairs, and an $O(nm\,\mathrm{width}(G))$ algorithm, where $\mathrm{width}(G)$ is the size of the largest antichain (independent set) in the transitive closure of G. The ideas are applicable for fast matrix multiplication implying an $O(n^{3.616})$ algorithm.

1 Introduction

Directed acyclic graphs (dags) are powerful tools for modelling causality systems or other kinds of entity dependencies. If we think of causal relations among a set of events, natural questions come up, such as: Which events are entailed by two given events? What is the first event which is entailed by two given events? In dags, these questions can be answered by computing common ancestors (CAs), i.e., vertices that are reachable via any path from each of the given vertices, and computing lowest common ancestors (LCAs), i.e., those common ancestors that are not reachable from any other common ancestor of the two given vertices.

Although LCA algorithms for general dags are indispensable computational primitives, they have been found an independent subject of studies only recently [5, 16]. There is a lot of sophisticated work devoted to LCA computations for the special case of trees (see, e.g., [14, 21, 5]), but due to the limited expressive power of trees they are often applicable only in restrictive or over-simplified settings. In [5], a list of examples can be found where LCA queries on dags are necessary. We add two more applications.

A first one concerns *phylogenetic networks.* LCA algorithms have been frequently used in the context of phylogenetic trees, i.e., trees that depict the ancestor relations of species, genes, or features. Bacteria obtain a large portion of their genetic diversity through the acquisition of distantly related organisms, via horizontal gene transfer (HGT) or recombination. While views as to the extent of HGT and cross species recombination in bacteria differ, it is widely accepted that they are among the main processes driving prokaryotic evolution and are (with random mutations) mainly responsible for the development of antibiotic resistance. Such evolutionary developments cannot be modeled by trees, and thus there has been an increasing interest in phylogenetic dag networks and appropriate analysis methods [18, 17, 19]. Unfortunately, many of the established approaches from phylogenetic trees cannot trivially be extended to dags. This is particularly true for the computation of ancestor relationships.

A second application is related to *Network protocols.* Currently, Internet inter-domain routing is mainly done using the Border Gateway Protocol (BGP). BGP allows participating autonomous systems to announce and withdraw routable paths over physical connections with their neighbors. This process is governed by local routing policies which are rationally based on commercial relationships between autonomous systems. It has been recognized that, even if globally well-configurated, these local policies have a critical influence on routing stability and quality [13, 12]. An orientation of the underlying connectivity graph imposed by customer-to-provider relations can be viewed as a dag. Routes through the Internet have the typical structure of uphill and downhill parts to the left and right of a top provider in the middle of the route (see, e.g., [11]). Computing such top providers, which are just CAs and LCAs, is needed for reliability or efficiency analyses. In some experimental setting, we have a small Internet sample (taken from supplementary material for [23]) which constitutes a dag with 11,256 vertices and 13,655 edges. Finding top providers for each of the 63,343,140 pairs makes fast CA and LCA algorithms an issue.

In this paper, we continue the study of efficient LCA computation by considering the weighted problem versions, which are not covered by the recent work in [5, 16]. More precisely, we are interested in computing, on a given weighted dag, for each pair of vertices those CAs or LCAs that have in total the shortest distance to both vertices of the pair. We call these problems All-Pairs Min-Weight CA and All-Pairs Min-Weight LCA, respectively. Weights are of importance as most phylogenetic networks do define some type of evolutionary distances between species. In the Internet, weights model communication costs or traffic loads.

Results. We summarize the technical contributions of this paper.

All-Pairs Min-Weight CA: We use two types of algorithms—one based on dynamic programming and another based on matrix multiplication. Dynamic programming yields an $O(nm)$ algorithm for a dag having n vertices, m edges, and arbitrary edge weights. This algorithm has optimal running time on sparse dags. Using fast matrix multiplication for computing all-pairs shortest distances leads to an algorithm with time complexity of $O(n^{2.575})$ on dags with n vertices and small integer weights. To obtain this algorithm, we adapt techniques to identify witnesses for shortest paths (see [22, 27]).

All-Pairs Min-Weight LCA: We approach this problem by designing algorithms for the All-Pairs All LCA problem, i.e., the problem of computing for all pairs of vertices of a dag, the set of all LCAs. This problem is a computational upper bound. We describe a dynamic-programming-based algorithm for All-Pairs All LCA, generally achieving only the trivial upper bound $O(n^2m)$ but which allows modification to an $O(nmk^2)$ algorithm, where k is a bound on the maximum size of LCA sets. We further use Dilworth's theorem to devise a well-scaling algorithm with running time $O(nm\,\mathrm{width}(G))$, where $\mathrm{width}(G)$ is the size of the largest antichain (independent set) in the transitive closure of G. This approach is also applicable for fast matrix multiplication implying an $O(n^{3.616})$ algorithm. Although there is an information-theoretic gap of factor n between the matrix representations for All-Pairs Min-Weight LCA and All-Pairs All LCA (a $\Theta(n^2)$-space matrix versus a worst-case $\Theta(n^3)$-space matrix), we do not know how to avoid using All-Pairs All LCA.

Note that after computing the LCA matrices query times are all $O(1)$.

Related work. LCA algorithms have been extensively studied in the context of trees with most of the research rooted in [1, 24]. The first optimal algorithm for the all-pairs LCA problem in trees, with linear preprocessing time and constant query time, was given in [14]. The same asymptotics was reached using a simpler and parallelizable algorithm in [21]. Recently, a reduction to range minimum queries has been used to obtain a further simplification with optimal bounds on running time [5]. More algorithmic variants can be found in, e.g., [6, 26, 25, 7].

In the more general case of dags, a pair of nodes may have more than one LCA, which leads to the distinction of *representative* versus *all* LCA solutions. In early research both versions still coincide by considering dags with each pair having at most one LCA. Extending the work on LCAs in trees, in [20], an algorithm was described with linear preprocessing and constant query time for the LCA problem on arbitrarily directed trees (or, causal polytrees). Another solution was given in [2], where the representative problem in the context of object inheritance lattices was studied. The approach in [2], which is based on poset embeddings into boolean lattices yielded $O(n^3)$ preprocessing and $O(\log n)$ query time on lower semilattices.

