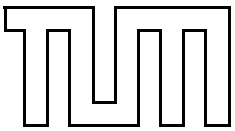
Technische Universität München
Fakultät für Informatik



Lehrstuhl für Effiziente Algorithmen

Algorithmic Methods for Lowest Common Ancestor Problems in Directed Acyclic Graphs

Johannes Nowak

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. A. Kemper, Ph.D.
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. E. W. Mayr
2.	Univ.-Prof. Dr. F. J. Esparza Estaun

Die Dissertation wurde am 04.02.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 15.09.2009 angenommen.

Abstract

Lowest common ancestor (LCA) problems in directed acyclic graphs (dags) have attracted scientific interest in the recent years. Directed acyclic graphs are powerful tools for modeling causality systems or other kind of entity dependencies, and efficient solutions to the respective lowest common ancestor problems are indispensable computational tools with regard to proper analysis of these systems. Similar problems in trees are widely understood, however, the generalizations to dags fall short of achieving comparable efficiencies.

In this work, we develop and analyze algorithmic techniques for solving LCA problems in dags. The main focus is on *all-pairs* LCA problems, i.e., problems that require the answers to the respective LCA queries for all vertex pairs in the dag. In particular, the basic problems studied are finding one arbitrary (*representative*) LCA for all vertex pairs and listing *all* LCAs for all vertex pairs. We identify and describe in-depth three basic algorithmic approaches that lead to efficient solutions to these problems, namely dynamic programming, matrix multiplication, and a path cover approach.

The dynamic programming method in combination with using transitive reduction yields algorithms that are efficient in the average case and – as a result of an experimental study also described in this thesis – in practice. However, the running times depend on the number of edges in the transitive reduction of the input dags and are hence vulnerable to special constructs such as dense bipartite graphs.

Matrix multiplication approaches lead to general upper time bounds for the two problems that improve upon hitherto best solutions. More specifically, we prove that representative LCAs for all vertex pairs can be computed in time $O(n^{2.575})$, and all LCAs for all vertex pairs can be computed in time $O(n^{3.334})$ for a dag with n vertices. We note in this place that any improvement of the matrix multiplication exponent automatically improves these bounds for the LCA problems.

The third algorithmic approach, a path cover technique, yields efficient solutions for the two problems in dags G with small width $\mathrm{w}(G)$, namely $O(n^2\mathrm{w}(G)\log n)$ for computing representative and $O(n^2\mathrm{w}(G)^2\log^2 n)$ for computing all LCAs. However, perhaps the most important result connected with the path cover technique is an improved algorithm for finding representative LCAs in general that restricts the class of dags for which the upper bound $O(n^{2.575})$ is actually attained significantly. Further research in this direction might ultimately improve this bound.

Additionally, we review algorithmic applications of the presented techniques, namely problem variants in dynamic settings, in weighted dags, and under space-efficiency considerations. Although some of the upper time bounds that we present in this work might turn out to be tight, further study of these and alike problems seems to be a promising direction for future research.

Finally, we present the results of an experimental study of some of the algorithms described in this work, in particular, the algorithms that are based on dynamic programming. As a consequence, we conclude that both representative and all LCAs for all vertex pairs can be computed reasonably fast in practice, i.e., with runtime close to $O(n^2)$.

Acknowledgments

In the first place, I thank my advisor Ernst W. Mayr for his helpful, encouraging guidance and generous support throughout my time of doctoral studies. Furthermore, I am indebted to all the current and former members of the Efficient Algorithms Group for providing a stimulating and motivating working environment. In particular, I am thankful to Stefan Eckhardt, Moritz Maaß, and Hanjo Täubig who have contributed significantly to my academic progress. Special thanks goes to Arno Buchner (and his wife) for providing food on the week-ends. Finally, I thank my parents exceptionally, not only for their contribution to this thesis but for my education in general.

CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER

1

INTRODUCTION

1.1 Motivation

Lowest common ancestor problems in directed acyclic graphs have attracted scientific interest in the recent years. Directed acyclic graphs are powerful tools for modeling causality systems or other kind of entity dependencies, and efficient solutions to the respective lowest common ancestor problems are indispensable computational tools with regard to proper analysis of these systems. Similar problems in trees are widely understood, however, the generalizations to dags fall short of achieving comparable efficiencies.

If we think of causal relations among a set of events, natural questions come up, such as: Which events entail two given events? What is the first event that entails two given events? In dags, these questions can be answered by computing common ancestors (CAs), i.e., vertices that reach each of the given vertices via a path, 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 essential computational primitives, they have become an independent subject of studies only recently, initiated by the seminal paper of Bender et al. [BFCP$^+$05]. This work has led to an increasing interest for this topic in the community, and a number of follow-up papers have been published since [KL05, BEG$^+$07, CKL07, KL07, EMN07].

There is a lot of sophisticated work devoted to LCA computations for the special case of trees, see, e.g., [HT84, SV88, BFCP$^+$05], but due to the limited expressive power of trees they are often applicable only in restrictive or over-simplified settings.

A number of natural applications of (lowest) common ancestor computations in dags have been described in [BFCP$^+$05, BEG$^+$07, EMN07]. We review these applications below.

- *Object inheritance in programming languages.* In *object-oriented programming languages* like Java and C++, objects are instances of classes. By *object-inheritance* one denotes the fact that objects inherit properties of super-classes. Moreover, the object inheritance relation of a

given set of objects depends on the temporal order in which the classes are defined and appear in a certain expression. The structure of object-inheritance is commonly modeled by partial orders. Modern object-oriented programming languages support multiple inheritance; hence the resulting partial order can not expected to be tree-like. The topology of the inheritance order has to be evaluated at several stages of the compilation and execution of a program. Lowest common ancestor computations are a natural tool for resolving and analyzing such orders. Early research [AKBLN89] in this context is restricted to posets with lattice properties, i.e., upper semi-lattices. However, in general lowest common ancestors of pairs of objects are not unique.

So far this problem has been only considered in a static context, i.e., static compilation, and hence static lowest common ancestor solutions can be used. However, there are systems that use *incremental* compilation where compilation is done during the execution of a program. For such cases we extend the study of the respective lowest common ancestor computations to dynamic cases, see Chapter 6.

- *Genealogical linkage analysis. Genetic linkage* refers to the fact that sets of genetic loci, e.g., genes, are often inherited jointly. The linkage of these loci has physical reasons, i.e., physical connection on the same chromosome. The probability that closely situated loci are separated during the crossing over of the DNA is lower than for loci that are far apart.

  Pedigree graphs are used to represent family or hereditary relationships. Such graphs are used for *linkage analysis*, a technique to identify genes associated we certain features, e.g., a genetic disease. The basic idea of linkage analysis is to identify genetic markers that are linked to a gene of interest.

  In the presence of *inbreeding*, that is, spouses might be genetically correlated, linkage analysis techniques rely on a method for quantifying pairwise degrees of inbreeding in such graphs. This is readily done by identifying so-called inbreeding loops. Such loops are formed by ancestral and descendant paths of vertex pairs. The degree of inbreeding in turn can be derived from shortest inbreeding loops. Such loops can be found by using (lowest) common ancestor algorithms on weighted dags (Chap. 6).

- *Analysis of the Internet graph.* 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-configured, these local policies have a critical influence on routing stability and quality [GW99, GR01]. 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., [Gao01]). Computing such top providers, which are just CAs and LCAs, is needed for reliability or efficiency analyses.

  The snapshot of the Internet dag that we use in our experimental setting (Chap. 7) has 22,218 vertices and 57,413 edges. Finding top providers for each of the 493,617,306 pairs makes fast CA and LCA algorithms an issue.

- *Analysis of phylogenetic networks.* Lowest common ancestor 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 [NSW+03, MNW+04, NW05]. 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.

- *Analysis of citation data.* Citation graphs are obtained from scientific literature and have been studied and analyzed extensively, see e.g., [AJM04]. Basically, the vertex set of such a graph is given by a set of publications and an edge represents a citation from one publication to another. Computing LCAs in such dags may provide valuable information, e.g., which papers are original to two or more papers on a particular topic.

1.2 Results

The main focus in this thesis is on algorithmic solutions for *all-pairs* LCA problems in directed acyclic graphs. All-pairs in this context means computing a solution matrix that contains the answers for the LCA problem in question for all vertex pairs. We distinguish at first between two problem variants, namely ALL-PAIRS REPRESENTATIVE LCA, computing one (representative) LCA, and ALL-PAIRS ALL LCA, listing all LCAs. We outline the contents and main results described in this work below.

Notational concepts and concise problem statements are introduced in Chapter 2. Chapters 3 through 5 are dedicated to the description of three basic algorithmic techniques that lead to efficient solutions to the considered problems.

Chapter 3 describes dynamic programming algorithms for LCA problems. It is shown that this approach combined with using a transitive reduction of the input dag yields algorithms whose running times are (a) efficient if the number of edges in the transitive reduction is reasonably small and (b) close to trivial lower bounds in the average case. For ALL-PAIRS REPRESENTATIVE LCA we prove the following.

Theorem 3.7. *In a dag G with n vertices, the time needed by the dynamic programming algorithm (Algorithm 2) to solve* ALL-PAIRS REPRESENTATIVE LCA *can be bounded by $O(n m_{\text{red}})$, where m_{red} is the number of edges in the transitive reduction of G.*

The average case time complexity of this approach can be bounded by $O(n^2 \log n)$[1] (Cor. 3.8) and is hence optimal up to a logarithmic factor. The main result in this chapter for ALL-PAIRS ALL LCA is stated in Theorem 3.15.

Theorem 3.15. *In a dag G with n vertices, the time needed by the dynamic programming algorithm (Algorithm 4) to solve* ALL-PAIRS ALL LCA *can be bounded by $O(n m_{\text{red}} \cdot \min\{(\kappa^2 + \kappa \log n), n\})$, where m_{red} is the number of edges in the transitive reduction of G and κ is the maximum LCA set cardinality in G.*

[1] Throughout this work, we denote by $\log n$ the logarithm with base 2 of n.

Average case analysis yields upper time bounds of $O(n^3 \log n)$ in the general case and, if κ is assumed to be bounded by a constant, even $O(n^2 \log n)$ (Cor. 3.16), which is an assumption supported by experimental evidence in many cases.

In Chapter 4 algorithms that use (rectangular) matrix multiplication are studied in detail. The two major results presented in this chapter are the currently tightest upper time bounds for ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA.

Theorem 4.8. ALL-PAIRS REPRESENTATIVE LCA *can be solved in time* $O\left(n^{2+\mu}\right)$ *on a dag G with n vertices, where* μ *satisfies* $1+2\mu = \omega(1,\mu,1)$.

Here, $\omega(a,b,c)$ denotes the exponent of the multiplication of an $n^a \times n^b$ by an $n^b \times n^c$ matrix. The numerical value of μ can currently be bounded by $\mu < 0.575$. For more details on the derivation and implications of parameters related to matrix multiplication we refer to Chapter 4.

Theorem 4.19. ALL-PAIRS ALL LCA *can be solved in time* $O\left(n^{\omega(2,1,1)}\right)$ *on a dag G with n vertices.*

We note that this theorem implies an upper time bound of $O(n^{3.334})$ for ALL-PAIRS ALL LCA. Further results described in this chapter involve computational equivalence proofs of so-called *fixed-vertex all LCA variants* and Boolean matrix multiplication (Thm. 4.9). These considerations lead to algorithms that improve the general upper time bound for ALL-PAIRS ALL LCA on dags that are moderately sparse (Thm. 4.21, Thm. 4.24). Note, however, that this improvement is stronger than the results in Chapter 3. For example, while the running time of the dynamic programming algorithm can be bounded by $O(n^3)$ in the worst case only if $m_{\text{red}} = O(n)$, the results of Chapter 4 imply that this bound can already be achieved if $m_{\text{red}} = O(n^{1.294})$.

Chapter 5 deals with algorithmic approaches that use derivatives of a path cover approach, i.e., a set of directed paths in a dag G that cover the whole vertex set. Applying a fairly general path cover technique that essentially decomposes the LCA problem into a number of LCA problems in trees yields the following theorem.

Theorem 5.16. ALL-PAIRS REPRESENTATIVE LCA *can be solved in time* $O(n^2 \mathrm{w}(G) \log n)$, *and* ALL-PAIRS ALL LCA *can be solved in time* $O(n^2 \mathrm{w}(G)^2 \log^2 n)$ *on a dag G with n vertices and width* $\mathrm{w}(G)$.

It is also shown that the path cover technique can be combined with an earlier algorithm [CKL07] for ALL-PAIRS REPRESENTATIVE LCA that exploits low depth in the input dag. As a consequence we present an algorithm that has worst case time complexity $\widetilde{O}(n^{2+\mu})$ (Thm. 5.20), which corresponds to the current best upper time bound, up to polylogarithmic factors. Most importantly, however, we show that the class of dags in which this bound is actually attained is significantly restricted.

Theorem 5.22. *For a dag G with n vertices and an arbitrary constant* $\mu \geq \delta > 0$ ALL-PAIRS REPRESENTATIVE LCA *can be solved in time* $\widetilde{O}(n^{2+\mu-\delta})$ *if G does not contain a subgraph H that contains at least* $n^{\mu-\delta}$ *chains of length at least* $n^{1-\mu-\delta}$.

