TUM

INSTITUT FÜR INFORMATIK

Fast Lowest Common Ancestor Computations in Dags

Stefan Eckhardt Andreas M. Mühling Johannes Nowak

TUM-I0707
Januar 07

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-01-I0707-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Fast Lowest Common Ancestor Computations in Dags

Stefan Eckhardt *Andreas M. Mühling* *Johannes Nowak*
Fakultät für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
{eckhardt,muehling,nowakj}@in.tum.de

Abstract

This work studies lowest common ancestor problems in directed acyclic graphs. We present fast algorithms for solving the All-Pairs Representative LCA and All-Pairs All LCA problems with expected running time of $O(n^2 \log n)$ and $O(n^3 \log \log n)$ respectively. The speed-ups over recently developed methods are achieved by applying transitive reduction on the input dags. The algorithms are experimentally evaluated against previous approaches demonstrating a significant improvement. On the purely theoretical side, we improve the upper bound for All-Pairs Representative LCA to $O(n^{2.575})$ and the upper bound for All-Pairs All LCA to $O(n^{3.3399})$. We give first fully dynamic algorithms for both All-Pairs Representative LCA and All-Pairs All LCA . Here, the update complexities are $O(n^{2.5})$ and $O(n^3)$ respectively, with constant query times.

1 Introduction

Causality systems or other kinds of entity dependencies are naturally modeled by directed acyclic graphs (dags). Thinking of causal relations among a set of events, natural questions come up, such as: Which event entails two given events? What is the last event which entails two given events? Transferring these questions to dags, answers are found by computing common ancestors (CAs), i.e., vertices that reach via any path each of the given vertices, and computing lowest common ancestors (LCAs), i.e., those common ancestors that do not reach 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 [7, 6, 17]. There is a lot of sophisticated work devoted to LCA computations for the special case of trees (see, e.g., [16, 20, 6]), but due to the limited expressive power of trees they are often applicable only in restrictive or over-simplified settings. There are numerous applications for LCA queries in dags, e.g., object inheritance in programming languages, lattice operations for complex systems, lowest common ancestor queries in phylogenetic networks, or queries concerning customer-provider relationships in the *Internet*. For a more detailed description of possible applications, we refer to [6, 5]. The algorithmic variants studied in this paper are:
All-Pairs Representative LCA : Compute one (representative) LCA for each pair of

vertices, and
All-Pairs All LCA : Compute the set of all LCAs for each pair of vertices.

1.1 Related Work

LCA algorithms have been extensively studied in the context of trees with most of the research rooted in [2, 22]. The first optimal algorithm for the all-pairs LCA problem in trees, with linear preprocessing time and constant query time, was given in [16]. The same asymptotics was reached using a simpler and parallelizable algorithm in [20]. Recently, a reduction to range minimum queries has been used to obtain a further simplification with optimal bounds on running time [6]. More algorithmic variants can be found in, e.g., [8, 25, 24, 9].

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 [18], 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 [3], where the representative problem in the context of object inheritance lattices was studied. The approach in [3], which is based on poset embeddings into boolean lattices yielded $O(n^3)$ preprocessing and $\log n$ query time on lower semi-lattices.

The representative LCA problem on general dags has been recently studied in [7, 6, 17]. Both works rely on fast matrix multiplications (currently the fastest known algorithm needs $\tilde{O}(n^\omega)$, with $\omega < 2.376$ [11]) to achieve $\tilde{O}(n^{\frac{\omega+3}{2}})$ [6] and $\tilde{O}(n^{2+\frac{1}{4-\omega}})$ [17] preprocessing time on dags with n nodes and m edges. For sparse dags, in [17], an $O(nm)$ algorithm has been presented as well.

The authors of [5] study variants of the representative LCA problem, namely (L)CA computations in weighted dags and the All-Pairs All LCA problem, i.e. finding all LCAs for each vertex pair. For the latter problem, an upper bound of $O(n^{3+\mu})$ [1] is shown, as well as algorithms with scaling properties. In [6], a short experimental study of the performance of All-Pairs Representative LCA algorithms is presented. It is demonstrated, that a suboptimal but simple $O(n^3)$ algorithm is the best practical choice.

1.2 Results

We summarize the technical contributions of this paper.

1. We present fast algorithms for solving the All-Pairs Representative LCA and All-Pairs All LCA problems with expected running time of $O(n^2 \log n)$ and $O(n^3 \log\log n)$ respectively. The speed-ups over recently developed methods is achieved by applying transitive reduction on the input dags.

[1]Throughout this work, $\omega(x, y, z)$ is the exponent of the algebraic matrix multiplication of a $n^x \times n^y$ with a $n^y \times n^z$ matrix. Let μ be such that $\omega(1, \mu, 1) = 1 + 2\mu$ is satisfied. The fastest known algorithms for rectangular matrix multiplication imply $\mu < 0.575$ and $\omega(2, 1, 1) < 3.3399$

2. For All-Pairs Representative LCA , we improve previous approaches in [6, 17] to $O(n^{2+\mu})$ by straightforward application of fast rectangular matrix multiplication.

3. We improve the recently shown upper bound for All-Pairs All LCA from $O(n^{3+\mu})$ to $O(n^{\omega(2,1,1)})$. This result is achieved by applying matrix multiplication to computing witness matrices $W^{(v)}$ for each vertex $v \in V$, where a $W^{(v)}[x,y]$ indicates that v is a CA, but not a LCA of (x,y).