The representative LCA problem on general dags has been recently studied in [5, 16]. Both works rely on fast matrix multiplications (currently the fastest known algorithm needs $\tilde{O}(n^\omega)$, with $\omega < 2.376$ [8]) to achieve $\tilde{O}(n^{\frac{\omega+3}{2}})$ [5] and $\tilde{O}(n^{2+\frac{1}{4-\omega}})$ [16] preprocessing time on dags with n nodes and m edges. For sparse dags, in [16], an $O(nm)$ algorithm has been presented as well.

2 Preliminaries

Let $G = (V, E)$ be a directed graph with a weight function $w \colon E \to \mathbb{R}$. Throughout this work we denote by n the number of vertices and by m the number of edges. G is a directed acyclic graph (dag) if and only if G contains no cycles. We say G is *unweighted* if $w : E \to c$ for some constant $c \in \mathbb{R}$ and *weighted* otherwise. Let $\mathrm{TC}(G)$ denote the transitive closure of G, i.e., the graph having an edge (u, v) if v is reachable from u over some directed path in G.

A dag $G = (V, E)$ imposes a partial ordering on the vertex set. Let N be a bijection from V into $\{1, \ldots, n\}$. N is said to be a *topological ordering* if $N(u) < N(v)$ whenever v is reachable from u in G. Such an ordering is consistent with the partial ordering of the vertex set imposed by the dag. A value $N(v)$ is said to be the *topological number* of v with respect to N. Observe that a graph G is a dag if and only if it allows some topological ordering (folklore result). Moreover, a topological ordering can be found in time $O(n + m)$ [9]: perform a depth-first-search on G and insert each finished vertex at the front of a linked list. Then, the order of the vertices in the list from left to right is a topological ordering. For technical reasons, we assume

$N(\textsc{nil}) = 0$ throughout this work. We usually consider dags equipped with some topological ordering. In such cases we often omit the ordering.

Let $G = (V, E)$ be a dag and $x, y, z \in V$. The vertex z is a *common ancestor* (CA) of x and y if both x and y are reachable from z, i.e., (z, x) and (z, y) are in the transitive closure of G. By $\mathrm{CA}(x, y)$, we denote the set of all CAs of x and y. A vertex z is a *lowest common ancestor* (LCA) of x and y if and only if $z \in \mathrm{CA}(x, y)$ and for each $z' \in V$ with $(z, z') \in E$ we have $z' \notin \mathrm{CA}(x, y)$. $\mathrm{LCA}(x, y)$ denotes the set of all LCAs of x and y.

3 The All-Pairs Minimum-Weight CA Problem

In this section we consider the problem of finding minimum-weight common ancestors for each pair of vertices. More specifically, we define the following problem:

Problem:	All-Pairs Min-Weight CA
Input:	A dag $G = (V, E)$ with weight function $w : E \to \mathbb{R}$
Output:	An array M of size $n \times n$ such that for all pairs of vertices $x, y \in V$, $M[x, y] = \operatorname{argmin}_{z \in \mathrm{CA}(x,y)} d_G(z, x) + d_G(z, y)$; if $\mathrm{CA}(x, y) = \emptyset$ for some vertex pair (x, y), then $M[x, y] = \textsc{nil}$. (Ties are arbitrarily broken.)

Note that we want to compute vertices and not only ancestral distances. By computing all-pairs shortest distances of G, this information is easily deduced from the matrix M.

3.1 Applying Dynamic Programming

The first approach is based on a simple dynamic-programming technique for the closely related All-Pairs Representative LCA problem [5]: on a given unweighted dag $G = (V, E)$, compute an array R of size $n \times n$ where $R[x, y] = z$ is an LCA of x and y, if one exists. Like the algorithms in [5, 16], our algorithm outputs the vertex z with the highest topological number $N(z)$ among all CAs as a representative. We will refer to a vertex z which has the maximal topological number $N(z)$ among all vertices in a set as the *rightmost* vertex.

Proposition 1. [5, 16] *Let $G = (V, E)$ be a dag and let N be a topological ordering. Furthermore, let $x, y \in V$ be vertices with a non-empty set of CAs. If $z \in V$ is the rightmost vertex in $\mathrm{CA}(x, y)$, then z is an LCA of x and y.*

Lemma 2 is more specific to dynamic programming solutions of ancestor problems.

Lemma 2. *Let $G = (V, E)$ be a dag and let $x, y \in V$ be any pair of vertices. Furthermore, let z be the rightmost LCA of x and y.*

1. *If $(x, y) \in \mathrm{TC}(G)$ then $z = x$.*
2. *If $(x, y) \notin \mathrm{TC}(G)$ then the following holds: Let $x_1, \ldots, x_k$ be the parents of x. Let $z_1, \ldots, z_k$ be the rightmost LCAs of the pairs $(x_1, y), \ldots, (x_k, y)$. Then z is the rightmost vertex in $\{z_1, \ldots, z_k\}$.*

Algorithm 1: ALL-PAIRS REPRESENTATIVE LCA

Input: A dag $G = (V, E)$

Output: An array R of size $n \times n$ where $R[x, y]$ is an LCA of x and y

1 **begin**
2   Initialize $R[x, y] \leftarrow$ NIL
3   Compute the transitive closure $\mathrm{TC}(G)$ of G
4   Compute a topological ordering N
5   **foreach** $v \in V$ *in ascending order of* $N(v)$ **do**
6   **foreach** $(v, x) \in E$ **do**
7   **foreach** $y \in V$ *with* $N(y) \geq N(v)$ **do**
8   **if** $(x, y) \in \mathrm{TC}(G)$ **then** $R[x, y] \leftarrow x$
9   **else if** $N(R[v, y]) > N(R[x, y])$ **then** $R[x, y] \leftarrow R[v, y]$
10   **end**
11   **end**
12   **end**
13 **end**

Proof. If $(x, y) \in \mathrm{TC}(G)$, x is the only LCA of x and y and hence the rightmost. This proves the first statement.

For the second statement, set $Z = \{z_1, \ldots, z_k\}$ where the z_i's are vertices as described, under the assumption that $(x, y) \notin \mathrm{TC}(G)$. We only have to show that $z \in Z$ since $Z \subseteq \mathrm{CA}(x, y)$. Suppose for the sake of contradiction that $z \notin Z$. Since $z \neq x$, there is a path from z to x which includes some parent x_ℓ, $1 \leq \ell \leq k$, of x. Observe that z is a CA of x_ℓ and y and z_ℓ is a CA of x and y. Recall that the rightmost CA is an LCA. Since $z \neq z_\ell$, we have $N(z) \neq N(z_\ell)$. Assume first that $N(z) > N(z_\ell)$. This contradicts the assumption that z_ℓ is the rightmost LCA of x_ℓ and y. On the other hand, if $N(z) < N(z_\ell)$, z cannot be the rightmost LCA of x and y. It follows that $z \in Z$. $\square$

Algorithm 1 is directly based on Proposition 1 and Lemma 2.