As an additional application, the combined algorithm can be used to improve a recently established upper time bound for the ALL-K-SUBSETS REPRESENTATIVE LCA problem for $k = 3$.

Theorem 5.23. *The* ALL-K-SUBSETS REPRESENTATIVE LCA *problem on a dag G with n vertices can be solved in time $O(n^{3.5214})$ for $k = 3$ and $\widetilde{O}(n^{k+\frac{1}{2}})$ for $k \geq 4$.*

In Chapter 6 algorithmic applications of the techniques developed are described. The considered problems extend the study of the two problem primitives, ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA. The first extension concerns edge- and vertex-weighted dags. Instead of computing arbitrary LCAs one is interested in (L)CAs that minimize (or maximize) ancestral distances or weights. Non-trivial solutions are given for each of the problem variants. Furthermore, reductions to better understood problems are given, namely:

Theorem 6.2. ALL-PAIRS SHORTEST DISTANCE CA *and can be reduced to* ALL-PAIRS SHORTEST DISTANCES*, up to a polylogarithmic factor.*

Theorem 6.6. ALL-PAIRS MINIMUM WEIGHT CA *and* ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION *are computationally equivalent.*

However, there is a complexity gap between the CA and the LCA versions of the problems. It is not clear whether this gap can be closed.

For ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA in a fully-dynamic environment, an algorithmic solution is described that decreases the update time under *vertex-centered* updates (Thm. 6.17) compared to recomputing a solution from scratch. This can be further improved for an incremental model on dense dags (Thm. 6.24).

Finally, it is shown that using the path cover technique developed in Chapter 5, ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA can be solved in a space-efficient way for dags with small width.

Theorem 6.25. *The representative LCA problem on a dag G with n vertices and width $\mathrm{w}(G)$ can be solved with $O(\mathrm{w}(G)n^2 \log n)$ preprocessing time, $O(\mathrm{w}(G)n \log n)$ space, and $O(\mathrm{w}(G) \log n)$ query time.*

The approach can also be extended to the all LCA problem and k-subset LCA problem variants.

An experimental study of some of the algorithms described in this thesis is presented in Chapter 7. In particular, the dynamic programming approaches are evaluated and tested against results of an earlier experimental analysis. As the first main result, we conclude that ALL-PAIRS REPRESENTATIVE LCA can be solved in time close to $O(n^2)$ for a wide spectrum of random and real-world dags by applying the dynamic programming approach. In all cases its performance is superior to the algorithms tested in the previous experimental study. The second result concerns ALL-PAIRS ALL LCA algorithms. Again, results imply that ALL-PAIRS ALL LCA can be computed in time close to $O(n^2)$ in most cases. However, medium-sized input dags, i.e., dags where the number of edges is close to $n \log n$ turn out to be more difficult. For such dags, none of our algorithms are significantly subcubic.

1.3 Publications

Parts of the results in this thesis have been presented previously at the 1^{st} International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE'07) in

Hangzhou, China [BEG+07] and at the 15^{th} Annual European Symposium on Algorithms (ESA'07) in Eilat, Israel [EMN07]. In addition to that two technical reports [BEG+06, EMN06] contain material described in this thesis. Finally, large parts of the results described in Chapter 5 and partly Chapter 6 can be found in [KLN08], which is submitted for publication.

As a citation policy, we have decided to prioritize journal versions over conference works throughout this thesis. Note that this might lead to chronological inconsistencies since not all of the relevant works have appeared in journals. That is, a result from 2007 (journal version) might be improved in 2006 (conference version). Furthermore, the respective research history cannot directly be derived from citations prioritized in this way. However, journal versions can in general be expected to have a higher quality, both with respect to the correctness and completeness of the presented results.

Results by other authors that are used in this thesis are generally labeled "Proposition", regardless of the significance of the results or the type of usage (theorem, lemma) in the respective work. Moreover, results that belong to the standard repertoire in computer science such as *depth first traversal* or standard asymptotic notation are not referenced explicitly, but by references to standard textbooks like [CLRS01].

CHAPTER

2

PRELIMINARIES

2.1 Elementary Concepts

Basic Graph Theory

We assume basic knowledge of graph theory throughout this work. Nevertheless, we review some basic facts for the sake of introducing our notational concepts.

An (undirected) graph is a pair $G = (V, E)$ of sets such that the elements of E are unordered pairs $\{u, v\}$ of elements of V. The elements of V are called the *vertices* or *nodes* of the graph G, the elements of E are called the *edges* of G. In a *directed* graph or *digraph*, the edges have a direction as an additional attribute, i.e., the elements of E are *ordered* pairs (u, v) of vertices. The edge (u, v) is said to be *directed* from u to v. In what follows we restrict our attention to directed graphs without explicitly stating this fact.

The cardinality of the vertex set V is denoted by n, and the cardinality of the edge set E is denoted by m unless stated otherwise. Let $e = (u, v) \in E$ be an edge in G. We say the two vertices u and v are *adjacent* to each other. On the other hand, e is said to be *incident* to both u and v. We restrict ourselves to the consideration of *simple* graphs, i.e., there is at most one edge between each pair of vertices in G. A *loop* is an edge (v, v) for some $v \in V$. A graph without loops is called *loopless*.

The following notational concepts are equivalent for a directed graph $G = (V, E)$:

1. $(u, v) \in E$.
2. u is a *parent* of v and v is *child* of u respectively.
3. u is *direct predecessor* of v and v is a *direct successor* of u in G respectively.

A *path* of length $l - 1$ for $l \geq 1$ in $G = (V, E)$ is a sequence $(v_1, \ldots, v_l)$ of vertices such that $(v_i, v_{i+1}) \in E$ for all $(1 \leq i \leq l - 1)$, i.e., v_l can be reached from v_1 by a sequence of $l - 1$ directed edges. In this work, we consider only *simple* paths, i.e., paths such that all vertices on the path are

pairwise disjoint.

Again, the following are equivalent:

1. There exists a path $(v_1,\ldots,v_l)$ such that $v_1 = u$ and $v_l = v$.
2. u reaches v and v is *reachable* from u.
3. u is a *predecessor* or *ancestor* of v and v is a *successor* or *descendant* of u.

Observe that in our terminology a vertex v reaches itself by a path of length 0. By $u \rightsquigarrow v$ we denote u reaches v.

Let $v \in V$ be a vertex. By $N(v)$ we denote the set of vertices u that are adjacent to v in G. We say that $N(v)$ is the *neighborhood* of v. $N(v)$ can be further subdivided into the *in-neighborhood* $N^{\text{in}}(v)$ and the *out-neighborhood* $N^{\text{out}}(v)$. For a vertex $u \in V$, $u \in N^{\text{in}}(v)$ if $(u,v) \in E$, whereas $u \in N^{\text{out}}(v)$ if $(v,u) \in E$. Observe, however, that $N^{\text{out}}(v)$ and $N^{\text{in}}(v)$ are not necessarily disjoint in general digraphs. For a vertex v, the *in-degree* of v, denoted by $\text{indeg}(v)$, is equal to the cardinality of $N^{\text{in}}(v)$. Conversely, the *out-degree* of v, denoted by $\text{outdeg}(v)$, is equal to the cardinality of $N^{\text{out}}(v)$.

Directed Acyclic Graphs

Directed acyclic graphs are used in numerous applications. This includes, for example, Bayesian networks, directed acyclic word graphs, dependency graphs of classes in object-oriented programming languages, or representing spacetime as a causal set in theoretical physics. In general, dags are used to model systems in which entities are related or depend on each other in a non-cyclic way, e.g., imposed by causal or temporal dependencies.

A *cycle* is a path $(v_1,\ldots,v_l)$ such that $l \geq 2$ and $v_1 = v_l$. A *simple cycle* is a cycle $(v_1,\ldots,v_l)$ such that $(v_2,\ldots,v_l)$ is a simple path. A directed graph $G = (V,E)$ that does not contain a cycle as a subgraph is called a *directed acyclic graph* (dag). A *topological ordering* of the vertex set V is an injective mapping $top : V \to \{1,\ldots,n\}$ such that for all vertices $u,v \in V$, v is reachable from u implies $top(u) < top(v)$. A topological ordering can be found in time $O(n+m)$ through a *depth first traversal* (DFS) of G. It is well known that a directed graph $G = (V,E)$ is a dag if and only if there is a topological ordering of V.

A *source* is a vertex with no incoming edges, and a *sink* is a vertex with no outgoing edges. The *depth* of a vertex $v \in V$, denoted by dp(v), in a dag is defined as the length of a longest path $p = (s,\ldots,v)$ from a source s to v in G. Analogously, the height of a vertex v, denoted by $\text{h}(v)$, is the length of a longest path $p = (v,\ldots,s)$ from v to a sink s in G. The *depth* of G is given by

$$\text{dp(G)} = \max_{\text{v}\in\text{V}} \text{dp(v)}. \tag{2.1}$$

Observe that $\max_{v\in V} \text{dp(v)} = \max_{\text{v}\in\text{V}} \text{h(v)}$, and therefore the terms "depth" and "height" of a dag are commonly intermixed in the literature. Let $G = (V,E)$ be a dag with depth dp(G). The vertex set V of G can be divided into $\text{dp(G)}+1$ *levels* $L_0,\ldots,L_{\text{dp(G)}}$, where $L_i = \{v \in V | \text{dp(V)} = \text{i}\}$ for all $0 \leq i \leq \text{dp(G)}$.

The notions of *transitive closure* and *transitive reduction* of a directed acyclic graph $G = (V,E)$ play a prominent role in this work.

Definition 2.1 (Transitive Closure)**.** *The reflexive, transitive closure of a dag $G = (V,E)$, denoted by $G_{\text{clo}} = (V,E_{\text{clo}})$, is a dag with vertex set V and edge set E_{clo} such that for all $u,v \in V$, $(u,v) \in E_{\text{clo}}$ if and only if u is a predecessor of v in G.*

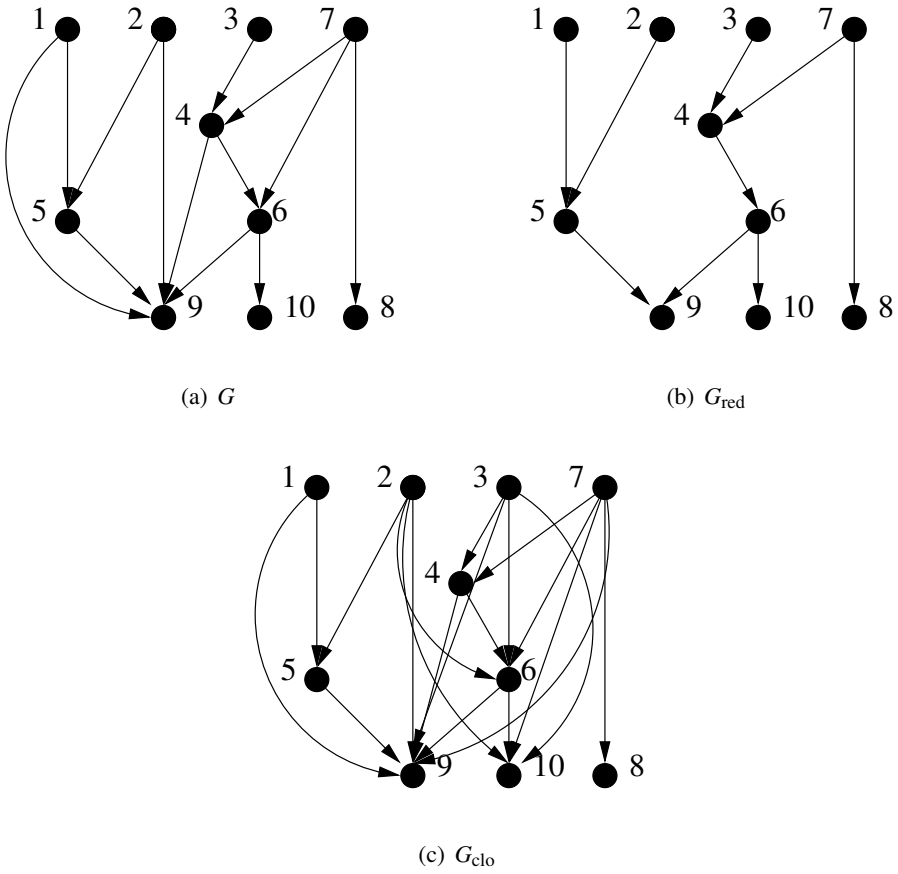
Again, we note that we consider a vertex v to be a predecessor of itself. Throughout this work, we denote the number of edges in G_{clo} by m_{clo}.

Definition 2.2 (Transitive Reduction)**.** *The transitive reduction $G_{\text{red}} = (V,E_{\text{red}})$ of a dag $G = (V,E)$ is a dag with vertex set V and edge set E_{red} such that*

(i) G_{red} and G have the same transitive closure and

(ii) G_{red} is the smallest (with respect to the number of edges) dag G' on the vertex set V such that $G_{\text{clo}} = G'_{\text{clo}}$.

Throughout this work, let $|E_{\text{red}}| = m_{\text{red}}$ be the number of edges in the transitive reduction G_{red} of G.



(a) G (b) G_{red}

(c) G_{clo}

Figure 2.1: Transitive closure and transitive reduction of a dag $G = (V,E)$.