4. We give first fully dynamical algorithms for both All-Pairs Representative LCA and All-Pairs All LCA . Here, the update complexities are $O(n^{2.5})$ and $O(n^3)$ respectively, with constant query times. The update complexities improve over recomputing the solutions from scratch. The dynamic algorithms are based on a technique which reduces the matrix multiplications in the static solutions to transitive closure computations in corresponding dags.

5. We study LCA computations experimentally. We demonstrate that simplicity and the transitive reduction technique are fundamental to practically efficient algorithms. Furthermore, we evaluate the running time of several All-Pairs Representative LCA and All-Pairs All LCA algorithms in different settings.

2 Preliminaries

Let $G = (V, E)$ be a directed graph. 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. Let $G_{\text{clo}} = (V, E_{\text{clo}})$ 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. Similarly, let $G_{\text{red}} = (V, E_{\text{red}}$ be the transitive reduction of G, i.e., the smallest graph G' such that $G_{\text{clo}} = G'_{\text{clo}}$. Throughout this work, let $| E_{\text{red}} | = m_{\text{red}}$. Observe that the transitive reduction of a dag G is a subgraph of G [1].

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. In many contexts a consistent ordering is called a topological ordering of the vertex set. Observe that a graph G is a dag if and only if it allows some topological ordering (folklore). Moreover, a topological ordering can be found in time $O(n+m)$ [12]: 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 $N(\text{NIL}) = 0$ throughout this work. We usually consider dags equipped with some topological ordering. In such cases we often omit the ordering. We refer to a vertex z which has the maximal topological number $N(z)$ among all vertices in a set as the *rightmost* vertex.

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 $\text{CA}(x, y)$, 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 $z \in \text{CA}(x, y)$ and for each $z' \in V$ with $(z, z') \in E_{\text{clo}}$ we have $z' \notin \text{CA}(x, y)$. $\text{LCA}(x, y)$ denotes the set of all LCAs of x and y.

3 Fast LCA Algorithms

The algorithms we present in this section are based on the algorithms described in [5]. The following lemma is fundamental to the speed-ups that lead to practically fast algorithms.

Lemma 1. *For all $x, y, z \in V$, it is true that $z = \mathrm{LCA}_G(x, y)$ if and only if $z = \mathrm{LCA}_{G_{\mathrm{red}}}(x, y)$.*

Proof. Suppose, for the sake of contradiction, that z is a LCA of (x, y) in G, but not in G_{red}. Then, by definition, either z is not a common ancestor of x and y in G_{red} or there exists a common ancestor z' of x and y such that z reaches z' in G_{red}. Since the transitive relationships between each pair of nodes is maintained between G and G_{red}, both implies that z cannot be a LCA of x and y, contradiction the assumption. The opposite direction is analogous. $\square$

The above lemma implies that LCA computations in a dag G can be restricted to the graph G_{red}. Observe, however, that this does not directly apply to LCA problems involving distances [5]. The restriction to the transitive reduction turns out to be of critical importance in designing practically fast algorithms for All-Pairs Representative LCA and All-Pairs All LCA . Moreover, it is possible to explain the fast running times by analyzing the algorithms on random dags.

3.1 All-Pairs Representative LCA

In [5], a dynamic programming algorithm which solves All-Pairs Representative LCA in worst case time $O(nm)$ is given. Applying Lemma 1 yields Algorithm 1

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] ← NIL
3      Compute the transitive reduction G_red and the transitive closure G_clo of G
4      Compute a topological ordering N
5      foreach v ∈ V in ascending order of N(v) do
6          foreach (v, x) ∈ E_red do
7              foreach y ∈ V with N(y) ≥ N(v) do
8                  if (x, y) ∈ E_clo then R[x, y] ← x
9                  else if N(R[v, y]) > N(R[x, y]) then R[x, y] ← R[v, y]
10             end
11         end
12     end
13 end
```

Theorem 2. *Algorithm 1 solves* All-Pairs Representative LCA *in $O(n\, m_{\mathrm{red}})$.*

Proof. The algorithms presented [15] and [21] can be used to compute the transitive reduction and the transitive closure of G in time $O(n\, m_{\mathrm{red}})$. The improvement in [21] is even faster. The running time of the loop in lines 5-9 can clearly be bounded by $O(n\, m_{\mathrm{red}})$ by construction. $\square$

Although the modified algorithm offers no theoretical advantage in the worst case, its superior practical performance can be explained by considering the expected value of the parameter m_{red} on random dags. Undirected graphs are studied under two popular random graph models, $G_{n,p}$ and $G_{n,M}$. Throughout this work, we restrict ourselves to random dags in the $G_{n,p}$ model. In this model, each possible edge in the graph is chosen independently with equal probability p, where $0 < p < 1$. A generalization of the random graph models for undirected graphs to directed acyclic graphs was offered by Barak and Erdős[4]. Consider an arbitrary (but fixed) ordering of the vertices of an undirected random graph and direct each edge from the lower to the higher indexed vertex.

The following lemma is due to Simon [21].

Lemma 3. *Let $G = (V, E)$ be a random dag in the $G_{n,p}$ model. Let m_{red} be a random variable denoting the number of edges in the transitive reduction of G. Then*

$$\mathbb{E}[m_{\text{red}}] = O(n \log n)$$

Corollary 4. *Let $G = (V, E)$ be a random dag in the $G_{n,p}$ model. Then, the expected running time of Algorithm 1 on G is $O(n^2 \log n)$.*

3.2 All-Pairs All LCA

We improve the algorithms for All-Pairs All LCA presented in [5] by applying Lemma 1.

Algorithm 2 is quite natural. First, it computes the transitive closure of G in $O(nm)$. Then, for every vertex z and every pair (x, y) it determines in time $O(\text{out-deg}(z))$ if z is an LCA of (x, y). If z is a CA of (x, y), but none of its children, then z is LCA of (x, y). Checking whether a given vertex is a CA of a vertex pair can be done by simple transitive closure look-up. The total running time of the original algorithm is $O(n^2 m)$. For this approach, Lemma 1 yields directly an improvement by considering G_{red} instead of G. The bound on the expected running time follows from Lemma 3

Theorem 5. *The time needed by Algorithm 2 to solve* All-Pairs All LCA *on $G = (V, E)$ is bounded by $O(n^2\, m_{\text{red}})$. Let $G = (V, E)$ be a random graph in the $G_{n,p}$ model. Then, Algorithm 2 solves* All-Pairs All LCA *on $G = (V, E)$ in expected time $O(n^3 \log n)$.*

Algorithm 3 is based on dynamic programming and 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 $(z, z') \in E_{\text{clo}}$.

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. In [5], it is shown that the merging operation can be performed in time $O(\min\{n, k^2\})$, where k is the maximum cardinality of any LCA set. This finding allows for scaling with the maximal LCA sets. Observe, however, that even for sparse graphs with $m = O(n)$, the total size of the LCA sets can be $\Omega(n^3)$ [5].

Note that if we do not know k in advance, we can decide on-line which merging strategy to use, without changing the asymptotic 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.

Algorithm 2: All-Pairs All LCA 1