Theorem 3. *Algorithm 1 solves* ALL-PAIRS REPRESENTATIVE LCA *in time* $O(nm)$.

Proof. The correctness follows from Lemma 2. Observe that in Line 6 the rightmost LCA of (v, y) is already determined since the vertices are visited in topological order, i.e., all parents of v are already processed. The running time of the above algorithm is clearly $O(nm)$. The preprocessing steps can be implemented in time $O(nm)$. Then, every edge is considered exactly once and for each edge (v, x) the entry $R[x, y]$ is updated in constant time. This amounts also to $O(nm)$ total expense. $\square$

We turn our attention to ALL-PAIRS MIN-WEIGHT CA. The problem is related to computing the shortest ancestral distance of two vertices. Let $d_G(u, v)$ be the shortest distance of two vertices u, v in G. For two vertices x, y and their CA z, the ancestral distance of x and y with respect to z is $d_G(z, x) + d_G(z, y)$. The minimum-weight CA minimizes the ancestral distance. A naive solution is as follows:

1. Compute the all-pairs shortest distance matrix D of G.

Algorithm 2: All-Pairs Min-Weight CA

```
   Input: A dag G = (V, E) with a weight function w : E → ℝ
   Output: An array M of size n × n where M[x, y] is a minimum-weight CA of x and y
1  begin
2      Compute the all-pairs shortest-distance matrix D of G
3      Compute a topological ordering N
4      foreach (x, y) with D[x, y] < ∞ do
5          M[x, y] ← x
6      end
7      foreach v ∈ V in ascending order of N(v) do
8          foreach (v, x) ∈ E do
9              foreach y ∈ V with N(y) ≥ N(v) do
10                 if D[M[v, y], x] + D[M[v, y], y] < D[M[x, y], y] + D[M[x, y], y] then
11                     M[x, y] ← M[v, y]
12                 end
13             end
14         end
15     end
16 end
```

2. For each pair x, y choose z such that $D[z, x] + D[z, y]$ is minimized.

As the all-pairs shortest-distance matrix of a dag can be computed for arbitrary edge weights in time $O(nm)$ (see [9] for more details) and the second step takes time $O(n)$ for each pair once the shortest distances are known, the naive solution takes time $O(n^3)$.

We give an $O(nm)$ dynamic programming solution to All-Pairs Min-Weight CA which is similar to Algorithm 1. The following proposition describes the structure of the dynamic-programming matrix.

Proposition 4. *Let $G = (V, E)$ be a weighted dag and let x, y be two vertices that have at least one CA. Furthermore, let $x_1, \ldots, x_k$ be the parents of x and let $Z = \{z_1, \ldots, z_k\}$ be the set of the corresponding minimum-weight CAs of (x_i, y) for $1 \leq i \leq k$. Then, for the minimum-weight CA z of x and y it holds that $z = \operatorname{argmin}_{z' \in Z \cup \{x\}} d_G(z', x) + d_G(z', y)$.*

Theorem 5. *Algorithm 2 solves* All-Pairs Min-Weight CA *in time $O(nm)$.*

Proof. The analysis is similar to the proof of Theorem 3. The correctness follows from Proposition 4: since vertices are visited in ascending topological order, the minimum weight CA of (v, y) has already been determined at the first visit of edge (v, y) in Line 8. Finally, the running time is clearly $O(nm)$, because Line 2 can be achieved in time $O(nm)$, as mentioned before. ☐

3.2 Applying Fast Matrix-Multiplication

As mentioned above, there is a close relationship between minimum-weight CAs and shortest ancestral distances. Moreover, it is possible to reduce the problem of computing shortest ancestral distances to the All-Pairs Shortest-Distance problem in dags. The reduction is inspired by ideas from [5]. There are classes of dags on which All-Pairs Shortest-Distance can be solved in time $o(nm)$. As an example of such a class, consider dags with small integer weights and $m = \omega(n^{1+\mu})$, where μ satisfies $\omega(1, \mu, 1) = 1 + 2\mu$ and $\omega(1, \mu, 1)$ is the exponent of the algebraic multiplication of an $n \times n^\mu$ matrix with an $n^\mu \times n$ matrix. Currently, the best available bounds [8] imply $\mu < 0.575$ (see [27] for more details). However, it is not obvious how to derive the minimum-weight CAs from shortest ancestral distances. Our approach is based on ideas in [22, 27] used for computing witnesses for shortest paths. A detailed description of the reduction and the identification of minimum-weight CAs is given in the proof of Theorem 6

Theorem 6. *Let $\mathcal{A}$ be any* All-Pairs Shortest-Distance *algorithm with running time $t_\mathcal{A}(n, m)$ on weighted dags with n vertices and m edges. Then, there is an algorithm for* All-Pairs Min-Weight CA *with running time $\tilde{O}(t_\mathcal{A}(n, m) + n^2)$. Here, $t_\mathcal{A}$ is required to satisfy $t_\mathcal{A}(O(n), O(m)) = O(t_\mathcal{A}(n, m))$.*

Proof. Let $G = (V, E)$ be a dag with weight function $w : E \to \mathbb{N}$. Let $\overline{G} = (\overline{V}, \overline{E})$ be the *reflected graph* of G with weight function $\overline{w} : \overline{E} \to \mathbb{N}$, where for each $x \in V$ there is a vertex $\overline{x} \in \overline{V}$, $(\overline{x}, \overline{y}) \in \overline{E} \Leftrightarrow (y, x) \in E$, and $\overline{w}(\overline{x}, \overline{y}) = w(y, x)$. Further, let $G' = (V', E')$ be the following graph with weight function w': set $V = \{1, \ldots, n\}$ and set $\overline{V} = \{2n + 1, \ldots, 3n\}$, i.e., $2n + k \in \overline{V}$ is the corresponding vertex of $k \in V$ for $1 \leq k \leq n$. Let additionally $V_C = \{n + 1, \ldots, 2n\}$ be a set of vertices representing edges between V and $\overline{V}$. Define $E_C = \{\ (i, n + i) \mid 1 \leq i \leq n\ \} \cup \{\ (n + i, 2n + i) \mid 1 \leq i \leq n\ \}$. Then,

$$\begin{aligned} V' &=_{\text{def}} V \cup V_C \cup \overline{V}, \\ E' &=_{\text{def}} E \cup \overline{E} \cup E_C, \end{aligned}$$

and

$$w'(x, y) =_{\text{def}} \begin{cases} w(x, y) & \text{if } (x, y) \in E, \\ \overline{w}(x, y) & \text{if } (x, y) \in \overline{E}, \\ 0 & \text{if } (x, y) \in E_C. \end{cases}$$