The following well-known proposition is due to Aho, Garey, and Ullman [AGU72].

Proposition 2.3. *Let $G = (V,E)$ be a directed acyclic graph. Then*

(i) G_{clo} and G_{red} are unique.

(ii) G_{red} is a subgraph of G.

Observe that property (ii) of the above proposition is not true for general digraphs, which makes it considerably harder to compute transitive reductions. In the case of dags, however, computing the transitive closure and the transitive reduction are computationally equivalent [AGU72], see also Proposition 4.1.

The transitive closure G_{clo} of a dag $G = (V,E)$ is in one-to-one correspondence with a partially ordered set (poset) $P = (V, \leq)$. Here, V is the ground set and the relation $\leq$ corresponds to the edges in G_{clo}. In this sense, by slight abuse of notation, we refer to G_{clo} also as a poset. Observe that by definition G_{red} and G correspond to the same poset G_{clo}. We say that two vertices $u, v \in V$ are *comparable* if either $(u,v) \in E_{\text{clo}}$ or $(v,u) \in E_{\text{clo}}$. Otherwise, they are *incomparable*. An *antichain* in a dag G is a vertex subset $V' \subseteq V$ such that the vertices in V' are pairwise incomparable. A *maximum antichain* is an antichain of maximum cardinality. The width of a dag G, denoted by $\mathrm{w}(G)$, has a prominent role in several places of this thesis, most notably in Chapter 5.

Definition 2.4 (Dag Width)**.** *For a dag $G = (V,E)$ the width $\mathrm{w}(G)$ of G is equal to the cardinality of a maximum antichain.*

For a dag $G = (V,E)$, a *path cover* $\mathscr{P}$ is a set of (directed) paths $\{p_1, \ldots, p_r\}$ in G such that for every $v \in V$ there exists at least one path $p \in \mathscr{P}$ such that v lies on p. A minimum path cover is a path cover $\mathscr{P}$ of minimum cardinality. The cardinalities of a maximum antichain and a minimum path cover are related by the famous Dilworth Theorem [Dil50]:

Proposition 2.5 (Dilworth '50)**.** *The width $\mathrm{w}(G)$ of a dag $G = (V,E)$ equals the size of a minimum path cover of G.*

In the context of posets one usually considers *chain covers* instead of path covers. In general, a chain cover of V in a dag $G = (V,E)$ is a path cover $\mathscr{P}$ of V such that the paths in $\mathscr{P}$ are vertex-disjoint, i.e., $p_i \cap p_j = \emptyset$ for all $p_i, p_j \in \mathscr{P}$ with $i \neq j$. The terms "path cover" and "chain cover" are sometimes intermixed due to the fact that there is a one-to-one correspondence between path covers in G and chain covers in G_{clo}. In particular, this means that each chain cover in G_{clo} corresponds to one or more path covers in G and *vice versa*. Furthermore, it holds that the cardinality of a minimum chain cover of G_{clo} is equal to the cardinality of a minimum path cover of G. Consider, e.g., the graph G_{red} depicted in Figure 2.1(b). A possible path cover of size 4 for G_{red} is given by $\mathscr{P} = \{(1,5,9),(2,4,6),(3,4,6,10),(7,8)\}$. The corresponding chain cover in G_{clo} (Fig. 2.1(c)) is $\mathscr{C} = \{(1,5,9),(2,4,6),(3,10),(7,8)\}$, i.e., the path $(3,4,6,10)$ is simply replaced by the chain $(3,10)$.

2.2 Analysis of Algorithms

While it is difficult to give a formal definition of an *algorithm*, one can easily agree that an algorithm can be informally described as a precisely defined sequence of instructions to solve a certain task. The

analysis of algorithms is a fundamental subject in computer science. Discovering the characteristics of specific algorithms enables us to compare algorithms or evaluate the usability of algorithms in certain applications. Before we describe how we approach the task of analyzing algorithms in this thesis, we briefly clarify our view on algorithms and their underlying *computational problems*.

We usually consider algorithms for computational problems. A computational problem is an (somehow) abstract formalization of concrete problems sharing particular characteristics. For example, computing shortest paths in graphs is an abstraction of finding the fastest route from A to B in a given road network. In algorithmics one usually takes this abstract point of view. That is, one concentrates on computational problems and general solutions for them instead of tailoring a solution for a specific underlying application. However, in many cases it is possible to improve general algorithms for computational problems by exploiting properties of specific concrete applications.

In order to assess the quality of an algorithm one most commonly analyzes the *cost* of the algorithm in terms of resources, e.g., time and space. Algorithms can be analyzed mathematically and experimentally. While experimental analysis mainly aims at predicting the behavior of algorithms in a realistic environment, i.e., with respect to concrete problems, as accurately as possible, mathematical analysis often restricts to asymptotic classification of algorithms and the underlying computational problem.

Computational Complexity

The *complexity* of an algorithm is a description of the cost of the algorithm in dependence of the *input size*. In this sense complexities are commonly expressed as functions of the input size. That is, for an algorithm $\mathscr{A}$ we denote by $\mathscr{T}_{\mathscr{A}}(n)$ the cost of $\mathscr{A}$ for an input of size n. The *worst case* complexity refers to the maximum cost of the algorithm on inputs of a given size, whereas *average case* complexity is with respect to the *expected* cost of the algorithm; for more details, see below. We usually consider worst case complexities in this work and omit explicitly stating this to ease exposition.

Only in rare cases it is possible to find functions that describe the complexity of a given algorithm exactly. Therefore, one usually neglects constant terms. This gives rise to the standard asymptotic notation or *Landau symbols*. For more details we refer to standard text books on algorithms, e.g., [CLRS01].

Since we consider algorithms for computational problems, we naturally extend the notion of computational complexity to problems. More specifically, the computational complexity of a problem is equal to the best possible complexity of a (possibly not yet found) respective algorithm. Observe that, in order to determine the complexity of a problem, we have to find a function that serves both as appropriate (asymptotic) upper and lower bound. Hence, we say a computational problem $\mathscr{B}$ has complexity $\mathscr{T}_{\mathscr{B}}(n)$ if the complexity of an optimal algorithm for the problem is $\Theta(\mathscr{T}_{\mathscr{B}}(n))$.

An upper bound for a computational problem is readily given by specifying an algorithm and proving that the cost of the algorithm is bounded accordingly. For direct lower bounds, one often needs more complicated mathematical arguments, and in many cases finding good lower bounds seems to be a demanding task. Therefore, one often characterizes the relationship of computational problems instead. This gives rise to the notion of *computational equivalence*. For example, the only known lower time bound for Boolean matrix multiplication is the trivial lower bound $\Omega(n^2)$. On the other hand, it is well-known that Boolean matrix multiplication is computationally equivalent to a variety

of other problems, e.g., finding the transitive closure of a digraph [AHU76]. Informally speaking, this means that these two problems have the same complexity, i.e., upper and lower bounds. Equivalence implies further that an upper (lower) bound for one of the problems can be directly transferred to the other problem. Using asymptotic notation one can define computational equivalence as follows.

Definition 2.6 (Computational Equivalence)**.** *Two computational problems $\mathscr{B}$ and $\mathscr{C}$ with computational complexities $\mathscr{T}_{\mathscr{B}}(n)$ and $\mathscr{T}_{\mathscr{C}}(n)$ respectively are computationally equivalent if and only if $\mathscr{T}_{\mathscr{B}}(n) = \Theta(\mathscr{T}_{\mathscr{C}}(n))$ for inputs of size n.*

Alternatively, $\mathscr{B}$ and $\mathscr{C}$ are computationally equivalent if and only if the two problems can be *reduced* to each other. A reduction from $\mathscr{B}$ to $\mathscr{C}$ can be viewed as an algorithm that (i) transforms problem $\mathscr{B}$ into problem $\mathscr{C}$ and (ii) specifies how a solution to $\mathscr{B}$ can be obtained from a solution to $\mathscr{C}$. In this work we consider reductions that allow upper bound transfers in order to establish relationships between computational problems. That is, a *valid* reduction from $\mathscr{B}$ to $\mathscr{C}$ implies $\mathscr{T}_{\mathscr{B}}(n) = O(\mathscr{T}_{\mathscr{C}}(n))$. To this end, the transformation algorithm is obviously required to be upper bounded by $O(\mathscr{T}_{\mathscr{C}}(n))$. Observe that this is in a certain sense the same as requiring a linear transformation. Just consider a virtual problem that incorporates the non-linear part of the transformation and the solution to $\mathscr{C}$. Then, $\mathscr{B}$ can be transformed in linear time into the virtual problem which is, by construction, computationally equivalent to $\mathscr{C}$. This in turn implies that $\mathscr{B}$ is equivalent to $\mathscr{C}$. In some cases we drop the requirement of a valid reduction as described above in the following way. We say that $\mathscr{B}$ can be reduced to $\mathscr{C}$, up to a polylogarithmic factor, if the transformation algorithm is upper bounded by $O(\mathscr{T}_{\mathscr{C}}(n) \cdot \log^c n)$ for a constant c. As a consequence of a transformation of this kind we obtain weaker complexity results, i.e., $\mathscr{T}_{\mathscr{B}}(n) = O(\mathscr{T}_{\mathscr{C}}(n) \cdot \log^c n)$. However, in some cases we are satisfied with rougher complexity classifications so that ignoring polylogarithmic factors is acceptable.

A Note on Polylogarithmic Factors

We extend the common asymptotic symbols slightly by using the so-called *soft-Oh notation*, which subsumes polylogarithmic factors. That is, $\widetilde{O}(f(n))$ is short for $O(f(n) \cdot \log^c n)$ for a constant c. The reason for neglecting polylogarithmic terms in the context of some considered problems and algorithms is that – as of now – one is primarily interested in finding the appropriate exponents with respect to the complexities. Polylogarithmic factors, on the other hand, are asymptotically negligible compared to constant improvements in the exponent. While we apply the soft-Oh notation mainly to ease exposition, the usage can cause confusion in some places, and the relevant literature is scattered with inaccuracies. This is particularly true when using exponents whose numerical value is not yet finally determined. For example, ω is commonly used to denote the exponent of square matrix multiplication, i.e., two $n \times n$ matrices can be multiplied in time $O(n^\omega)$. Currently, we have a bound of $\omega < 2.376$ for ω [CW90]. However, actually the assertion of this bound is slightly stronger, namely $\omega \leq 2.376 - \delta$ for a constant $\delta > 0$. Having this in mind, a complexity of $\widetilde{O}(n^\omega)$ implies $O(n^{2.376})$. Similarly, $\widetilde{O}(n^{2+\mu})$ implies $O(n^{2.575})$ for μ satisfying $1 + 2\mu = \omega(1, \mu, 1)$, which in turn implies $\mu < 0.575$, see Chapter 4.

Model of Computation

We adopt the *uniform cost model*, that is, with regard to time complexity, an algorithm consists of *primitive operations* and each operation is assigned a unit cost. In contrast, the *logarithmic cost model*

takes the number of bits on which the operation operates into account. The rational behind the uniform cost model is that primitive operations with moderately sized operands are usually implemented in the main processor to take only a constant number of clock ticks. Although certain operations like multiplication depend on the number of bits of the operands, assuming constant cost is a reasonable assumption on modern system architectures. With regard to memory requirements, the complexity is measured in the uniform cost model with respect to the number of words, i.e., chunks of bits of a certain size.

Average Case Analysis

Average case analysis of algorithms aims at answering the following question: What is the complexity of an algorithm for a *typical* input. In this context, a typical input of a given length n is an input drawn at random from an underlying input space I_n according to some distribution. Consider, for example, the problem of sorting arbitrary permutations of $\{1,\ldots,n\}$ by using the well known *Quicksort* algorithm with a deterministic pivot rule. The worst case running time of Quicksort is $\Theta(n^2)$, whereas the average case running time is only $\Theta(n\log n)$ under the assumption that all possible inputs, i.e., all permutations, are uniformly distributed. That is, the probability that a fixed permutation is chosen as input is given by $1/n!$. In the case that every input has the same probability, the average case complexity can be calculated as follows. Let $\mathscr{T}(i)$ be the running time of a given algorithm on input i.

$$\mathscr{T}_{\text{avg}}(n) = \frac{\sum_{i\in I_n}\mathscr{T}(i)}{|I_n|} \tag{2.2}$$

The accuracy of average case analysis with respect to predicting the behavior of an algorithm in practice depends heavily on the considered distribution of the input space. That is, the adopted probabilistic model has to be sufficiently realistic. Given an arbitrary distribution of the input space, the average case complexity of an algorithm corresponds to the expected value of the considered resource:

$$\mathscr{T}_{\text{avg}}(n) = \sum_{i\in I_n}\mathscr{T}(i)\Pr[i] \tag{2.3}$$

We note that although the value of the considered resource can be considered as a random variable in this context one usually avoids to use the term *expected complexity*. This is done to prevent confusion with complexity results for *randomized* algorithms, which are usually referred to as expected complexities.