```
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
begin
    Compute the transitive closure G_clo and the transitive reduction G_red of G
    Compute a topological ordering N
    foreach v ∈ V do
        foreach x, y ∈ V with N(v) ≤ N(x), N(y) do
            LOWEST = FALSE
            if (v, x) ∈ E_clo AND (v, y) ∈ E_clo then LOWEST = TRUE
            foreach z ∈ V such that (v, z) ∈ E_red do
                if (z, x) ∈ E_clo AND (z, y) ∈ E_clo then LOWEST = FALSE
            end
            if (LOWEST) then A[x, y] ← {v}
        end
    end
end
```

Theorem 6. *The time needed by Algorithm 3 to solve* All-Pairs All LCA *on* $G = (V, E)$ *is bounded by* $O(n\, m_{\text{red}} \min\{n, k^2\})$*, where* k *is the maximum LCA set. Let* $G = (V, E)$ *be a random dag in the* $G_{n,p}$ *model. Then, Algorithm 3 solves the* All-Pairs All LCA *problem in expected time* $O(n^2 \log n \min\{n, k^2\})$*.*

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

Corollary 7. *Let* $G = (V; E)$ *be a random dag in the* $G_{n,p}$ *model. Testing whether a given dag is a lower semi-lattice, an upper semi-lattice, or a lattice can be done in expected time* $O(n^2 \log n)$*.*

Algorithm 4 is based on the following idea. 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 using a topological ordering that maximizes $N(z)$. Since All-Pairs Representative LCA algorithms return rightmost LCAs, this approach guarantees that z is returned for the appropriate pairs. This can even be improved by maximizing the topological numberings of vertices an a path simultaneously. For more details, we refer to [5]. In Section 5, we empirically demonstrate the benefits of using path covers.

Theorem 8. *The time needed by Algorithm 4 to solve* All-Pairs All LCA *on* $G = (V, E)$ *is bounded by* $O(n\, m_{\text{red}}\, w)$*, where* w *is the size of path cover. Let* $G = (V, E)$ *be a random dag in the* $G_{n,p}$ *model. Then, Algorithm 4 solves* All-Pairs All LCA *in expected time* $O(n^3)$ *for* $\log^2 n/n \le p < 1$ *and expected time* $O(n^3 \log\log n)$ *for* $0 < p < \log^2 n/n$*.*

Proof. The running time of algorithm 4 is given by $O(n\, m_{\text{red}}\, w)$ where w is the width of the path cover of G. To analyze the expected running on a random dag, we use the following lemma [21]:

Algorithm 3: All-Pairs All LCA 2

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 the transitive closure G_{clo} and the transitive reduction G_{red} of G
- Compute a topological ordering N
- **foreach** $v \in V$ *in ascending order of* $N(v)$ **do**
  - **foreach** $y \in V$ *with* $N(v) < N(y)$ **do**
    - **if** $(v, y) \in E_{\text{clo}}$ **then** $A[v, y] \leftarrow \{v\}$
  - **end**
  - **foreach** $(v, x) \in E_{\text{red}}$ **do**
    - **foreach** $y \in V$ *with* $N(x) < N(y)$ **do**
      - **if** $(x, y) \in E_{\text{clo}}$ **then** $A[x, y] \leftarrow \{x\}$
      - **else** $A[x, y] \leftarrow \text{Merge}(A[v, y], A[x, y])$
    - **end**
  - **end**
- **end**

end

Lemma 9. *Let $G = (V, E)$ be a random dag in the $G_{n,p}$ model. Then, there exists an algorithm that computes a path cover of G of width w and the transitive reduction of G in time $O(w\, m_{\text{red}})$. Moreover*

$$\mathbb{E}[w\, m_{\text{red}}] = \begin{cases} O(n^2) & \textit{for } \log^2 n/n \leq p < 1 \\ O(n^2 \log\log n) & \textit{otherwise} \end{cases}$$

Algorithm B in [21] with running time $O(n+m)$ yields a path cover of size w satisfying the conditions of the above lemma. Hence, the theorem follows. □

4 Theoretical Improvements

4.1 All-Pairs Representative LCA

Kowaluk et al. [17] observed that rightmost LCAs are found by computing maximum witnesses for boolean matrix multiplication. Let A, B, and C be boolean $n \times n$ matrices such that $C = A \cdot B$. We have $C[i, j] = 1$ if and only if there exists $1 \leq k \leq n$ such that $A[i, k] = 1 = B[k, j]$; in this case, we call k a witness for $C[i, j]$. We call k a maximum witness for $C[i, j]$ if k is the maximum one among all $C[i, j]$-witnesses.

Proposition 10. [17] *Let $G = (V, E)$ be a dag and let N be a topological ordering. Let A be the adjacency matrix of the transitive closure $({}_{\text{clo}}G)$ such that the vertices are ordered corresponding to N. Then z is a rightmost ancestor of (x, y) if and only if z is a maximum witness for $C[x, y]$, where $C = A^T \cdot A$.*