The graph G' (see Figure 1) is structurally similar to the graph used in [5] for computing representative LCAs. However, we represent edges between vertex pairs $(u, \overline{u})$ by additional vertices and use original edge weights. It is easy to see that

$$d_{G'}(i, 2n + j) = \min_{z \in \text{CA}(i,j)} d_G(z, i) + d_G(z, j),$$

i.e., the distance between i and $2n + j$ in G' equals the ancestral distance between i and j in G for all vertices $i, j \in V$. Obviously, the graph G' can be constructed in time $O(n + m)$.

Now let $\mathcal{A}$ be an All-Pairs Shortest-Distance algorithm for weighted dags. Given a dag $G = (V, E)$ we can construct the graph G', solve All-Pairs Shortest-Distance on G' using Algorithm $\mathcal{A}$ and then construct the $n \times n$ ancestral-distance matrix D of G with

$$D[i, j] = d_{G'}(i, 2n + j)$$

for all $i, j \in V$. The overall process takes time $O(t_\mathcal{A}(3n, 2m + 2n))$. However, we cannot directly read off the minimum weight ancestors themselves. Instead, we make use of techniques

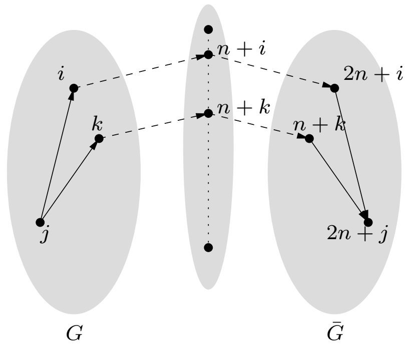

Figure 1: Solving All-Pairs Min-Weight CA with All-Pairs Shortest-Distance. Dashed lines have weight zero.

described in [27] to compute witnesses for distance products. This only adds a polylogarithmic factor to the overall running time.

The main idea behind the witnesses construction is the following: Suppose, G'' is a graph in which one of the vertices from V_C, say the vertex $n+k$, $1 \leq k \leq n$, is contained, but all other vertices from V_C and all their adjacent edges are deleted. Further, let D' and D'' be the distance matrices of G' and G'', respectively. Apparently, for all pairs i, j with $1 \leq i, j \leq n$, $d_{G''}(i, 2n+j) = d_{G'}(i, 2n+j)$ implies that k lies on some shortest path from i to $2n+j$ in G' and, by construction of G', that k lies on some shortest ancestral path in G. Thus, we could find all pairs $i, j \in V$ such that k is minimum-weight CA of i and j by solving the All-Pairs Shortest-Distance probem on G'' and comparing D'' with D', element by element. Alas, trying all vertices from $V_C = \{n+1, \ldots, 2n\}$ one by one would yield $O(n)$ All-Pairs Shortest-Distance computations on a graph on $\Omega(n)$ vertices with $\Omega(m)$ edges.

In the following we outline a randomized approach. Assume that a pair $i, j \in V$ has exactly c minimum weight CAs. We first describe how to sample a subset $V_C' \subseteq V_C$ (which represents the candidates for minimum weight CAs in G) such that there is only one candidate in V_C' for the pair i, j with constant positive probability. Let $d \in \mathbb{N}$ satisfy $\frac{n}{2} \leq c \cdot d \leq n$ and suppose that we draw V_C' as a multiset of size d uniformly at random from $\{n+1, \ldots, 2n\}$ with repetition. The probability of choosing exactly one of the c ancestors into V_C' is greater than $\frac{1}{2e}$. This holds for each pair which has c minimum weight CAs such that $\frac{n}{2} \leq c \cdot d \leq n$, independently. By amplification, we increase the probability for each pair to $1 - \frac{1}{n}$. Construct $\lceil (\log \frac{2e}{2e-1})^{-1} \log n \rceil$ such candidate multisets V_C' of size d each. Then, the probability that in none of these sets there is exactly one of the c minimum weight CAs is less than $(1 - \frac{1}{2e})^{\lceil (\log \frac{2e}{2e-1})^{-1} \log n \rceil} \leq \frac{1}{n}$, for each pair independently. Finally, letting d range from 1 to $\lceil \log n \rceil$ makes sure that the condition $\frac{n}{2} \leq c \cdot d \leq n$ is satisfied for each pair with c minimum weight CAs at least once with the desired probability.

Next we show how to find all unique minimum weight CAs in a candidate set V_C'. For all pairs $i, j \in V$ such that there is exactly one minimum weight CA k in V_C', we can construct k deterministically as follows: for all $1 \leq \ell \leq \lceil \log n \rceil$, let I_ℓ be the set of all indices $n+k$ in V_C' such that the ℓ-th bit in the binary representation of k is one. Further, let $G^{(\ell)}$ be the graph G', containing all vertices $v \in V_C' \cap I_\ell$, but with all other vertices from V_C and all their adjacent edges *removed*. Let $D^{(\ell)}$ denote the distance matrix of $G^{(\ell)}$. Again, $D^{(\ell)}[i, j] = D'[i, j]$ if and only if the ℓ-th bit of k is one. Also, the (unique) minimum weight CA of i and j in G must have its ℓ-th bit equal to one, too. Hence all unique minimum-weight CAs in V'_C can be found by computing $\lceil \log n \rceil$ distance matrices $D^{(\ell)}$ and comparing them to D.