2.3 Common Ancestor Problems in Directed Acyclic Graphs

Lowest Common Ancestors in Trees

The problem of finding lowest common ancestors in trees is defined as follows: Given a pair of vertices $\{x,y\}$ in a rooted tree T with n vertices, find the deepest vertex z that is an ancestor of both x and y. In the context of trees the depth of a vertex v is usually defined as its distance from the root vertex – as opposed to the maximum length of a path from a source vertex to v in a dag. However, the definitions are not contradicting since the path from the root to a vertex v is unique in a tree. In addition to its inherent algorithmic beauty, the problem of finding lowest common ancestors in trees

is an indispensable algorithmic tool in many areas of computer science and in particular in the context of string algorithms and computational biology. In the latter case LCA computations are frequently carried out on suffix trees, e.g., in order to find the longest common prefix of a given pair of substrings. For more applications see, e.g., [Gus97].

In a seminal paper Harel and Tarjan [HT84] showed that it is possible to preprocess a tree in linear time in order to enable constant time LCA queries. The algorithm given in the above paper was later simplified by Schieber and Vishkin [SV88] and again by Berkman and Vishkin [BV94]. Bender et al. [BFCP$^+$05] use a reduction to *range minimum queries* in an array to give a simple and practical LCA solution. The connection between LCA queries and range minimum queries in an array that is obtained from an *Euler tour* of the tree was first discovered by Gabow et al. [GBT84]. Cole and Hariharan [CH05] showed that a datastructure supporting constant time LCA queries can be maintained in a dynamic setting with constant update time.

In spite of the broad spectrum of research done in the context of LCAs in trees, generalizations to directed acyclic graphs have only been an independent subject of research in the last decade.

Basic Definitions

We formally define *common ancestors* (CAs) and *lowest common ancestors* (LCAs) of vertex pairs in directed acyclic graphs.

Definition 2.7 (Common Ancestor)**.** *Let $G = (V,E)$ be a dag and $x,y \in V$. A vertex $z \in V$ is a common ancestor (CA) of x and y if and only if both $z \rightsquigarrow x$ and $z \rightsquigarrow y$ in G.*

By $\mathrm{CA}_G\{x,y\}$, we denote the set of all CAs of x and y in G.

Definition 2.8 (Lowest Common Ancestor)**.** *Let $G = (V,E)$ be a dag and $x,y \in V$. A vertex $z \in V$ is a lowest common ancestor (LCA) of x and y if and only if $z \in \mathrm{CA}\{x,y\}$ and there is no vertex $z' \in \mathrm{CA}\{x,y\}$ distinct from z such that $z \rightsquigarrow z'$.*

$\mathrm{LCA}_G\{x,y\}$ denotes the set of all LCAs of x and y in G. We omit the subscript and use $\mathrm{CA}\{x,y\}$ and $\mathrm{LCA}\{x,y\}$ whenever it is clear from the context which graph G is referred to. Alternatively, one can define $\mathrm{LCA}\{x,y\}$ as follows. Consider the subgraph G^S that is induced by the vertices of $\mathrm{CA}\{x,y\}$. Then $\mathrm{LCA}\{x,y\}$ is equal to the set of vertices having an empty out-neighborhood in G^S.

Let $z \in \mathrm{CA}\{x,y\}$, but $z \notin \mathrm{LCA}\{x,y\}$ for some $x,y,z \in V$. From the above definitions, it follows that there exists a vertex $w \in V$, such that $w \in \mathrm{CA}\{x,y\}$ and $z \rightsquigarrow w$. We call such a vertex *w witness* to the fact that $z \notin \mathrm{LCA}\{x,y\}$, or simply witness for z and $\{x,y\}$. We capture the following easy observation.

Observation 2.9. *For a vertex pair $\{x,y\}$ and a vertex z such that $z \in \mathrm{CA}\{x,y\}$ and $z \notin \mathrm{LCA}\{x,y\}$ the following holds:*

(i) There exists a witness w for $\{x,y\}$ and z such that w is a child of z.

(ii) There exists a witness w for $\{x,y\}$ and z such that $w \in \mathrm{LCA}\{x,y\}$.

In general, LCAs of vertex pairs are not unique in dags. However, there are special classes of dags, most prominently trees and (upper) semi-lattices in which the LCAs are unique. Observe, however, that unless stated otherwise we consider the general case and treat $\mathrm{LCA}\{x,y\}$ as a set.

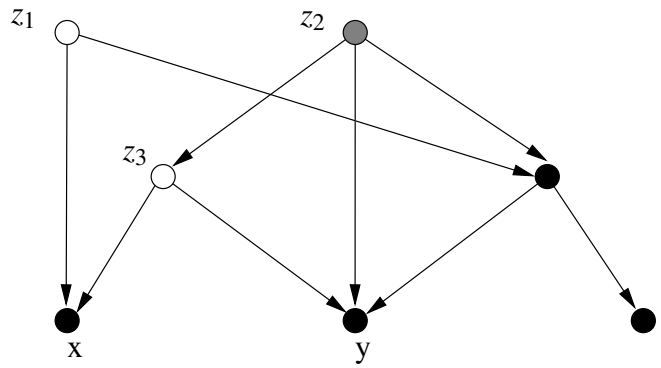


Figure 2.2: (Lowest) common ancestors of a vertex pair $\{x,y\}$. $\mathrm{CA}\{x,y\} = \{z_1,z_2,z_3\}$ and $\mathrm{LCA}\{x,y\} = \{z_1,z_3\}$. Observe that z_3 is a witness for z_2 and $\{x,y\}$.

Problem Definitions and Related Work

This thesis is mainly concerned with *all-pairs* problem variants. This means that we are interested in computing answers to certain (lowest) common ancestor query problems for each vertex pair $\{x,y\}$. Solutions to all-pairs problem variants are usually outputted by specifying a solution matrix L, i.e., $L[x,y]$ holds the solution for the given query with respect to $\{x,y\}$.

In the more general case of dags where a pair of nodes may have more than one LCA, one distinguishes – in contrast to trees – *representative* and *all* LCA problems. 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 [NU94], 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 [AKBLN89], where the representative LCA problem in the context of object inheritance lattices was studied. The approach in [AKBLN89], which is based on poset embeddings into Boolean lattices, yielded $O(n^3)$ preprocessing and $O(\log n)$ query time on lower semilattices.

Historically, the first considered all-pairs LCA problem on general dags was the ALL-PAIRS REPRESENTATIVE LCA problem [BFCP$^+$05]. In this problem, we are interested in computing an arbitrary (representative) LCA for each pair of vertices.

Problem:	ALL-PAIRS REPRESENTATIVE LCA
Input:	A dag $G = (V,E)$
Output:	A matrix R of size $n \times n$ such that for all vertices $x,y \in V$, $R[x,y] \in \mathrm{LCA}\{x,y\}$; if $\mathrm{LCA}\{x,y\} = \emptyset$ for some vertex pair $\{x,y\}$, then $R[x,y] = \mathrm{NIL}$.

In the context of representative LCA computations, the notion of *maximum* common ancestors (MCAs) is of particular importance. Let $G = (V,E)$ and let $top : V \to \{1,\ldots,n\}$ be a topological ordering of V. For an arbitrary vertex subset $V' \subseteq V$, let $u \in V'$ be a vertex such that $top(u)$ is maximal among all vertices in V', i.e., $u = \mathrm{argmax}_{v \in V'}\{top(v)\}$. Then u is said to be the maximum vertex in

V'. The following proposition is fundamental to many solutions to representative LCA problems.

Proposition 2.10 ([BFCP$^+$05, KL05])**.** *Let $G = (V,E)$ be a dag and let top 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 maximum vertex in $\mathrm{CA}\{x,y\}$ with respect to top, then z is an LCA of x and y.*

Proof. By definition, a common ancestor z is a lowest common ancestor if no other common ancestor is reachable from z. That is, if a common ancestor z is not a lowest common ancestor, there must be a witness z', i.e., $z' \in \mathrm{CA}\{x,y\}$ and $z \rightsquigarrow z'$. For every vertex z' that is reachable from z in G it is true that $top(z) < top(z')$ in any consistent ordering. Hence, if $top(z)$ is the maximum vertex in $\mathrm{CA}\{x,y\}$, there is no witness, and it follows that z is a lowest common ancestor of x and y. ■

Most algorithms for the ALL-PAIRS REPRESENTATIVE LCA problem presented in this work make use of the above proposition. That is, they compute maximum common ancestors (MCAs) with respect to some topological ordering. However, the problems do not seem to be equivalent. For example, the ALL-PAIRS REPRESENTATIVE LCA algorithm for dags of small depth given by Czumaj et al. [CKL07] does not necessarily compute maximum CAs but only CAs of maximum depth. A variation of the problem of computing maximum CAs with respect to a topological ordering is the problem of computing maximum CAs with respect to an arbitrary vertex weight function, which is considered in Chapter 6.

The first subcubic algorithm for ALL-PAIRS REPRESENTATIVE LCA was given by Bender et al. [BFCP$^+$05]. The authors essentially make use of Proposition 2.10 to transform the problem to an all-pairs shortest paths computation. Using the approximate all-pairs shortest distances algorithm by Zwick [Zwi98], the sets of candidate vertices are narrowed down to a sublinear fraction of the original vertex sets to achieve an upper bound of $\widetilde{O}(n^{\frac{\omega+3}{2}})$. This was improved by Kowaluk and Lingas [KL05] by reducing the problem to computing maximum witnesses for Boolean matrix multiplication; again by a straightforward application of Proposition 2.10. The resulting bound was $O(n^{2+\frac{1}{4-\omega}})$. In [KL05] and [BEG$^+$07], $O(nm)$ time algorithms, which are efficient for sparse dags, have been presented as well.

The following non-trivial lower bound for ALL-PAIRS REPRESENTATIVE LCA is also due to [BFCP$^+$05].

Proposition 2.11. *The problem of computing the transitive closure of an arbitrary digraph $G = (V,E)$ can be reduced to* ALL-PAIRS REPRESENTATIVE LCA *on G.*

The above proposition follows from the fact that $u \rightsquigarrow v$ if and only if $\mathrm{LCA}\{u,v\} = u$. Furthermore, the problem of computing the transitive closure of a general digraph is not harder than computing the transitive closure of a dag. Since computing the the transitive closure of a digraph is computationally equivalent to Boolean matrix multiplication, Proposition 2.11 essentially implies a lower bound of $\Omega(n^\omega)$, where ω is the exponent of square matrix multiplication, for ALL-PAIRS REPRESENTATIVE LCA.

A more general problem variant is the ALL-PAIRS ALL LCA problem where one asks for the sets $\mathrm{LCA}\{x,y\}$, i.e., listing all LCAs for all vertex pairs.

Problem:	ALL-PAIRS ALL LCA
Input:	A dag $G = (V, E)$
Output:	A matrix A of size $n \times n$ such that for all vertices $x, y \in V$, $R[x, y] = \text{LCA}\{x, y\}$.

Although this problem is of interest in its own right, algorithmic solutions for it have an interesting implication on LCA problems in edge- or vertex-weighted dags. By now, the asymptotically best algorithms for computing LCAs that minimize constraints related to the weight functions use an ALL-PAIRS ALL LCA solution as a subroutine. Again, for more details on problems related to weighted dags we refer to Chapter 6.

Theorem 2.12. ALL-PAIRS ALL LCA *in a dag G with n vertices has a lower time bound of $\Omega(n^3)$ in the worst case.*

Proof. It is easy to see that the output complexity of ALL-PAIRS ALL LCA is $\Omega(n^3)$ in the worst case. This is even true for sparse dags with $m = O(n)$ as the example in Figure 2.3 shows. ■

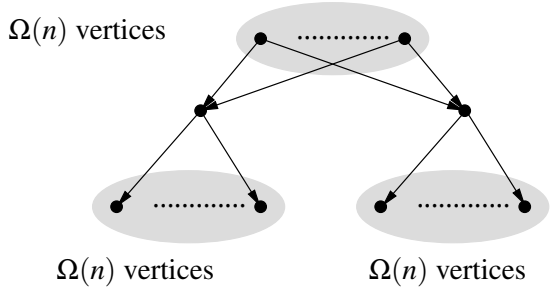


Figure 2.3: Sparse dags with an $\Omega(n^3)$ total number of LCAs.

CHAPTER

3

DYNAMIC PROGRAMMING ALGORITHMS

3.1 Introduction

Dynamic programming is a general algorithm design technique that solves a problem by combining solutions to subproblems. In this sense, dynamic programming is similar to the well-known *divide-and-conquer* paradigm. However, dynamic programming is preferred over divide-and-conquer when the considered subproblems are dependent, that is, they share subsubproblems. The advantage of dynamic programming in this scenario is that the repeated solution of the same subproblem is avoided.

Typically, the process of developing a dynamic programming algorithm includes the following two steps:

1. Decompose the problem into subproblems and find a recursive formulation for the problem solution.
2. Compute the optimal solution in a bottom-up fashion. To this end, it is important that there exists a natural order of the (not too many) subproblems from small to large. The solutions to the subproblems are typically stored in a table. Using a bottom-up approach enables dependent subproblems to share solutions to smaller subsubproblems.

An introductory example for dynamic programming is the iterative computation of Fibonacci numbers. Here, the dynamic programming approach yields a linear time solution whereas the pure divide-and-conquer approach results in an exponential running time. More examples for efficient dynamic programming algorithms are, e.g., the Floyd-Warshall algorithm [CLRS01] for computing all-pairs shortest paths, the Cocke-Younger-Kasami algorithm (CYK) [HMU01] for determining if and how a given string can be generated by a given context-free grammar, many string algorithms in general and sequence alignment in particular.