We briefly describe how to improve the approach taken in [17] by using fast rectangular matrix multiplication [10] for computing the maximum witnesses. In the following, we assume that we have already computed the adjacency matrix A of the transitive closure $({}_{\text{clo}}G)$ of G in

Algorithm 4: ALL-PAIRS ALL LCA 3

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 the transitive closure G_{clo} and the transitive reduction G_{red} of G
  Compute a path cover $\mathcal{P}$ of 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 on G_{red} 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 multi-set-union)
    end
  end
  Remove elements of multiplicity greater than one from $A[x, y]$ for all $x, y \in V$
end

Algorithm 5: 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

begin
  Initialize $R[x, y] \leftarrow$ NIL
  Compute the transitive closure TC(G) and a topological ordering N of G
  foreach $v \in V$ *in ascending order of* $N(v)$ **do**
    foreach $(v, x) \in E$ **do**
      foreach $y \in V$ *with* $N(y) \geq N(v)$ **do**
        if $(x, y) \in (_{\text{clo}}G)$ **then** $R[x, y] \leftarrow x$
        else if $N(R[v, y]) > N(R[x, y])$ **then** $R[x, y] \leftarrow R[v, y]$
      end
    end
  end
end

time $O(\min\{nm, n^\omega\})$. Also, we assume implicitly that the rows and columns in A are ordered according to a topological order of G's vertex set.

Let $\mu \in [0; 1]$ be a parameter. We divide V into equal-sized sets $V_1, \ldots, V_r$ of consecutive vertices (with respect to the topological ordering), where $r = \lceil n^{1-\mu} \rceil$. Thus, the size of the sets is $O(n^\mu)$. Maximum witnesses are found in two steps:

1. For each pair (x, y), determine l such that the maximum witness of (x, y) is in V_l.

2. For each (x, y) and l, search V_l to find the maximum witness.

The implementation of these two steps is straightforward: for a vertex set V_l, let $A^T_{*V_l}$ denote the matrix A^T restricted to the columns corresponding to vertices in V_l. Let $M^{(l)} = A^T_{*V_l} \cdot (A^T_{*V_l})^T$.

Observation 11. *A pair (x, y) of vertices has a common ancestor $z \in V_l$ if and only if $M^{(l)}[x, y] = 1$.*

Hence for $l \in \{1, \ldots r\}$, we compute the $O(n^{1-\mu})$ (rectangular) matrix products $M^{(l)}$ and choose for each pair the maximum index l such that $M^{(l)}[x, y] = 1$. This takes time $O(n^{1-\mu+\omega(1,\mu,1)})$. Recall that $\omega(1, \mu, 1)$ is the exponent of the algebraic matrix multiplication of an $n \times n^\mu$ with an $n^\mu \times n$ matrix. For the second step, we simply search for each pair (x, y) and the corresponding index l of the set V_l manually, that is, for each $z \in V_l$ in descending order until we find z with both (z, x) and (z, y) are in $({}_{\mathrm{clo}}G)$. This takes time $O(n^2 \cdot |V_l|) = O(n^{2+\mu})$. For μ satisfying $1 - \mu + \omega(1, \mu, 1) = 2 + \mu$ we get the optimal complexity. Currently, the best known upper bounds for rectangular matrix multiplication [10] imply $\mu < 0.575$.

Theorem 12. All-Pairs Representative LCA *can be solved in time $O(n^{2+\mu})$, where μ satisfies $1 + 2\mu = \omega(1, \mu, 1)$.*

4.2 All-Pairs All LCA

We start by considering the following problem which we call One-Vertex All-Pairs LCA . Given a dag G and a vertex v, find all pairs (x, y) such that v is an LCA of (x, y). Clearly, from a One-Vertex All-Pairs LCA solution for each $v \in V$, a solution to All-Pairs All LCA can be derived in $O(n^3)$.

One-Vertex All-Pairs LCA reduces to the following. Find all pairs (x', y') such that v is a CA of (x', y'). Then, test for each such pair, if there exists a witness, i.e. a successor of v that is also a CA of this pair, or not. To this end, it is enough to consider the children of v in G.

A solution to this problem works as follows. We initialize an $n\times$ matrix $C^{(v)}$, such that $C^{(v)}[x, y] = 1$ if and only if v is a CA of (x, y). Obviously, this can be done in time $O(n^2)$ with knowledge of the transitive closure. Let $A^{(v)}$ be a $n \times n$ matrix such that $A^{(v)}[x, z] = 1$ if and only if x is reachable from z and z is a direct child of v. Let $A^{(v)}[x, z] = 0$ otherwise. This can be thought of as the transpose of the transitive closure restricted to the rows indexed with children of v, all other entries set to zero. Let A be the adjacency matrix corresponding to G_{clo}. Let $W^{(v)} = A^{(v)} \cdot A$ be the *witness matrix* of v.