Thus, by choosing $\lceil (\log \frac{2e}{2e-1})^{-1} (\log n)^2 \rceil$ candidate sets each of size at most n and carrying out $O(\log n)$ All-Pairs Shortest-Distance computations for each set, it follows that the probability that the number of pairs for which no candidate is found is greater than $\lceil (\log n)^3 \rceil$ is less than γ^n for some positive $\gamma < 1$ by applying a Chernoff bound. For the remaining such pairs, we simply test all possible ancestors. This postprocessing takes time $O((\log n)^3 \cdot n^2)$ in expectation. The above can be derandomized by the method of *c-wise ε-independent random variables* (see [3] for more details). The derandomized approach takes $O((\log n)^6)$ All-Pairs Shortest-Distance computations and the theorem follows. □

Corollary 7. All-Pairs Min-Weight CA *can be solved in time $\tilde{O}(n^{2+\mu})$, restricted to dags with integer weights from the integer interval $(-n^{3-\omega}, n^{3-\omega})$, where μ satisfies $\omega(1, \mu, 1) = 1+2\mu$ and $\omega = \omega(1, \mu, 1)$ is the exponent of the algebraic multiplication of an $n \times n^{\mu}$ matrix with an $n^{\mu} \times n$ matrix.*

From the above-mentioned, currently best bound for μ, we obtain an $O(n^{2.575})$ algorithm for All-Pairs Min-Weight CA for moderate integral edge weights.

4 The All-Pairs Min-Weight LCA Problem

In this section we consider the minimum-weight lowest common ancestor problem.

Problem:	All-Pairs Min-Weight LCA
Input:	A dag $G = (V, E)$ with weight function $w : E \to \mathbb{R}$
Output:	An array L of size $n \times n$ such that for all pairs of vertices $x, y \in V$, $L[x, y] = \operatorname{argmin}_{z \in \mathrm{LCA}(x,y)} d_G(z, x) + d_G(z, y)$; if $\mathrm{LCA}(x, y) = \emptyset$ for some vertex pair (x, y), then $L[x, y] = $ nil. (Ties are arbitrarily broken.)

4.1 Solving the All-Pairs All LCA Problem

A generic solution for finding minimum-weight lowest common ancestors uses a solution of the All-Pairs All LCA problem which is defined as follows: on a given dag $G = (V, E)$, compute an array A of size $n \times n$ such that $A[x, y] = \mathrm{LCA}(x, y)$. We suppose that LCA sets are stored as lists in the matrix A. We give the following generic solution for All-Pairs Min-Weight LCA:

1. Solve All-Pairs Shortest-Distance on dag G (understood as a weighted dag).
2. Solve All-Pairs All LCA on dag G (understood as an unweighted dag).
3. For each pair (x, y) choose in time $O\big(\|\mathrm{LCA}(x, y)\|\big)$ the vertex $z \in \mathrm{LCA}(x, y)$ which minimizes the ancestral distance $d_G(z, x) + d_G(z, y)$ and set $L[x, y] = z$.

The main part of the forthcoming is dedicated to the problem of finding all LCAs of all vertex pairs. From a solution to this problem, the minimum-weight LCA can be readily derived in time $O\big(\sum_{x,y \in V} \|\mathrm{LCA}(x, y)\|\big)$. A trivial lower bound for All-Pairs All LCA is $\Omega(n^3)$, even for dags with $m = O(n)$, as can be seen in Figure 2.

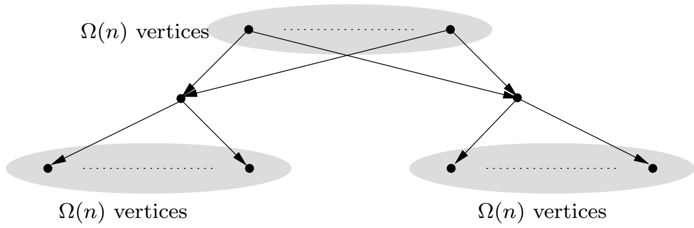

Figure 2: Sparse dags with an $\Omega(n^3)$ total number of LCAs

Theorem 8. *Let $\mathcal{A}$ be an algorithm that solves* All-Pairs All LCA *in time $t_{\mathcal{A}}(n, m)$. Then,* All-Pairs Min-Weight LCA *can be solved in time $O(t_{\mathcal{A}}(n, m))$.*

Since all upper bounds for algorithms for All-Pairs All LCA apply to algorithms for All-Pairs Min-Weight LCA, we concern ourselves with All-Pairs All LCA solely. Note that in contrast, it is not clear whether it is possible to compute minimum-weight LCAs without implicitly solving All-Pairs All LCA.

The definition of LCAs immediately yields several $O(n^2m)$ algorithms. A rather trivial one, first, computes the transitive closure of G in $O(nm)$. Then, it determines for every vertex z and every pair (x, y) in time $O(\text{out-deg}(z))$ if z is an LCA of (x, y). Amazingly, this trivial algorithm is optimal on sparse dags.

We proceed by giving an $O(n^2m)$ dynamic programming approach which is more suitable for later use. This algorithm adopts ideas from Algorithm 1. Recall that $z \in \text{CA}(x, y)$ is an LCA of x and y if there is no other vertex $z' \in \text{CA}(x, y)$ such that $(z, z') \in \text{TC}(G)$. The following is a generalization of Lemma 2.

Lemma 9. *Let $G = (V, E)$ be any dag. Let x and y be vertices of G. Let $x_1, \ldots, x_k$ be the parents of x and S the union of the sets $\text{LCA}(x_\ell, y)$ for all $1 \leq \ell \leq k$.*

1. *If $(x, y) \in \text{TC}(G)$ then $\text{LCA}(x, y) = \{x\}$.*
2. *If $(x, y) \notin \text{TC}(G)$ then $\text{LCA}(x, y) \subseteq S$. More specifically, for all $v \in V$ it holds that $v \in \text{LCA}(x, y)$ if and only if $v \in S$ and for all $v' \in S$, $(v, v') \notin \text{TC}(G)$.*

Proof. Case 1 is trivial. For Case 2 let z be an arbitrary LCA of x and y. Observe that all vertices in S are CAs of x and y. Hence, we only have to show that $z \in S$. Suppose for the sake of contradiction that $z \notin S$. Then, there is a path from z to x through some parent x_ℓ of x. Thus, z is a CA of x_ℓ and y. By assumption, z is not an LCA of x_ℓ and y. This implies, that there is a witness z_ℓ which is an LCA of x_ℓ and y and which is reachable from z. Yet as stated above, every CA of x_ℓ and y is also a CA of x and y. Hence, z_ℓ is also a witness to the fact that z is not an LCA of x and y contradicting our assumption. $\square$

Lemma 9 is implemented by Algorithm 3 in the following way. The set $\text{LCA}(x, y)$ is iteratively constructed by merging the sets $\text{LCA}(x_\ell, y)$. In the merging steps, all those vertices are discarded that are predecessors of some other vertices in the set.