In this chapter we describe algorithmic solutions to LCA problems based on dynamic programming. More specifically, we obtain the following results for a dag G with n vertices whose transitive reduction has m_{red} edges.

- ALL-PAIRS REPRESENTATIVE LCA can be solved in $O(n\,m_{\text{red}})$ time in the worst case and $O(n^2 \log n)$ time in the average case.
- ALL-PAIRS ALL LCA can be solved in $O(n^2\,m_{\text{red}})$ time in the worst case and $O(n^3 \log n)$ time in the average case.
- ALL-PAIRS ALL LCA can be solved in $O(n\,m_{\text{red}} \min\{\left(\kappa^2 + \kappa \log n\right), n\})$ time in the worst case and $O(n^2 \log n \min\{(\kappa^2 + \kappa \log n), n\})$ time in the average case, where κ is the maximum number of LCAs of a vertex pair $\{x,y\}$.

As a result of an experimental study described in Chapter 7, the algorithms presented in this chapter turn out to be practical.

We give a brief chapter outline. In Section 3.2 we describe basic concepts and algorithms that are used by our dynamic programming approach. In particular, we describe how transitive reduction can be used to improve algorithms that depend on the number of edges, an algorithm by Simon [Sim88] to compute the transitive reduction and closure of a dag, and, finally, how we approach the matter of average case analysis with respect to graph algorithms throughout this work. We proceed by developing dynamic programming algorithms for ALL-PAIRS REPRESENTATIVE LCA in Section 3.3 and ALL-PAIRS ALL LCA in Section 3.4. For all of the described algorithms we provide appropriate worst case and average case upper time bounds.

3.2 Preliminaries

Transitive Reduction as a Speed-Up

The dynamic programming approaches presented in this chapter depend heavily on the number of edges m in the considered dag $G = (V,E)$. This leads to efficient solutions for (moderately) sparse dags only. However, the following lemma enables significant improvements for dense instances in the majority of cases.

Lemma 3.1. *For a dag $G = (V,E)$ and vertices $x,y,z \in V$, it holds that $z \in \text{LCA}_G\{x,y\}$ if and only if $z \in \text{LCA}_{G_{\text{red}}}\{x,y\}$.*

Proof. Suppose, for the sake of contradiction, that z is an 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 $\{x,y\}$ such that z reaches z' in G_{red}. Since G and G_{red} share the same transitive closure by definition, both implies that z cannot be an LCA of x and y in G, contradicting the assumption. The opposite direction is analogous. ■

Computing Transitive Closures and Reductions

A computational equivalence between transitive closure and transitive reduction computation on a given dag G was shown by Aho, Garey, and Ullman [AGU72]. Moreover, both problems are computational equivalent to Boolean matrix multiplication. This implies a general upper bound of $O(n^\omega)$

for both problems, where ω is the exponent of square matrix multiplication. For more details on algorithms using matrix multiplication we refer to Chapter 4. The following more practical result is due to Simon [Sim88].

Proposition 3.2. *The transitive closure G_{clo} and the transitive reduction G_{red} of a dag $G = (V,E)$ can be computed in time $O(m_{\text{clo}} + m_{\text{red}} \cdot k)$, where k is the width of a greedily constructed but not necessarily minimum chain cover, m_{clo} is the number of edges in G_{clo} and m_{red} is the number of edges in G_{red}.*

In practice Simon's algorithm constructs the transitive closure and/or reduction of a dag reasonably fast, i.e., in time close to the optimum of $O(n^2)$. Since we use the algorithm throughout this work, we give a brief informal description in this place.

The algorithm is an improvement of an earlier algorithm given by Goralćíková and Koubek [GK79]. The basic observation of their approach can be summarized as follows. Let $v \in V$ be a vertex and let $N^{\text{out}}(v) = \{v_1, \ldots, v_k\}$ be the out-neighborhood of v in increasing topological order. Then

- $N^{\text{out}}_{\text{clo}}(v) = v \cup \bigcup_{1 \le i \le k} N^{\text{out}}_{\text{clo}}(v_i)$.
- $N^{\text{out}}_{\text{red}}(v) = \{v_l \in N^{\text{out}}(v) | v_l \notin \bigcup_{1 \le i \le l-1} N^{\text{out}}_{\text{clo}}(v_i)\}$ for all $1 \le l \le k$.

The above observation translates easily to an algorithm with running time $O(n m_{\text{red}})$. The improvement to $O(k m_{\text{red}})$ relies on using a greedily constructed chain cover of the dag to prune out unnecessary operations. Recall that a chain cover of a dag $G = (V,E)$ is a set of vertex-disjoint paths $\mathscr{C} = \{c_1, \ldots, c_k\}$ such that $\bigcup_{1 \le i \le k} c_i = V$. Observe that here, a chain cover is constructed on the non-transitive dag G.

A chain cover can be greedily constructed by simply repeating the following until the whole vertex set V of $G = (V,E)$ is covered: Identify an uncovered vertex with minimum topological number and construct a path by always adding the next uncovered child with the minimum topological number, see Algorithm 1. In the rest of this chapter, we refer to chain covers as covers constructed by Algorithm 1. We note at this point that we use another greedy chain cover construction in Chapter 5. While both follow the greedy paradigm, we stress the fact that the approaches are not equivalent.

The improvement over the method by Goralćíková and Koubek is based on the following observation. Recall again that we compare vertices with respect to some topological ordering top. That is, for $u, v \in V$, $u > v$ if and only if $top(u) > top(v)$.

- $N^{\text{out}}_{\text{clo}}(v) = v \cup \bigcup_{1 \le i \le k} \{w \in c_i | w \ge \min\{N^{\text{out}}_{\text{clo}}(v) \cap c_i\}\}$

Thus, it is sufficient to compute the minimum vertices of the intersections of transitive closure out-neighborhoods and the chains. The complexity for this task can be bounded by $O(k m_{\text{red}})$, for details we refer to [Sim88].

Average Case Analysis of Graph Algorithms

Throughout this chapter and occasionally in the rest of this thesis we perform average case analysis of the algorithms presented. To this end, we assume a probabilistic distribution of the input space, see Section 2.2, Equation (2.3). In the average case analysis of problems on undirected graphs, the most prominent probabilistic models are the $G_{n,p}$ and the $G_{n,m}$ model, see, e.g., [Bol01]. Let in the

Algorithm 1: Greedy Chain Cover Construction

Input: A dag $G = (V, E)$.
Output: A chain cover $\mathscr{C} = c_1, \ldots, c_k$ of the dag.

```
1  begin
2      Initialize an empty chain cover 𝒞.
3      while there exists an uncovered vertex do
4          Initialize a chain c with start vertex v, where v is the uncovered vertex with minimum
           label.
5          while there exists an uncovered child of v do
6              Let u be the child of v with minimum label.
7              Add u to c and set v ← u.
8          end
9          Add c to 𝒞.
10     end
11 end
```

following N be the maximum number of edges in a graph. For example, if we consider undirected simple graphs without loops, $N = \binom{n}{2}$.

$G_{n,p}$ In the $G_{n,p}$ model each possible edge of a graph $G = (V, E)$ with $|V| = n$ is chosen with a prescribed probability $0 < p < 1$. Thus, for a given (labeled) graph $G = (V, E)$ with m edges, the probability that G is chosen from an input space that is modeled according to the $G_{n,p}$ model is given by $\Pr[G] = p^m(1-p)^{N-m}$.

$G_{n,m}$ In the $G_{n,m}$ model, the graphs with m edges are assumed to be uniformly distributed, that is, given two arbitrary (labeled) graphs G_1 and G_2 with m edges, we have $\Pr[G_1] = \Pr[G_2] = 1/\binom{N}{m}$.

Both models are readily extended to directed acyclic graphs by simply fixing an arbitrary order of the vertices and directing each edge from the smaller to the larger vertex with respect to this order. The $G_{n,p}$ model was extended in this way by Barak and Erdős themselves [BE84] and is sometimes referred to as *random order model*. In the rest of this work we implicitly assume this extension whenever referring to $G_{n,p}$ and $G_{n,m}$ random dags.

The average case analysis in this chapter is restricted to $G_{n,p}$ random dags. However, it should be noted that in many cases the asymptotic properties of random dags in the $G_{n,p}$ and $G_{n,m}$ models are practically identical if $p \cdot N$ is close to m. The following proposition was given in [AF07]. It is a generalization of an analogous result on general graphs [Bol01].

Proposition 3.3. *Let f be a non-negative function defined over the set of dags with n vertices such that $f(G) \leq f(H)$ for $G \subseteq H$. Let $\mathbb{E}_p[f]$ and $\mathbb{E}_m(f)$ denote the expected value of f with respect to random dags in the $G_{n,p}$ and $G_{n,m}$ models.*

1. *If* $\lim_{n\to\infty} p(1-p)N = \lim_{n\to\infty} \frac{pN-m}{\sqrt{p(1-p)N}} = \infty$ *then* $\mathbb{E}_m(f) \leq \mathbb{E}_p(f) + o(1)$.
2. *If* $\lim_{n\to\infty} p(1-p)N = \lim_{n\to\infty} \frac{m-Np}{\sqrt{p(1-p)N}} = \infty$ *then* $\mathbb{E}_p(f) \leq \mathbb{E}_m(f) + o(1)$.

As stated above a parameter of particular interest in the analysis of the algorithms described in this chapter is m_{red}, the number of edges in the transitive reduction of the input dag G. The expected value

of m_{red} in the $G_{n,p}$ model is given in the following proposition, which is due to Simon [Sim88]. We note that m_{red} is not monotonic in the sense of Proposition 3.3, i.e., $H \subseteq G$ does not imply that the transitive reduction of H is smaller than or equal to the transitive reduction of G. However, as a result of the experimental study conducted in Chapter 7, the expected values of m_{red} seem to be very similar for $G_{n,p}$ and $G_{n,m}$ random dags. In the following, however, we restrict our attention to the $G_{n,p}$ model without explicitly stating this.

Proposition 3.4. *Let $G = (V,E)$ be a random dag in the $G_{n,p}$ model. Let m_{red} be the number of edges in the transitive reduction. Then $\mathbb{E}[m_{\text{red}}] = O(n \log n)$.*

Moreover, Crippa improved earlier bounds on $\mathbb{E}[k\, m_{\text{red}}]$ to $O(n^2)$ in his PhD thesis [Cri94]. Recall that k is the cardinality of a chain cover constructed by Algorithm 1. This implies in particular the following proposition.

Proposition 3.5. *The transitive closure and reduction of a dag G with n vertices can be computed in time $O(n^2)$ in the average case.*

We extensively make use of the above two propositions in the remainder of this chapter. Observe that, e.g., a worst case bound of $O(n\, m_{\text{red}})$ implies an average case bound of $O(n^2 \log n)$ by linearity of expectation.

3.3 All-Pairs Representative LCA

We start by describing our dynamic programming approach for the ALL-PAIRS REPRESENTATIVE LCA problem. To this end, we make use of Proposition 2.10, i.e., we compute the maximum CAs in order to find representative LCAs. Recall again that we assume that the dag $G = (V,E)$ is equipped with some topological order $top : V \to \{1,\ldots,n\}$ and a comparison of two vertices x and y is with respect to top, i.e., $x < y$ if and only if $top(x) < top(y)$. Suppose further that reachability queries in G can be answered in constant time, i.e., G_{clo} is known. The idea of the approach is as follows. Given a vertex $x \in V$, suppose we want to determine the maximum CA for the pairs $\{x,y\}$, for all $y \in V$. Let $\{x_1,\ldots,x_l\}$ be the parents of x in G and suppose that $\text{MCA}\{x_i,y\}$ is known for all $1 \leq i \leq l$. The following lemma shows how to derive the maximum CA of $\{x,y\}$.

Lemma 3.6. *Let $G = (V,E)$ be a dag and let $\{x,y\}$ be a pair of vertices. Furthermore, let z be the maximum CA of $\{x,y\}$, i.e., $z = MCA\{x,y\}$.*

(i) If $x \rightsquigarrow y$, then $z = x$.

(ii) If y is not a successor of x, then the following holds: Let $\{x_1,\ldots,x_l\}$ be the parents of x. Let $Z = \bigcup_{1\leq i\leq l} MCA\{x_i,y\}$. Then z is the maximum vertex in Z.

Proof. If $(x,y) \in E_{\text{clo}}$, x is the only LCA of $\{x,y\}$ and hence the maximum CA. This proves the first statement.