Lemma 13. *Let $W^{(v)}$ be the witness matrix of vertex v. Then, v is a LCA of (x, y) if and only if $W^{(v)}[x, y] = 0$ and $C^{(v)}[x, y] = 1$.*

Proof. Suppose v is a LCA. Then clearly, $C^{(v)}[x, y] = 1$ by definition. On the other hand there exists no witness. Suppose that $A^{(v)}[x, z] \cdot A[z, y] = 1$ for some z. Then obviously by construction, z is a child of v and z reaches both x and y contradicting our assumption. That is, $A^{(v)}[x, z] \cdot A[z, y] = 0$ for all z and hence $W^{(v)}[x, y] = 0$. The other direction is analogous. ☐

Corollary 14. One-Vertex All-Pairs LCA *can be solved in time $O(n^\omega)$.*

The approach leads immediately to an $O(n^{1+\omega})$ algorithm for All-Pairs All LCA . Fast rectangular matrix multiplication yields even a stronger upper bound. We can compute the witness matrices $W^{(v)}$ for vertices $v \in V$ in one step by multiplying a $n^2 \times n$ and a $n \times n$ matrix in time $O(n^{\omega(2,1,1)})$.

$$\begin{pmatrix} A^{(v_1)} \\ A^{(v_2)} \\ \vdots \\ A^{(v_n)} \end{pmatrix} \cdot A \quad = \quad \begin{pmatrix} W^{(v_1)} \\ W^{(v_2)} \\ \vdots \\ W^{(v_n)} \end{pmatrix} \tag{1}$$

Theorem 15. All-Pairs All LCA *can be solved in time* $O(n^{\omega(2,1,1)})$.

Proof. The algorithm works as follows:

1. Compute the transitive closure of G
2. Compute the common ancestor matrices $C^{(v)}$ for all $v \in V$. Since G_{clo} is known, this can be achieved in time $O(n^3)$.
3. Compute the witness matrices $W^{(v)}$ for all $v \in V$ using Equation (1). This takes time $O(n^{\omega(2,1,1)})$. Currently, upper bounds for rectangular matrix multiplication imply $\omega(2,1,1) < 3.3399$.
4. Let $L^{(v)} = C^{(v)} - W^{(v)}$ for each $v \in V$. Then, for each pair (x, y), all LCAs can be read from the entries $L^{(v)}[x, y]$ for each v. This step takes a total of $O(n^3)$ time.

□

4.3 Dynamic Algorithms

All-Pairs Representative LCA . We show how to solve the fully dynamic All-Pairs Representative LCA problem with update time $O(n^{2.5})$ and query time $O(1)$. We consider *vertex-centered* updates, that is, given a vertex v, edges incident to v can be arbitrarily added or deleted in one update step. The approach is based on the matrix multiplication-based static All-Pairs Representative LCA solution described above. The static algorithm is based on rectangular matrix multiplications which serve as *CA existence* computations. To this end, the vertices are divided into n^{1-r} sets of size n^r for some $r \in [0, 1]$. To ease exposition, we assume in the sequel that n^r and n^{1-r} are integers. For each vertex set V^i one rectangular matrix product of an $n \times n^r$ and an $n^r \times n$ matrix is computed in order to identify all pairs for which a CA in the set V^i exists. The rightmost LCAs are then searched for in the vertex set with the largest index i. In order to improve the upper bound for vertex-centered updates, we reduce the rectangular matrix multiplications to transitive closure computations in dags $G^1, \ldots, G^l$, where $l = n^{1-r}$. The reduction has the following properties:

1. The adjacency matrices of the dags G^i correspond to the results of the rectangular matrix products in the static solution.
2. Vertex-centered updates incur at most a constant time of vertex-centered updates in each of the dags G^i.

Each of the transitive closures of the dags G^i can be updated in time $O(n^2)$ using recent results [19].

Theorem 16. *There exists an algorithm for dynamic* All-Pairs Representative LCA *with* $O(n^{1-r+\omega} + n^{2+r})$ *initialization,* $O(n^{1-r+2} + n^{2+r})$ *update, and* $O(1)$ *query time.*

Proof. We start by describing the reduction of the rectangular matrix multiplications to transitive closure computations in corresponding dags G^i. Let V^i be a subset of vertices of size n^r. Recall that the rectangular matrix multiplication is used to indentify all pairs (x, y) for which there exists a CA $z \in V^i$. Let $G^i = (V', E')$ be the following dag: let $V' = V_1 \cup V^i \cup V_2$, where V_1 and V_2 are copies of V. Observe that there exists a natural mapping $m : V' \to V$. Further, $(x, y) \in E'$ if and only if one of the following is true:

1. $x \in V_1, y \in V_1, (m(y), m(x)) \in E$, i.e. (x, y) is a *reverse* edge of an edge in G
2. $x \in V_2, y \in V_2, (m(x), m(y)) \in E$ or $x \in V_1, y \in V^i, m(x) = m(y)$, i.e. copies of vertices in V^i are connected
3. $x \in V^i, y \in V_2, m(x) = m(y)$, i.e. (x, y) is an edge in G.

This construction is similar to constructions used in [6] and [5]. Obviously, we have

Observation 17. *Let $x, y \in V$ be two vertices. Then, there exists a common ancestor $z \in V^i$ of x and y if and only if y is reachable from x in G^i*

This establishes property 1) of the reduction. Property 2), i.e., a vertex-centered update in G incurs only $O(1)$ vertex centered updates in G^i, also follows from the definition of G^i. The edges in G^i are either edges in G, reverse edges in G, or connecting edges. Hence a vertex-centered update in G incurs two vertex-centered updates in G^i. Now, the computation of $O(n^{1-r})$ transitive closures $G^{(i)}_{\text{clo}}$ yields the same information as the rectangular matrix products in the original approach. The rightmost LCAs can again be found by searching naively the corresponding vertex sets provided $({}_{\text{clo}}G)$ is updated along with the G^i_{clo}. However, the upper bound of the static solution not invariant to this modification. The computation of All-Pairs Representative LCA with transitive closures instead of rectangular matrix products takes time $O(n^{1-r+\omega} + n^{2+r})$. The above bound follows from the fact that corresponding transitive closures cannot be guaranteed to be computed faster than $O(n^\omega)$, unlike the rectangular matrix products. The best choice for r in this case is $r = 0.688$. The usage of transitive closures causes a slight overhead here, but improves the bound on updates in a dynamic setting. It is well known that the transitive closures of directed graphs and dags in particular can be updated in time $O(n^2)$ under vertex-centered udpates [19]. Hence, we get a dynamic algorithm with $O(n^{1-r+\omega} + n^{2+r})$ initialization, $O(n^{1-r+2} + n^{2+r})$ update, and $O(1)$ query time. □

Optimizing r for updates, we get $O(n^{2.876})$ initialization, $O(n^{2.5})$ update, and $O(1)$ query time.

All-Pairs All LCA . For All-Pairs All LCA we achieve $O(n^{1+\omega})$ initialization, $O(n^3)$ update and $O(1)$ query time for fully dynamic vertex-centered updates. Here, the key is to speed up witness computations in the dynamic setting. Again, the dynamic speed-up is achieved by using transitive closure computations instead of matrix products for the witness matrix computations. The application of the reduction technique described above to All-Pairs All LCA is straightforward.

Theorem 18. *There exists an algorithm for dynamic* All-Pairs Representative LCA *with $O(n^{3.376})$ initialization, $O(n^3)$ update, and $O(1)$ query time.*

Observe that $O(n^3)$ update time is optimal in the worst case, since a single update can change up to $O(n^3)$ entries. Combining the above two theorems with results on dynamic all-pairs shortest distance computations [23], we get the following two corollaries concerning upper bounds for dynamic LCA computations in weighted dags, see [5].

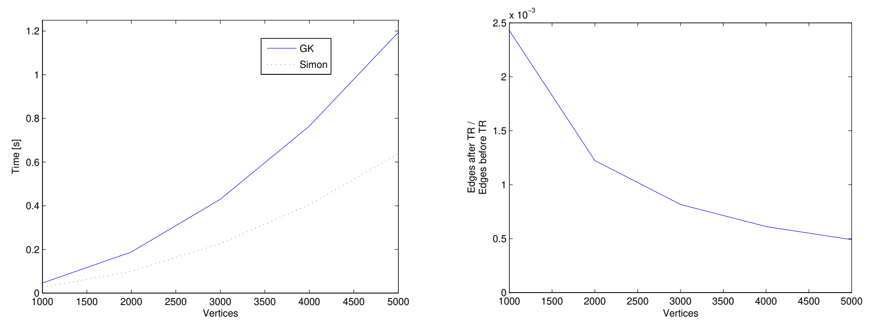
Corollary 19. *There exists an algorithm for dynamic* All-Pairs Shortest Distance LCA *with $O(n^{3.376})$ initialization, $O(n^3)$ update, and $O(1)$ query time.*

Corollary 20. *There exists an algorithm for dynamic* All-Pairs Shortest Distance CA *with APSD initialization, $\tilde{O}(n^2)$ update, and $O(1)$ query time.*

5 Experiments