Since the vertices are visited in increasing order with respect to a topological ordering N, the sets $A[v, y]$ are finally determined by the time that v is visited. All parents of v are visited before x. This establishes the correctness of Algorithm 3.

Proposition 10. *Let $t_{\text{merge}}(n_1, n_2)$ be an upper bound for the time needed by one merge operation on sets of sizes n_1 and n_2. Then, Algorithm 3 takes time $O(nm\, t_{\text{merge}}(n, n))$.*

Algorithm 3: All-Pairs All LCA

```
   Input: A dag G = (V, E)
   Output: An array A of size n × n where A[x, y] is the set of all LCAs of x and y
1  begin
2      Compute the transitive closure TC(G) of G
3      Compute a topological ordering N
4      foreach v ∈ V in ascending order of N(v) do
5          foreach y ∈ V with N(v) < N(y) do
6              if (v, y) ∈ TC(G) then A[v, y] ← {v}
7          end
8          foreach (v, x) ∈ E do
9              foreach y ∈ V with N(x) < N(y) do
10                 if (x, y) ∈ TC(G) then A[x, y] ← {x}
11                 else A[x, y] ← Merge(A[v, y], A[x, y])
12             end
13         end
14     end
15 end
```

Next we show how to implement the merge operation in Line 15. Let S_1 and S_2 be the two sets that are to be merged into one set S. Recall that a vertex $s_1 \in S_1$ is retained in S if there is no vertex $s_2 \in S_2$ that is reachable from s_1 and vice versa. Hence, naively, we could simply check this by a transitive closure look-up. To this end, a merge operation takes time $O\big(\|S_1\| \cdot \|S_2\|\big)$. For large sets, this can be improved to $O(n)$ as follows. Along with each set S keep a forbidden set $\tilde{S}$, where $v \in \tilde{S}$ if and only if there exists a vertex $s \in S$ such that s is reachable from v. Then, a vertex $s_1 \in S_1$ is retained if $s_1 \notin \tilde{S_2}$. If the forbidden sets are maintained as bitvectors, this can be checked in constant time and two sets S_1 and S_2 can be merged in time $O\big(\|S_1\| + \|S_2\|\big)$ to a new set S. The bottleneck of this merge operation is updating the forbidden set $\tilde{S}$ which is done by a bitwise-or combination of $\tilde{S_1}$ and $\tilde{S_2}$, thus, the merge operation takes time $\Theta(n)$.

Corollary 11. *Algorithm 3 using refined merging solves* All-Pairs All LCA *in time $O(n^2m)$.*

The advantage of this dynamic-programming-based algorithm over the trivial one is that if we can upper bound the size k of the sets $\mathrm{LCA}(x, y)$ for each pair of two vertices $x, y \in V$ then we have a better upper bound on the running time whenever $k = o(\sqrt{n})$. In such cases we use naive merging.

Corollary 12. *Algorithm 3 can be modified such that it solves the* All-Pairs All LCA *problem in time $O\left(nmk^2\right)$ where k is the maximum cardinality of LCA sets.*

Note that if we do not know k in advance, we can decide online which merging strategy to use, without changing the asymptotical run-time: start Algorithm 3 with naive merging until a vertex is reached in Line 4 having more LCAs with some neighbor vertex (Line 10) than prescribed by some threshold. If this happens start the algorithm anew with refined merging.

As an immediate consequence we obtain fast algorithms for testing lattice-theoretic properties of posets represented by dags.

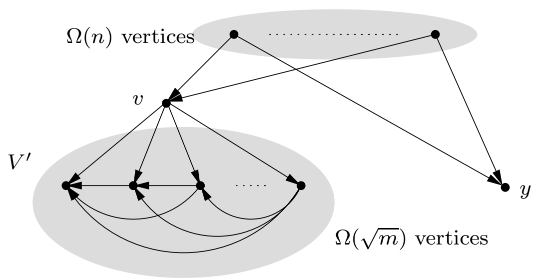

Figure 3: Bad graph for naive merging

Corollary 13. *Testing whether a given dag is a lower semilattice, an upper semilattice, or a lattice can be done in time $O(nm)$.*

Naturally, in this context, the question arises if it is advantageous to use naive merging if the average cardinality $\overline{k}$ of the LCA sets is bounded. That is, is the running time of Algorithm 3 in $o(n^2m)$ in this case? However, even for $\overline{k} \leq c$ for a constant c, the answer is "no", as the graph in Figure 3 shows. After processing the vertex v the sets $\mathrm{LCA}(w, y)$ are of size $\Omega(n)$ for all $w \in V'$. Next, we consider only merging steps between two vertices of V'. Observe that we merge along edges. Since there are $\Omega(m)$ edges among vertices of V', these merge operations alone amount to $\Omega(n^2m)$ if implemented naively. On the other hand, it is easy to see that the average size of the LCA sets is bounded by a constant in this example.

To summarize, we conclude with algorithmic bounds for the minimum-weight LCA problem. Note that no restrictions on edge weights are made.

Corollary 14. All-Pairs Min-Weight LCA *can be solved in time $O(\min\{n^2m, nmk^2\})$ where k is the maximum cardinality of all LCA sets.*

4.2 Scaling with Maximum Antichains

Let $G = (V, E)$ be a dag. Let V' be a subset of V. We call V' an *antichain* if no vertex in V' is reachable from another vertex in V'. That means that no two vertices of V' are comparable with respect to the partial order imposed by a dag. A maximum antichain of G is a set $V' \subseteq V$ such that V' is an antichain of maximal cardinality. The *width* of a dag G, denoted by $\mathrm{width}(G)$, is the size of a maximum antichain in dag G. Observe that a maximum antichain is a maximum independent set of the transitive closure. Moreover, the sets $\mathrm{LCA}(x, y)$ naturally are antichains by definition. In particular, their sizes are bounded from above by $\mathrm{width}(G)$.