For the second statement, suppose that y is not reachable from x. We only have to show that $z \in Z$ since $Z \subseteq \text{CA}\{x,y\}$. 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_k, $1 \leq k \leq l$, of x. Let now $z' = \text{MCA}(x_k, y)$. Observe the following:

1. $z \in \text{CA}\{x,y\}$ and $z \in \text{CA}(x_k,y)$
2. $z' \in \text{CA}\{x,y\}$ and $z' \in \text{CA}(x_k,y)$

Assume first that $top(z) > top(z')$. This contradicts the assumption that z' is the maximum CA of x_k and y. On the other hand, if $top(z) < top(z')$, z cannot be the maximum CA of x and y. It follows that $z \in Z$. ■

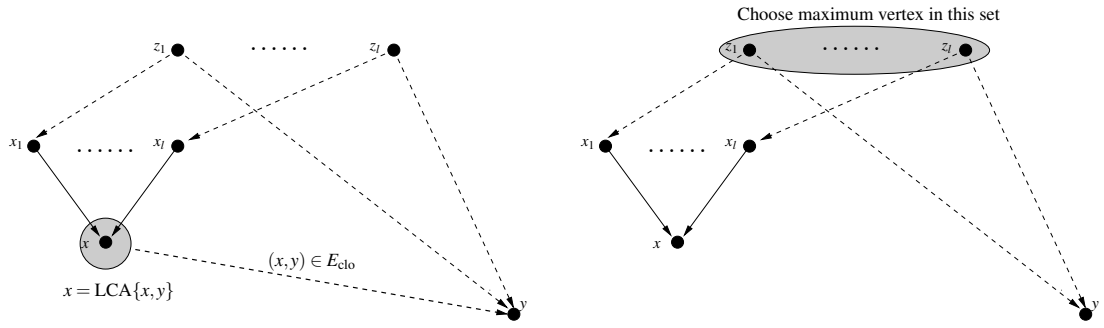


Figure 3.1: Idea of the dynamic programming approach for ALL-PAIRS REPRESENTATIVE LCA.

If we visit the vertices of V in ascending order with regard to top, the necessary maximum CAs are known and $\text{MCA}\{x,y\}$ can be determined by simply comparing the precomputed solutions. Observe that the design of this solution follows the dynamic programming paradigm. This is implemented in Algorithm 2.

Algorithm 2: Dynamic programming algorithm for ALL-PAIRS REPRESENTATIVE LCA

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

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

Theorem 3.7. *In a dag $G = (V,E)$ with n vertices, the time needed by the dynamic programming algorithm (Algorithm 2) to solve* ALL-PAIRS REPRESENTATIVE LCA *can be bounded by $O(n\,m_{\text{red}})$, where m_{red} is the number of edges in the transitive reduction of G.*

Proof. The correctness follows from Lemma 3.6. Observe that in Line 9 the maximum CA of $\{z,y\}$ is already determined since the vertices are visited in ascending topological order, i.e., all parents of x are already processed. Let $\mathscr{T}(G)$ be the running time of the algorithm on input G. The preprocessing steps can be implemented in time $O(n\,m_{\text{red}})$ by Proposition 3.2. We have

$$\mathscr{T}(G) = O(n\,m_{\text{red}}) + \sum_{v \in V} \text{indeg}_{G_{\text{red}}}(v) \cdot n = O(n\,m_{\text{red}}).$$

■

Recall again that for the purpose of average case analysis we assume that the input graphs are distributed according to the $G_{n,p}$ model. Proposition 3.4, i.e., $\mathbb{E}[m_{\text{red}}] = O(n \log n)$, yields the following.

Corollary 3.8. *In a dag $G = (V,E)$ with n vertices, the time needed by the dynamic programming algorithm (Algorithm 2) to solve* ALL-PAIRS REPRESENTATIVE LCA *can be bounded by $O(n^2 \log n)$ in the average case.*

3.4 All-Pairs All LCA

We turn our attention to the ALL-PAIRS ALL LCA problem. Before we embark on applying our dynamic programming approach, we start by specifying a simple, natural algorithm. First, it computes the transitive closure and reduction of G in time $O(n\,m_{\text{red}})$. Then, for each vertex z and each pair $\{x,y\}$, it determines in time $O(\text{outdeg}(z))$ (with respect to G_{red}) if z is an LCA of $\{x,y\}$. If z is a CA of $\{x,y\}$, but none of its children, then z is an 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 algorithm is $O(n^2\,m_{\text{red}})$.

Algorithm 3: Algorithm for ALL-PAIRS ALL LCA based on transitive closure look-ups

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.

```
1 begin
2     Compute a topological order top.
3     Compute the transitive closure G_clo and the transitive reduction G_red of G.
4     foreach z ∈ V in ascending order with respect to top do
5         foreach x, y ∈ V with z ≤ x, y do
6             if z ∈ CA{x,y} and w ∉ CA{x,y} for all w ∈ N^out_red(z) then add z to A[x,y]
7         end
8     end
9 end
```

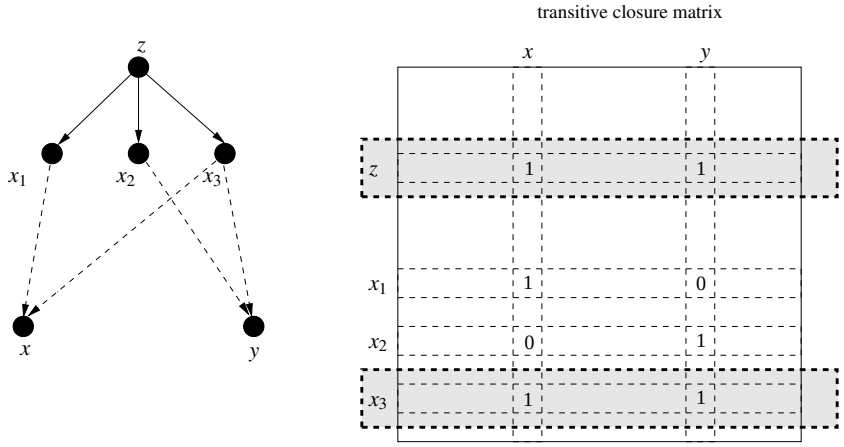


Figure 3.2: Idea of Algorithm 3. Observe that x_3 is a witness for z and $\{x,y\}$.

Theorem 3.9. *For a dag $G=(V,E)$ with n vertices the running time needed by Algorithm 3 to solve* ALL-PAIRS ALL LCA *is bounded by $O(n^2 m_{\text{red}})$, where m_{red} is the number of edges in the transitive reduction of G.*

Proof. Let $\mathscr{T}(G)$ be the running time of Algorithm 3 on $G=(V,E)$. Let $V=\{v_1,\ldots,v_n\}$ be the vertex set such that the vertices are ordered in ascending topological order. The preprocessing steps can be implemented in time $O(n\,m_{\text{red}})$. Let again $\mathscr{T}(G)$ be the running time of Algorithm 3. Decomposing the structure of the main loop of the algorithm, we find

$$\begin{aligned}\mathscr{T}(G) &= \sum_{i=1}^{n}\left(\sum_{j=i}^{n}\sum_{k=j}^{n}\text{outdeg}_{G_{\text{red}}}(v_i)\right)\\ &= O\left(\sum_{i=1}^{n}\text{outdeg}_{G_{\text{red}}}(v_i)\binom{n-i+1}{2}\right)\\ &= O(n^2 m_{\text{red}}).\end{aligned}$$

■

Observe that Algorithm 3 has a lower bound of $\Omega(n^3)$ whenever there is a linear fraction of vertices with at least one outgoing edge, i.e., in all but particularly degenerate cases. The next corollary is again a direct consequence of Proposition 3.4.

Corollary 3.10. *For a dag $G=(V,E)$ with n vertices, the running time needed by Algorithm 3 to solve* ALL-PAIRS ALL LCA *is bounded by $O(n^3 \log n)$ in the average case.*

We proceed by giving an $O(n^2 m_{\text{red}})$ dynamic programming approach which does not improve the naive algorithm in the worst case but turns out to be very efficient in practice, see Section 7. This

algorithm adopts ideas from Algorithm 2. Suppose that we want to determine $\mathrm{LCA}\{x,y\}$ and suppose the sets $\mathrm{LCA}\{x_1,y\},\ldots,\mathrm{LCA}\{x_l,y\}$ are known for all parents $\{x_1,\ldots,x_l\}$ of x. Instead of choosing the maximum CA out of the solutions for the parents, we "merge" the respective LCA sets in order to obtain $\mathrm{LCA}\{x,y\}$. Recall that $z \in \mathrm{CA}\{x,y\}$ is an LCA of x and y if there is no other vertex $z' \in \mathrm{CA}\{x,y\}$ such that $(z,z') \in E_{\mathrm{clo}}$.

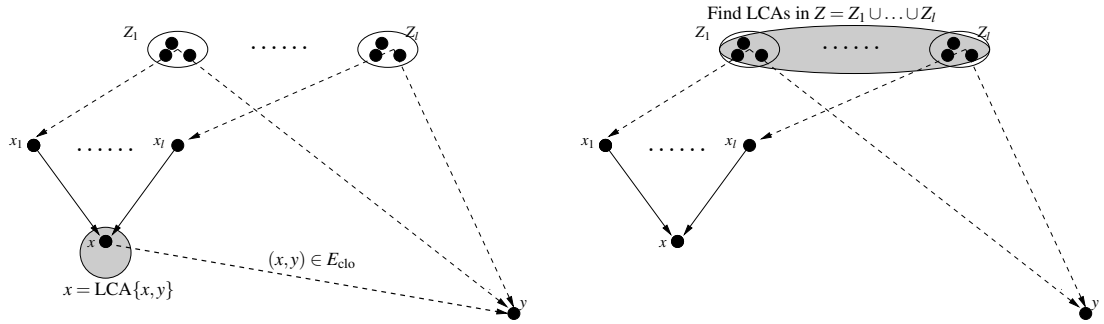


Figure 3.3: Idea of the dynamic programming approach for ALL-PAIRS ALL LCA.

The following is a generalization of Lemma 3.6.

Lemma 3.11. *Let $G=(V,E)$ be a dag. Let x and y be vertices of G. Let $\{x_1,\ldots,x_l\}$ be the parents of x and let Z be the union of the sets $\mathrm{LCA}\{x_i,y\}$ for all $1 \le i \le l$, i.e., $Z = \bigcup_{1\le i\le l}\mathrm{LCA}\{x_i,y\}$. Then*

(i) If $(x,y) \in E_{\mathrm{clo}}$, then $\mathrm{LCA}\{x,y\} = \{x\}$.

(ii) If $(x,y) \notin E_{\mathrm{clo}}$, then $z \in \mathrm{LCA}\{x,y\}$ if and only if $z \in Z$ and $\nexists z' \in Z$ such that $z \rightsquigarrow z'$.

Proof. The first statement is trivial. For the second statement, we observe that

$$\mathrm{LCA}\{x,y\} \subseteq Z \subseteq \mathrm{CA}\{x,y\}. \tag{3.1}$$

The second inclusion holds since $Z \subseteq \mathrm{CA}\{x,y\}$ by construction. For the first inclusion let $z \in \mathrm{LCA}\{x,y\}$ be an arbitrary LCA of x and y and suppose that $z \notin Z$. Obviously, since $z \neq x$, there is a path from z to x through some parent x_k of x. Thus, $z \in \mathrm{CA}\{x_k,y\}$. By assumption, z is not an LCA of x_k and y. This implies that there is a witness w for z and $\{x_k,y\}$ and $w \in \mathrm{LCA}\{x_k,y\}$ (Obs. 2.9). Yet, as stated above, every CA of x_k and y is also a CA of x and y. Hence, w is also a witness to the fact that z is not an LCA of x and y contradicting our assumption.

However, Equation (3.1) implies statement (ii). To see this, consider the subdag G^Z that is induced by the vertices of Z. Each $z \in Z$ such that $z \notin \mathrm{LCA}\{x,y\}$ has at least one child, namely a witness w such that $w \in \mathrm{LCA}\{x,y\}$ (which exists by Observation 2.9) since $\mathrm{LCA}\{x,y\} \subseteq Z$. On the other hand, since $Z \subseteq \mathrm{CA}\{x,y\}$, each $z \in Z$ such that $z \in \mathrm{LCA}\{x,y\}$ has no outgoing edges in G^Z. ■

Lemma 3.11 is implemented by Algorithm 4. We still have to specify the merge operation in Line 8. We propose two different strategies. The efficiency of each of the operations depends effectively on the maximum cardinality κ of an LCA set in G, see Equation (3.2). More specifically, we will elaborate below that the merging cost for a vertex x is bounded by $O\left(\mathrm{indeg}(x)\cdot\min\{(\kappa\log n+\kappa^2),n\}\right)$ by an

Algorithm 4: Dynamic programming algorithm for ALL-PAIRS ALL LCA

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.

```
1  begin
2      Compute the transitive closure G_clo and the transitive reduction G_red of G.
3      Compute a topological order top.
4      foreach x ∈ V in ascending order of top(x) do
5          foreach y ∈ V do
6              if (x, y) ∈ E_clo then A[x, y] ← {x}
7              else Let x_1, ..., x_l be the parents of x.
8              A[x, y] ← Merge(A[x_1, y], ..., A[v_l, y])
9          end
10     end
11 end
```

appropriate choice of the merging method. Observe that $\text{indeg}(x)$ is with respect to G_{red}; we omit the subscript to ease exposition. Let in the following $x, y \in V$ such that $(x, y) \notin E_{\text{clo}}$ and let $\{x_1, \ldots, x_l\}$ be the parents of x. Let $Z_i = \text{LCA}\{x_i, y\}$ for all $1 \leq i \leq l$.