Experimental Setup. We have implemented the All-Pairs Representative LCA (APRLCA) and All-Pairs All LCA algorithms (APA1, APA2, APA3) described in Section 3 in C++. For comparison with algoritms that were subject to the experimental study in [6], we adapted the code provided by the authors to fit in our C++ framework. Additionally, we implemented the two transitive closure algorithms described in [21]: the algorithm of Goralčíková (GK) and Koubek[15] with expected running time of $O(n^2 \log n)$ and the algorithm of Simon [21] (Simon) with expected running time of $O(n^2 \log \log n)$. Our preprocessing routines are complemented by the greedy algorithm for computing a path cover of G with running time $O(n+m)$[21]. Both of the transitive closure algorithms naturally compute the transitive reduction and are used to accomplish both tasks. Approaches based on fast algebraic matrix multiplication were excluded from this study. These methods are practically not efficient. The tests were done on a system with an Opteron CPU clocked at 1.8 GHz with 1 GB RAM and running on Linux . The test data is based on random dags. For each randomly created dag, a super-source, adjacent to all other vertices was included, all other edges were added with probability p. In order to mirror random dags of varying density we use the parameter p to control the expected number of edges in the dag. In all of our experiments we considered sparse ($\mathbb{E}[m] = \Theta(n)$), medium ($\mathbb{E}[m] = \Theta(n \log n)$), and dense ($\mathbb{E}[m] = \Theta(n^2)$) graphs. As a consequence of our experimental results, we restrict ourselves to sparse and dense dags in this presentation.

Transitive Reduction. We first compared the two methods for creating the transitive reduction of a dag, see Figure 1(a). Simon clearly outperforms GK, even on moderately sized graphs. As a result of the study in all further experiments, this method was used to create the transitive reductions. Additionally, we evaluated the effects of the transitive reduction on the number of edges, see Figure1(b).



(a) Comparison of the transitive reduction algorithms on dense dags

(b) Effects of transitive reduction on dense dags

Figure 1: Evaluation of transitive reduction

All-Pairs Representative LCA . We compared the performance of Algorithm 1 with two of the methods presented in [6]. The third algorithm tested in [6] was ruled due to its inferior performance. This finding is in accordance with the results of the study presented in [6]. *RMQuery* is based on combining ancestor lists for each of the vertices with LCA queries on a spanning tree of G. The running time of the algorithm is $O(n^3)$, but as a result of the experimental study in [6], it is efficient in practice. *TCQuery* is based on transitive closure queries. It compute G_{clo} first and then chooses the rightmost CA of a pair (x, y) by comparing the corresponding rows of the adjacency matrix of G_{clo}. The running time of the algorithm is $\Theta(n^3)$.

Furthermore, we examined the effects of transitive redcution on the running time of the algorithms, i.e., on each density level, we tested the algorithms with and without transitive reduction. The results can seen in Figure 2. The performance TCQuery was not considered with transitive reduction.

The dynamic programming algorithm with transitive reduction proofs to be the algorithm of choice. The benefits of using the transitive reduction significantly incraese at higher densities. Observe that in Figure 2(f), the performance line corresponding to APRLCA is almost identical to the x-axis.

All-Pairs All LCA . The All-Pairs All LCA algorithms were tested similarily. However, the large output complexity of the problem limits the tests to considerably smaller graphs, i.e. $n \leq 800$. The results of the eperiments are quite interesting. Obviously, APA2 is the best choice. Although APA3 has the best expected running time, the overhead caused by the more complicated data structures does not pay off for small graphs. The superior performance of APA2 is not surprising considering the fact that the LCA sets are usually small. The average number of LCAs for a pair is less then 2 in our experiments, even on large, dense dags.

We seperately tested the effects of using a path cover to maximize the topological number of several vertices in one step over maximizing only a single vertex. The results can be seen in Figures 3(e) and 3(f). While on sparse graphs, the additional cost of computing and maintaining the cover does not pay off, the effects on medium and dense dags are considerable.

6 Conclusion