In contrast to Algorithm 3, which scales quadratically with the size of LCA sets, we give an algorithm for All-Pairs All LCA that scales linearly with the width of dags. It is based on solutions for All-Pairs Representative LCA. We outline our approach.

Suppose we are given a vertex z and want to determine all pairs x, y for which z is an LCA. To this end, we employ an All-Pairs Representative LCA algorithm on G. Obviously, if z is a representative LCA of x, y then $z \in \mathrm{LCA}(x, y)$. Thus, if we could force the All-Pairs Representative LCA algorithm to return z as a representative LCA for x, y whenever z is an LCA of x and y, we could answer the above question by solving the representative LCA problem. This can be done as follows.

For a dag $G = (V, E)$ and a vertex $z \in V$, let $N^*(z)$ denote the maximal number of z in any topological ordering of G. It is easily seen that a topological ordering N satisfies $N(z) = N^*(z)$

if and only if for all $x \in V$ such that $N(x) \geq N(z)$, x is reachable from z. This immediately implies a linear-time algorithm to find a corresponding ordering.

Proposition 15. *A topological ordering realizing $N^*(z)$ for any vertex z in a dag can be computed in time $O(n + m)$.*

Proof. Given a dag $G = (V, E)$, remove vertex set $\{z\} \cup \{ x \mid x \text{ is reachable from } z \text{ in } G \}$ from V, topologically sort the dag (via DFS) induced by the remaining vertices arbitrarily, topologically sort the dag induced by the removed vertices arbitrarily (via DFS), and concatenate both vertex lists each of which is ordered ascendingly with respect to the topological numbers. □

If we fix a vertex z's number maximizing topological ordering, then z is the rightmost CA of all vertex pairs (x, y) such that $z \in \mathrm{LCA}(x, y)$. Now clearly, our strategy is to iterate for each $x \in V$ over the orderings that maximize $N^*(x)$. Note that the $o(n^3)$ algorithms in [5, 16] as well as Algorithm 1 naturally return the vertex z with the highest number $N(z)$ (for a fixed topological ordering) among all LCAs of any pair (x, y). This leads to Algorithm 4 and Theorem 16.

Theorem 16. *Algorithm 4 solves* All-Pairs All LCA *in time $\tilde{O}(\min\{n^{3+\frac{1}{4-\omega}}, n^2 m\})$, where ω is the exponent of the algebraic multiplication of two $n \times n$ Boolean matrices.*

Proof. The first bound is from [16]. The second bound is from the dynamic programming approach in Corollary 11. □

As it is currently known that $\omega < 2.376$, we have an $O(n^{3.616})$ algorithm on dense dags.

A key observation is that an algorithm for All-Pairs Representative LCA that outputs, with respect to a fixed topological ordering N, the vertex with the highest number as a representative LCA, does it for all z with $N(z) = N^*(z)$ in parallel. We aim at maximizing topological numbers simultaneously for as many vertices as possible. This can easily be reached for vertices in paths.

Proposition 17. *A topological ordering maximizing $N^*(z)$ for all vertices z in any path p of a dag G simultaneously, can be computed in time $O(n+m)$. Moreover, this can be done for all vertex subsets of the given path p.*

Proof. Iteratively use the algorithm described in the proof of Proposition 15 starting at the tail of the path and going to the head. In case of a vertex subset, iterate only over vertices from the given set. □

This proposition implies that given such an ordering, it is possible to process a path p in only one iteration of the algorithm, i.e., only one call of an algorithm for All-Pairs Representative LCA is needed for the vertices in p. Thus, we can reduce the running time if we minimize the number of paths to be processed.

For a dag $G = (V, E)$, a *path cover* $\mathcal{P}$ of G is a set of paths in G such for every $v \in V$ there exists at least one path $p \in \mathcal{P}$ such that v lies on p. A minimal path cover is a path cover $\mathcal{P}$ such that $\|\mathcal{P}\|$ is minimized. We briefly sketch how to find a minimal path cover in a dag $G = (V, E)$. First, we compute the transitive closure of G in time $O(n^{2.376})$ and construct a bipartite graph $G' = (V_1 \uplus V_2, E)$ with $V_1 = V_2 = V$ and $E = \{\{v, w\} \mid v \in V_1,\ w \in V_2 \text{ and } (v, w) \in \mathrm{TC}(G)\}$.

Algorithm 4: ALL-PAIRS ALL LCA using LCA representatives

Input: A dag $G = (V, E)$
Output: An array A of size $n \times n$ where $A[x, y]$ is the set of all LCAs of x and y

begin
  foreach $z \in V$ **do**
    Compute a topological ordering N such that $N(z)$ is maximal
    Solve ALL-PAIRS REPRESENTATIVE LCA using any algorithm that returns the LCA with highest topological number as representative and get array R
    foreach (x, y) *with* $R[x, y] = z$ **do** $A[x, y] \leftarrow A[x, y] \cup \{z\}$ (by multiset-union)
  end
  Remove elements of multiplicity greater than one from $A[x, y]$ for all $x, y \in V$
end

A maximum bipartite matching M in G' can be found in time $O(n^{2.5})$ applying the Hopcroft-Karp algorithm [15]. We use M to construct a path cover by repeatedly exploring along the matching edges starting at the unmatched vertices of V_2. For more details see [4] where another algorithm for minimal path covers with running time $O(nm)$ using dynamic trees is described. Thus, minimal path covers of dags can be computed in time $O(\min\{nm, n^{2.5}\})$. This suggests Algorithm 5. Note that we do not compute an exact path cover of G but rather of $\mathrm{TC}(G)$, which is enough for our purposes by Proposition 17.

Actually, Algorithm 5 is an improvement over Algorithm 4 if we know that, on the one hand, the size of a minimal path cover is an upper bound on LCA set-sizes, and at most n on the other hand. Fortunately, the famous Dilworth's Theorem [10] does exactly this.

Lemma 18. [10] *Let G be any dag. Then,* $\mathrm{width}(G)$ *equals the size of a minimal path cover of G.*

Indeed, we need both directions of Dilworth's theorem to obtain the following theorem.