- **Iterative Merging**: Initialize an empty set Z^M and merge $Z_1, \ldots, Z_l$ iteratively to Z^M as follows. Let $z \in Z_i$. Add z to Z^M if and only if z is not a predecessor of any of the vertices in the current set Z^M. Conversely, let $z \in Z^M$. Retain z in Z^M if and only if z is not a predecessor of any vertex in Z_i. Finally, set $\text{LCA}\{x, y\} \leftarrow Z^M$.
- **Lazy Merging**: Construct a multi-set $C = \text{LCA}(x_1, y) \uplus \ldots \uplus \text{LCA}(x_l, y)$. Sort the vertices in C in descending order according to their topological number and remove non-unique vertices. Initialize an empty set Z^M. Process the vertices in descending order as follows. Add $z \in C$ to Z^M if and only if z is not a predecessor of any vertex in the current set Z^M. Finally, set $\text{LCA}\{x, y\} \leftarrow Z^M$.

Before analyzing the time complexities of the two merging approaches, we proceed by verifying the correctness.

Lemma 3.12. *For vertices $x, y \in V$ such that $(x, y) \notin E_{\text{clo}}$ and $\{x_1, \ldots, x_l\}$ are the parents of x, the following holds.*

(i) The iterative merging method correctly constructs $\text{LCA}\{x, y\}$.

(ii) The lazy merging method correctly constructs $\text{LCA}\{x, y\}$.

Proof. For the rest of the proof, let again $Z_i = \text{LCA}\{x_i, y\}$ for all $1 \leq i \leq l$ and $Z = \bigcup_{1 \leq i \leq l} Z_i$. Again, we observe

$$\text{LCA}\{x, y\} \subseteq Z \subseteq \text{CA}\{x, y\},$$

compare the proof of Lemma 3.11.

(*i*) Let $A[x, y] = Z'$ after the iterative merging procedure. Let $z \in \text{LCA}\{x, y\}$ and suppose for the sake of contradiction that $z \notin Z'$. Since $\text{LCA}\{x, y\} \subseteq Z$ this implies that z is discarded in

some merging step. This in turn implies $z \rightsquigarrow z'$ for some vertex $z' \in Z$ distinct from z. Since $Z \in \mathrm{CA}\{x,y\}$, z' is a witness for z and $\{x,y\}$, a contradiction.

On the other hand, it is easy to see that each $z \in Z$ such that $z \notin \mathrm{LCA}\{x,y\}$ is discarded. By definition, there exists a witness w such that $w \in \mathrm{LCA}\{x_i,y\}$ and $w \in \mathrm{LCA}\{x,y\}$, compare Observation 2.9. Since $w \in Z$ and no LCA of $\{x,y\}$ is discarded during the iterative merging, the reachability of z and w is tested at some point of the iterative merging procedure and z is discarded.

(*ii*) Let $Z = \{z_1,\ldots,z_{|Z|}\}$ such that the vertices in Z are sorted in descending order with respect to their topological number.
Claim 3.13. *For all* $1 \leq j \leq |Z|$ *let* $\mathrm{LCA}^{(j)}\{x,y\}$ *be the intermediate LCA set of* $\{x,y\}$ *after vertex* z_j *has been processed. The following is true for all vertices* $z \in V$.

a) If $z \in \mathrm{LCA}\{x,y\}$ *and* $top(z) \geq top(z_j)$, *then* $z \in \mathrm{LCA}^{(j)}\{x,y\}$.

b) If $z \in \mathrm{LCA}^{(j)}$, *then* $z \in \mathrm{LCA}\{x,y\}$.

Proof. We establish this claim by induction on j. For $j = 1$, the claim is true by Proposition 2.10. Suppose now, the claim is true through $j-1$. We only have to show that z_j is added to $\mathrm{LCA}^{(j-1)}\{x,y\}$ if and only if $z_j \in \mathrm{LCA}\{x,y\}$. Suppose first that $z_j \in \mathrm{LCA}\{x,y\}$. Then, by the induction hypothesis, z_j is not a predecessor of any of the vertices in $\mathrm{LCA}^{(j-1)}\{x,y\}$ and is hence added. Assume now that $z_j \notin \mathrm{LCA}\{x,y\}$. We observe that there exists a witness w for z_j such that $w \in \mathrm{LCA}\{x,y\}$ (Obs.2.9) and $top(w) > top(z_j)$. Again, by the induction hypothesis, this implies that $w \in \mathrm{LCA}^{(j-1)}\{x,y\}$ and hence z_j is discarded. ■

The correctness of the lazy merging approach is immediate from the above claim.

■

Since in Algorithm 4 the vertices are visited in ascending order with respect to a topological order top, $A[x_i,y]$ corresponds to $\mathrm{LCA}\{x_i,y\}$ by the time that x is visited. All parents of x are visited before x. This and Lemmas 3.11 and 3.12 establish the correctness of Algorithm 4.

Before we bound the times needed by the two merging methods, we introduce a parameter κ, where

$$\kappa = \max_{x,y \in V}\{|\mathrm{LCA}\{x,y\}|\} \tag{3.2}$$

is the maximum cardinality of an LCA set in G. Recall further that the width $\mathrm{w}(G)$ of a dag G is defined as the maximum cardinality of an antichain in G.

Lemma 3.14. *For vertices* $x,y \in V$ *such that* $(x,y) \notin E_{\mathrm{clo}}$ *and* $\{x_1,\ldots,x_l\}$ *are the parents of* x, *the time needed by the merging methods to construct* $\mathrm{LCA}\{x,y\}$ *can be bounded as follows.*

(i) **Iterative Merging***:* $O(indeg(x) \cdot \min\{\mathrm{w}(G)\kappa, n\})$

(ii) **Lazy Merging** *:* $O(indeg(x)\kappa \log(indeg(x)\kappa) + indeg(x)\kappa^2)$

Proof. Recall again that $\{x_1,\ldots,x_l\}$ are the parents of x and let $Z_i = \mathrm{LCA}\{x_i,y\}$ for all $1 \leq i \leq l$.

(*i*) **Iterative Merging**: Let Z^M be an intermediate LCA set of $\{x,y\}$ and assume Z_i is supposed to be merged to Z^M. The straightforward approach simply checks pairwise reachability by transitive closure look-up in time $O(|Z_i| \cdot |Z^M|)$. Since all intermediate vertex sets are antichains (easy induction), the naive approach gives an upper time bound of $O(\mathrm{w}(G)\kappa)$.

For large sets, this can be improved to $O(n)$ as follows, see also Algorithm 5. To ease exposition we introduce some additional notation:

- Let $V' \subseteq V$ be any subset of V. We denote by $\overline{V'}$ the *forbidden set* of V', where $u \in \overline{V'}$ if and only if u is a predecessor of some vertex $v \in V'$.
- For all $1 \leq i \leq l$ we denote by $Z^{(i)}$ the intermediate set LCA$\{x,y\}$ after the sets $Z_1,\ldots,Z_i$ have been merged to the (initially empty) set Z^M.

Assume now that we want to merge Z_i to $Z^{(i-1)}$ and that we possess forbidden sets $\overline{Z^{(i-1)}}$ and $\overline{Z_i}$. Then, for any $z \in Z_i$, we merge z to $Z^{(i-1)}$ if $z \notin \overline{Z^{(i)}}$. Conversely, we retain $z \in Z^{(i-1)}$ if $z \notin \overline{Z_i}$. If the forbidden sets are maintained as bit vectors of length n the membership of z in the forbidden set can be checked in constant time and the two sets can be merged in time $O\big(|Z^{(i-1)}| + |Z_i|\big)$. The bottleneck of this merge operation is updating the forbidden set $\overline{Z^{(i)}}$ for $Z^{(i)}$, which is done by a *bitwise or* combination of $\overline{Z^{(i-1)}}$ and $\overline{Z_i}$. Thus, the merge operation takes time $\Theta(n)$.

(*ii*) **Lazy Merging**: The running time of the lazy merging approach can be bounded as follows.

- The construction of $C = Z_1 \uplus \ldots \uplus Z_l$ takes $\sum_{i=1}^{l} |Z_i|$ and can be bounded by $O(\mathrm{indeg}(x) \cdot \kappa)$.
- Sorting and removing non-unique vertices in C can be bounded by

$$O\left(\min\left\{\left(\sum_{i=1}^{l}|Z_i|\right)\log\left(\sum_{i=1}^{l}|Z_i|\right), n+\sum_{i=1}^{l}|Z_i|\right\}\right).$$

  To see this, observe that we can use bucket sort whenever

$$\left(\sum_{i=1}^{l}|Z_i|\right)\log\left(\sum_{i=1}^{l}|Z_i|\right) > n+\sum_{i=1}^{l}|Z_i|.$$

  However, in order to prove the lemma, it is enough to bound this step by $O((\mathrm{indeg}(x)\cdot\kappa\cdot \log(\mathrm{indeg}(x)\cdot\kappa))$.

- Finally, we check in time $O(\kappa)$ whether a vertex z is to be retained or discarded. Hence, we can bound this step by $O(\mathrm{indeg}(x)\kappa^2)$.

■

Theorem 3.15. *In a dag G with n vertices, the time needed by the dynamic programming algorithm (Algorithm 4) to solve* ALL-PAIRS ALL LCA *can be bounded by $O(n m_{\mathrm{red}} \cdot \min\{\left(\kappa^2 + \kappa\log n\right), n\})$, where m_{red} is the number of edges in the transitive reduction of G and κ is the maximum LCA set cardinality in G.*

Proof. Recall that we use transitive reduction as speed-up. We combine the two merging strategies as follows. Start with the lazy merging approach. If for some pair $\{x,y\}$ the merging cost exceeds some prescribed linear threshold, start anew with iterative merging.

By Lemma 3.14 the time for an iterative merging operation can be bounded by $O(\min\{\mathrm{w}(G)\kappa, n\})$. Let $\mathscr{T}(G)$ be the running time of Algorithm 4 on $G = (V,E)$. We get

$$\mathscr{T}(G) = \sum_{v \in V} \mathrm{indeg}(v) O(n) O(\min\{\mathrm{w}(G)\kappa, n\}) = O(n m_{\mathrm{red}} \min\{\mathrm{w}(G)\kappa, n\}). \tag{3.3}$$

For the lazy merging operation we have derived an upper bound of $O(\mathrm{indeg}(v)\kappa \cdot \log(\mathrm{indeg}(v)\kappa) + \mathrm{indeg}(v)\kappa^2)$. Hence:

$$\begin{aligned}
\mathscr{T}(G) &= \sum_{v \in V} O(\mathrm{indeg}(v)\kappa \log(\mathrm{indeg}(v)\kappa) + \mathrm{indeg}(v)\kappa^2) \cdot n \\
&= \sum_{v \in V} O(\mathrm{indeg}(v)\kappa (\log(\mathrm{indeg}(v)) + \log(\kappa)) + \mathrm{indeg}(v)\kappa^2) \cdot n \\
&\leq \sum_{v \in V} O(\mathrm{indeg}(v)\kappa (\log(n) + \log(\kappa)) + \mathrm{indeg}(v)\kappa^2) \cdot n \\
&= O(m_{\mathrm{red}} n \kappa \log n) + O(m_{\mathrm{red}} n \kappa^2)
\end{aligned} \tag{3.4}$$

The theorem follows by combining Equations (3.3) and (3.4). ■

Again, the next corollary is a consequence of Theorem 3.15 and Proposition 3.4.

Corollary 3.16. *In a dag $G = (V,E)$ with n vertices, the time needed by the dynamic programming algorithm (Algorithm 4) to solve* ALL-PAIRS ALL LCA *on G can be bounded by $O(n^2 \log n \cdot \min\{(\kappa^2 + \kappa \log n), n\})$ in the average case.*

Algorithm 5: Merge with Forbidden Sets

Input: Sets S_1 and S_2 and the corresponding forbidden sets $\overline{S_1}$ and $\overline{S_2}$.
Output: A new set S and a new forbidden set $\overline{S}$.

```
1  begin
2      S ← ∅
3      foreach s1 ∈ S1 do
4          if s1 ∉ S̄2 then
5              S ← S ∪ {s1}
6          end
7      end
8      foreach s2 ∈ S2 do
9          if s2 ∉ S̄1 then
10             S ← S ∪ {s2}
11         end
12     end
13     for i ← 1 to n do
14         S̄[i] ← S̄1[i] ∨ S̄2[i]
15     end
16 end
```

The advantage of this dynamic-programming-based algorithm over the trivial one is that if we can upper bound the size κ by $o(\sqrt{n})$ we have an improved upper bound on the running time. In fact,

the experimental evaluation of the algorithms in Section 7 reveals that κ can usually be bounded by a constant. This is the main reason for the fact that Algorithm 4 is the best choice in practice.

Moreover, the bounds on the running time given in Theorem 3.15 may be overly pessimistic due to rough estimates used in the proof of Lemma 3.14 (e.g., $|\mathrm{LCA}\{x_i, y\}| = \kappa$ or $\log(\mathit{indeg}(v)) = \log(n)$). Experimental evidence supports the conjecture that Algorithm 4 actually runs in time close to $O(n^2)$ in practice on most of the tested dag classes.

As an immediate consequence we obtain fast algorithms for testing lattice-theoretic properties of posets represented by dags. Observe that $\kappa = 1$ in lattices.