We have presented and experimentally evaluated fast algorithms for LCA problems in dags. As the main finding of our experimental study, we have identified two critical factors for fast LCA algorithms: simplicity of algorithms and usage of transitive reduction. More work remains to be done, e.g., designing All-Pairs All LCA algorithms that can handle larger input dags. Another line of future research is concerned with efficient implementations of algorithms for shortest distance LCA problems.

On the theoretical side, we have improved the upper bound for All-Pairs All LCA and described dynamical algorithms. Observe also that the results in this paper translate to improved upper bounds for some of LCA variants studied in [5], e.g., All-Pairs Shortest Distance LCAor testing lattice-theoretic properties of dags.

References

[1] Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

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

[3] Hassan Aït-Kaci, Robert S. Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, 1989.

[4] A.B. Barak and P. Erdős. On the maximal number of strongly independent vertices in a random directed acyclic graph. *SIAM Journal on Algebraic and Discrete Methods*, 5(4):508–514, 1984.

[5] Matthias Baumgart, Stefan Eckhardt, Jan Griebsch, Sven Kosub, and Johannes Nowak. All-pairs common-ancestor problems in weighted dags. Technical Report TUM-I0606, Institut für Informatik, Technische Universität München, 2006.

[6] Michael A. Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.

[7] Michael A. Bender, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Finding least common ancestors in directed acyclic graphs. In *SODA*, pages 845–854, 2001.

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

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

[10] Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complexity*, 13(1):42–49, 1997.

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

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[13] Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, (111), 2006.

[14] Stefan Eckhardt, Andreas M. Mühling, and Johannes Nowak. Fast lca computations in directed acyclic graphs. Technical Report TUM-I0707, Institut für Informatik, Technische Universität München, 2006.

[15] A. Goralčíková and Václav Koubek. A reduct-and-closure algorithm for graphs. In *MFCS*, pages 301–307, 1979.

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

[17] Miroslaw Kowaluk and Andrzej 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.

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

[19] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *FOCS*, pages 509–517, 2004.

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

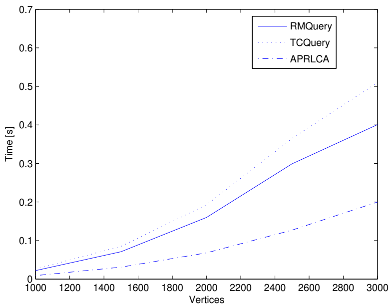
[21] Klaus Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theor. Comput. Sci.*, 58:325–346, 1988.

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

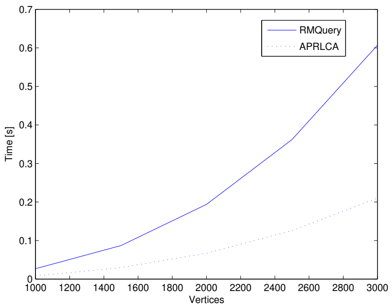
[23] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *SWAT*, pages 384–396, 2004.

[24] Biing-Feng Wang, Jiunn-Nan Tsai, and Yuan-Cheng Chuang. The lowest common ancestor problem on a tree with an unfixed root. *Information Sciences*, 119(1–2):125–130, 1999.

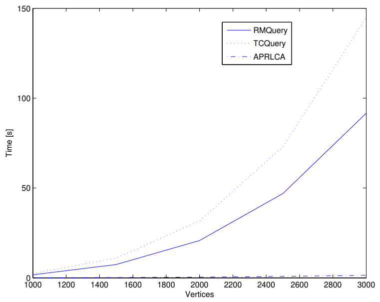
[25] Zhaofang Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Information Processing Letters*, 51(1):11–16, 1994.



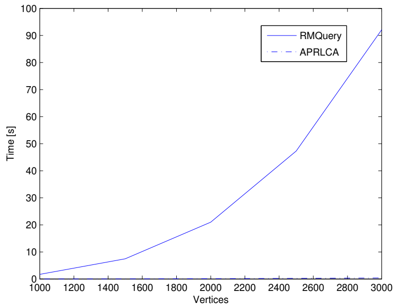
(a) sparse dags, no transitive reduction



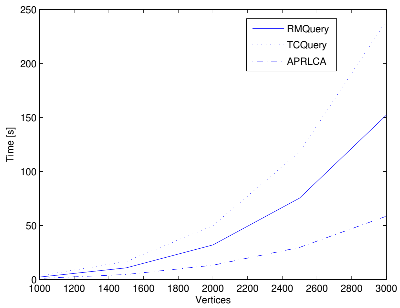
(b) sparse dags, transitive reduction



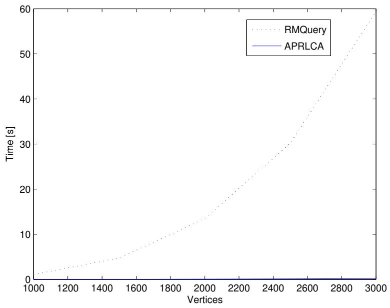
(c) medium dags, no transitive reduction



(d) medium dags, transitive reduction

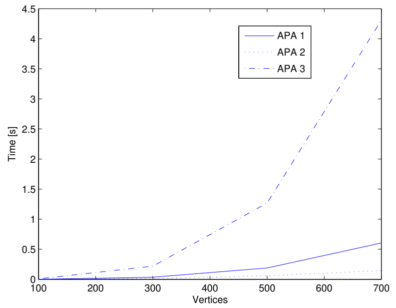


(e) dense dags, no transitive reduction

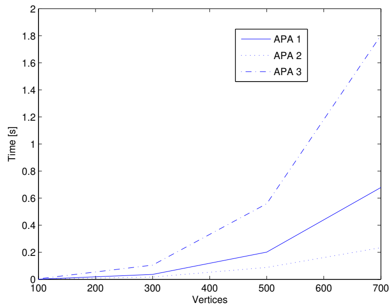


(f) dense dags, transitive reduction

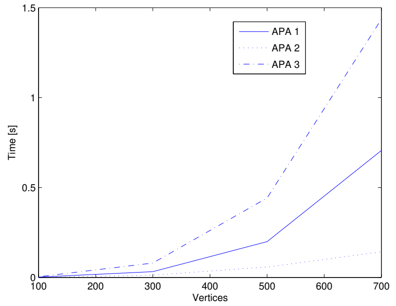
Figure 2: Evaluation of All-Pairs Representative LCA algorithms



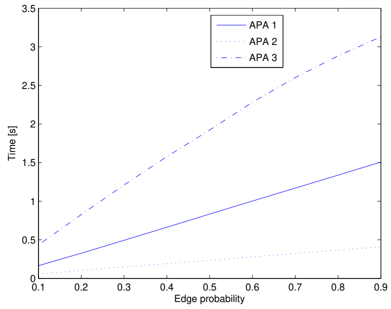
(a) All-Pairs All LCA algorithms on sparse dags



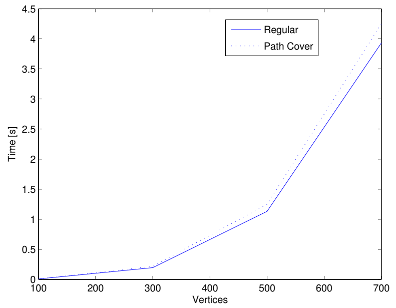
(b) All-Pairs All LCA algorithms on medium dags



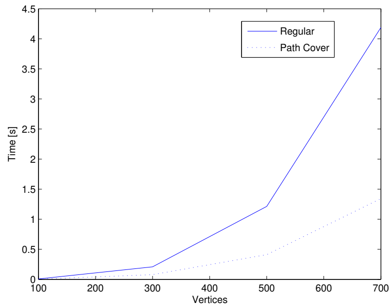
(c) All-Pairs All LCA algorithms on dense dags



(d) All-Pairs All LCA algorithms with varying p



(e) Algorithm 4 with and without the Path Cover Speed-Up on sparse dags



(f) Algorithm 4 with and without the Path Cover Speed-Up on dense dags

Figure 3: Evaluation of All-Pairs All LCA algorithms