Theorem 19. *Algorithm 5 solves* ALL-PAIRS ALL LCA *in* $\tilde{O}\big(\min\{n^{2+\frac{1}{4-\omega}} \cdot \mathrm{width}(G), nm \cdot \mathrm{width}(G)\}\big)$ *time where ω is the exponent of the algebraic multiplication of two $n \times n$ Boolean matrices.*

Algorithm 5 elegantly scales with dag widths automatically without saying which algorithmic branch should be used as it is necessary for scaling with maximum LCA set-sizes. However, Theorem 19 does not yield as many benefits as may be expected. For instance, rooted binary trees can be viewed as dags. The width of a tree is the number of its leaves which is in fully binary trees $\Omega(n)$. In contrast to this, each pair has exactly one LCA. As another example, in the experimental setting of Internet dags mentioned in the introductory section, we obtained a width of 9,604 (i.e., there is an antichain containing around 85% of all vertices), a maximum LCA set-size of 27, and an average LCA set-size of 9.66. All this shows that improving our algorithms towards a linear-scaling behavior with respect to LCA set-sizes is essential.

To sum up we conclude with algorithmic bounds for the minimum-weight LCA problem. Note that the weight restrictions in the second part of the statement are caused by computing all-pairs shortest distances using fast matrix multiplication.

Algorithm 5: All-Pairs All LCA using LCA representatives (improved)

Input: A dag $G = (V, E)$
Output: An array A of size $n \times n$ where $A[x, y]$ is the set of all LCAs of x and y

begin
  Compute a transitive closure $\mathrm{TC}(G)$ of G
  Compute a minimal path cover $\mathcal{P}$ of $\mathrm{TC}(G)$
  foreach $p \in \mathcal{P}$ **do**
    Compute a topological ordering N such that $N(z)$ is maximal for all vertices of p
    Solve All-Pairs Representative LCA with respect to N and get array R
    foreach (x, y) *with* $R[x, y] = z$ *and* $z \in P$ **do**
      $A[x, y] \leftarrow A[x, y] \cup \{z\}$ (by multiset-union)
    end
  end
  Remove elements of multiplicity greater than one from $A[x, y]$ for all $x, y \in V$
end

Corollary 20. All-Pairs Min-Weight LCA *can be solved in time* $\tilde{O}\big(\min\{n^{2+\frac{1}{4-\omega}} \cdot \mathrm{width}(G), nm \cdot \mathrm{width}(G)\}\big)$ *where* ω *is the exponent of the algebraic matrix multiplication of two* $n \times n$ *Boolean matrices. For the first bound, i.e., on dense dags, edge weights are limited to the integer interval* $(-n^{3-\omega'}, n^{3-\omega'})$*, where* $\omega' = \omega(1, \mu, 1)$ *is the exponent of the multiplication of an* $n \times n^{\mu}$ *matrix with an* $n^{\mu} \times n$ *matrix and* μ *satisfies* $\omega(1, \mu, 1) = 1 + 2\mu$*.*

5 Conclusion and Open Problems

We have described and efficiently solved all-pairs ancestor problems on weighted dags both for sparse and dense instances. All-Pairs Min-Weight CA is widely understood. On sparse graphs our solution is optimal, and on dense graphs the gap between All-Pairs Shortest Distance and All-Pairs Min-Weight CA is shown to be at most polylogarithmic. On the other hand, our algorithms for All-Pairs Min-Weight LCA exhibit nice scaling properties. Moreover, upper bounds for the scaling factors beautifully coincide by Dilworth's theorem. However, we are left with intriguing open questions:

1. Is it possible to solve the All-Pairs Min-Weight LCA problem without solving the All-Pairs All LCA problem?
2. Can we devise an algorithm that scales linearly in the size k of the maximal LCA set?

Acknowledgements. We thank Benjamin Hummel for providing us with experimental data on Internet graphs.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. On finding lowest common ancestors in trees. *SIAM Journal on Computing*, 5(1):115–132, 1976.

[2] H. Aït-Kaci, R. S. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, 1989.

[3] N. Alon and M. Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4–5):434–449, 1996.

[4] A. A. Benczúr, J. Förster, and Z. Király. Dilworth's theorem and its application for path systems of a cycle - implementation and analysis. In *Proceedings of the 7th Annual European Symposium on Algorithms (ESA'99)*, volume 1643 of *Lecture Notes in Computer Science*, pages 498–509. Springer-Verlag, Berlin, 1999.

[5] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.

[6] O. Berkman and U. Vishkin. Finding level-ancestors in trees. *Journal of Computer and System Sciences*, 48(2):214–230, 1994.

[7] R. Cole and R. Hariharan. Dynamic LCA queries on trees. *SIAM Journal on Computing*, 34(4):894–923, 2005.

[8] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[10] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[11] L. Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[12] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.

[13] T. G. Griffin and G. T. Wilfong. An analysis of BGP convergence properties. *ACM SIGCOMM Computer Communication Review*, 29(4):277–288, 1999.

[14] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[15] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[16] M. Kowaluk and A. Lingas. LCA queries in directed acyclic graphs. In *Proceedings of 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 241–248. Springer-Verlag, Berlin, 2005.

[17] B. M. E. Moret, L. Nakhleh, T. Warnow, C. R. Linder, A. Tholse, A. Padolina, J. Sun, and R. E. Timme. Phylogenetic networks: Modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004.

[18] L. Nakhleh, J. Sun, T. Warnow, C. R. Linder, B. M. E. Moret, and A. Tholse. Towards the development of computational tools for evaluating phylogenetic network reconstruction methods. In *Proceedings of the 8th Pacific Symposium on Biocomputing (PSB 2003)*, pages 315–326. World Scientific Publishing, Singapore, 2003.

[19] L. Nakhleh and L.-S. Wang. Phylogenetic networks: Properties and relationship to trees and clusters. In *Transactions on Computational Systems Biology II*, volume 3680 of *Lecture Notes in Computer Science*, pages 82–99. Springer-Verlag, Berlin, 2005.

[20] M. Nykänen and E. Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Information Processing Letters*, 50(1):307–310, 1994.

[21] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[22] R. Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[23] L. Subramanian, S. Agarwal, J. Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'02)*, pages 618–627. IEEE Computer Society Press, Washington, D.C., 2002.

[24] R. E. Tarjan. Applications of path compression on balanced trees. *Journal of the ACM*, 26(4):690–715, 1979.

[25] B.-F. Wang, J.-N. Tsai, and Y.-C. Chuang. The lowest common ancestor problem on a tree with an unfixed root. *Information Sciences*, 119(1–2):125–130, 1999.

[26] Z. Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Information Processing Letters*, 51(1):11–16, 1994.

[27] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.