Corollary 3.17. *Testing whether a given dag is a lower semilattice, an upper semilattice, or a lattice can be done in time $O(n\, m_{\mathrm{red}} \log n)$.*

3.5 Summary

Dynamic programming approaches find application in the realm of all-pairs LCA problems in a natural way. Essentially, the combinatorial properties expressed in Lemmas 3.6 and 3.11, i.e., solutions with respect to a vertex x can be derived from the solutions for its parent vertices, impose the kind of subproblem order that is necessary for applying efficient dynamic programming algorithms. While the main benefit of the pure dynamic programming approach is the efficiency on sparse input dags, the combination with using the transitive reduction makes the algorithmic tools described in this chapter versatile and powerful. Observe that the average complexities of the algorithms are at least close (up to logarithmic factors) to the trivial lower bounds for both ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA. Moreover, Theorem 3.15 implies that if the maximum LCA set size κ can be bounded by a constant, ALL-PAIRS ALL LCA can be solved in time $\widetilde{O}(n^2)$, which is a significant improvement of an order of magnitude compared to the by now optimal matrix-multiplication-based solutions, see Chapter 4 for details. As a result of the experimental analysis presented in Chapter 7, we conclude that κ is indeed small on all tested classes of input dags with exception of dags in which the number of edges is close to $n \log n$. However, while the ALL-PAIRS REPRESENTATIVE LCA solution is almost optimal in the average case, future research with respect to ALL-PAIRS ALL LCA might be directed towards finding algorithms that scale linearly with κ instead of quadratically. This would close the efficiency gap observed in practice between medium-sized ($m \approx n \log n$) on the one hand and sparse and dense input instances on the other hand.

Another open question in this context is if a bound on the average cardinality $\overline{\kappa}$ of the LCA sets implies a general improvement of the upper bound given in Theorem 3.15. That is, is the running time of Algorithm 4 in $o(n^2 m_{\mathrm{red}})$ in this case? At least for the dynamic programming algorithm without transitive reduction, the answer is "no", even for $\overline{\kappa} \leq c$ for a constant c, as the graph in Figure 3.4 shows. Since there are $\Omega(m)$ edges among vertices of V', the cost of merge operations associated with constructing the sets $\mathrm{LCA}\{w, y\}$ for $w \in V'$ alone amounts to $\Omega(n^2 m)$ 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. However, transitive reduction clearly eliminates this counterexample.

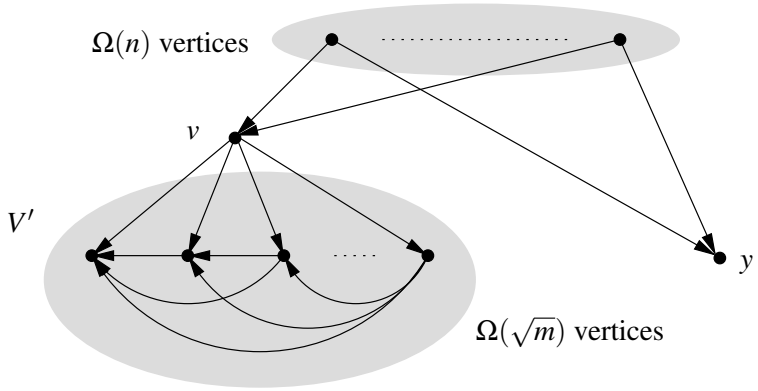


Figure 3.4: Bad graph for naive merging.

CHAPTER

4

MATRIX-MULTIPLICATION-BASED ALGORITHMS

4.1 Introduction

Matrix multiplication is a central problem in algorithmic linear algebra. This is particularly true since numerous other important problems have been shown to have the same algorithmic complexity, e.g., computing the matrix inverse, computing the determinant, or solving a system of linear equations. Furthermore, many algorithmic graph problems and algebraic problems are inherently connected. For example, Aho, Garey, and Ullman [AGU72] have established a computational equivalence between Boolean matrix multiplication and computing the transitive closure of a digraph, see Proposition 4.1. Another prominent example is the algorithm by Zwick [Zwi02] for solving the all-pairs shortest path problem in digraphs. Recently, matrix-multiplication-based approaches have been used to obtain improved dynamic algorithms for a variety of graph problems, see the thesis of Sankowski [San05] for details.

Proposition 4.1. *Let $G = (V, E)$ be a dag with n vertices. The following problems are computationally equivalent:*

(i) Computation of the transitive closure G_{clo} of G.

(ii) Computation of the transitive reduction G_{red} of G.

(iii) Multiplication of two Boolean $n \times n$ matrices.

A relationship between (lowest) common ancestors in dags and matrix multiplication was first observed by Bender et al. [BFCP+05]. The ALL-PAIRS COMMON ANCESTOR EXISTENCE problem is to determine for each vertex pair $\{x, y\}$ in a dag $G = (V, E)$ whether there exists a common ancestor of $\{x, y\}$ or not. The basic idea is to construct a dag G' from G and reduce each common ancestor

existence query in G to a reachability query in G', see Figure 4.1. A similar construction was also used to obtain the first subcubic ALL-PAIRS REPRESENTATIVE LCA algorithm, see [BFCP$^+$05] and Section 6.3. Recognizing its fundamental status, we briefly describe the construction below.

Let again $G = (V,E)$ be a dag with topologically sorted vertex set $V = \{v_1, \ldots, v_n\}$. Construct a dag $G' = (V', E')$ from G as follows.

- $V' = X \cup Y$, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$.
- $E' = E_X \cup E_Y \cup E''$, where
  - $E_X = (x_i, x_j)$ for all $1 \leq i, j \leq n$ such that $(v_j, v_i) \in E$.
  - $E_Y = (y_i, y_j)$ for all $1 \leq i, j \leq n$ such that $(v_i, v_j) \in E$.
  - $E'' = (x_i, y_i)$ for all $1 \leq i \leq n$.

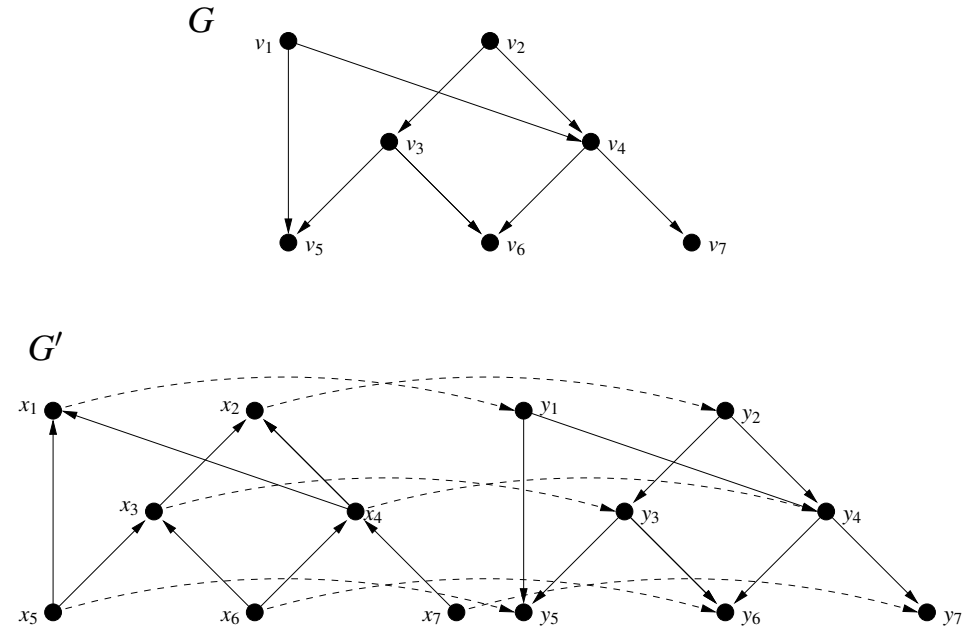


Figure 4.1: Construction of G' from G for reducing ALL-PAIRS COMMON ANCESTOR EXISTENCE to reachability. For all $1 \leq i, j \leq n$, $\mathrm{CA}\{v_i, v_j\} \neq \emptyset$ if and only if $x_i \rightsquigarrow y_j$ in G'. Roughly, G' is constructed from G by making a copy of G, reversing all edges in the copy, and connecting the respective vertex duplicates with edges.

It is not difficult to see that there exists a CA of $\{v_i, v_j\}$ in G if and only if y_j is reachable from x_i in G': $v_k \rightsquigarrow v_i$ and $v_k \rightsquigarrow v_j$ in G translates to $x_i \rightsquigarrow x_k$, (x_k, y_k), and $y_k \rightsquigarrow y_j$ in G'. This and Proposition 4.1 lead to the following proposition, which was first formulated in [BFCP$^+$05].

Proposition 4.2. *For a dag $G = (V,E)$ with n vertices, the* ALL-PAIRS COMMON ANCESTOR EXISTENCE *problem can be solved in $O(n^\omega)$.*

Recall that $O(n^\omega)$ is the number of arithmetic operations needed to multiply two $n \times n$ matrices. Another way of proving Proposition 4.2, which gives rise to the algorithmic techniques introduced in [CKL07], is as follows. Let A_{clo} be the adjacency matrix of G_{clo}. Obviously, $\mathrm{CA}\{x, y\} \neq \emptyset$ if and

only if there exists a vertex z such that $(z,x) \in E_{\text{clo}}$ and $(z,y) \in E_{\text{clo}}$. This easily reduces to matrix multiplication. Let $C = A_{\text{clo}}^T \cdot A_{\text{clo}}$. Observe that $A_{\text{clo}}^T[u,v] = 1$ if and only if $(v,u) \in E_{\text{clo}}$. Hence, there exists a common ancestor of $\{x,y\}$ if and only if $C[x,y] = 1$.

Kowaluk and Lingas [KL05] were the first to show that ALL-PAIRS REPRESENTATIVE LCA reduces to finding *maximum witnesses* for Boolean matrix multiplication. This reduction is a straightforward application of Proposition 2.10. Let A and B be two Boolean $n \times n$ matrices and let $C = A \cdot B$. Given i, j, $1 \leq i, j \leq n$ such that $C[i,j] = 1$, a maximum witness for (i,j) is the maximum index k such that $A[i,k] \cdot B[k,j] = 1$. Let now again $A = A_{\text{clo}}^T$ and $B = A_{\text{clo}}$ and suppose that the rows and columns of A_{clo} are sorted with respect to a topological ordering of the vertices. Further, let $C[x,y] = 1$ for $x, y \in V$. Then, a maximum witness for (x,y) corresponds to the maximum common ancestor of $\{x,y\}$ and hence to an LCA by Proposition 2.10.

Proposition 4.3. ALL-PAIRS REPRESENTATIVE LCA *on a dag $G = (V,E)$ with n vertices can be reduced to* ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION *for matrices of dimension $n \times n$.*

We note that representative CAs for all vertex pairs can be computed by determining arbitrary witnesses instead of maximum witnesses for the Boolean matrix product $C = A_{\text{clo}}^T \cdot A_{\text{clo}}$. This can be done in time $\widetilde{O}(n^\omega)$ using a sampling approach by Seidel [Sei95] and the according derandomization due to Alon and Naor [AN96]. The sampling technique is described in detail in the context of LCA problems in weighted dags, see Section 6.2 and in particular the proof of Theorem 6.2.

The main contributions of this chapter are improved general upper bounds for both the ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA problems by application of rectangular matrix multiplication methods. Moreover, we show how to improve the bounds for ALL-PAIRS ALL LCA if the considered dags are sparse. Additionally, we establish computational equivalences between problem variants of ALL-PAIRS ALL LCA in which one vertex is fixed and Boolean matrix multiplication. More specifically, we achieve the following results with regard to a dag G with n vertices:

- ALL-PAIRS REPRESENTATIVE LCA can be solved in time $O\left(n^{2+\mu}\right)$, where μ satisfies $1 + 2\mu = \omega(1,\mu,1)$. This and the currently best bounds for rectangular matrix multiplication imply an upper bound of $O(n^{2.575})$, which improves upon the $O(n^{2.616})$ bound of [KL05]. We note that the same improvement was discovered independently in [CKL07].
- ALL-PAIRS ALL LCA can can be solved in time $O\left(n^{\omega(2,1,1)}\right)$. Again, plugging in the current best upper bounds for matrix multiplication this implies $O(n^{3.334})$.
- The following upper time bounds for ALL-PAIRS ALL LCA improve upon the general upper bounds for sparse dag instances: $O\left(n^{\omega(1,r,1)+1}\right)$, where $r = \log_n(m_{\text{red}}/n)$ and m_{red} is the number of edges in the transitive reduction of G, and $O\left(n^{\omega(s,s,1)+1}\right)$, where $s = \log_n(m_{\text{clo}}/n)$ and m_{clo} is the number of edges in the transitive closure of G. We show that these values imply an improvement if we can bound m_{red} by $O(n^{1.92})$ or m_{clo} by $O(n^{1.96})$ respectively.
- The following problems are computationally equivalent: BOOLEAN MATRIX MULTIPLICATION, ALL-PAIRS FIXED-VERTEX LCA, FIXED-VERTEX-PAIRS ALL LCA.

We outline the contents of this chapter. First, we review briefly the relevant results on (rectangular) matrix multiplication and introduce the necessary notational concepts. In Section 4.3 we describe the improvement for the maximum witness computation by applying rectangular matrix multiplication. Section 4.4 is dedicated to ALL-PAIRS ALL LCA and the respective fixed-vertex variants. We start

by establishing the computational equivalence between BOOLEAN MATRIX MULTIPLICATION, ALL-PAIRS FIXED-VERTEX LCA, and FIXED-VERTEX-PAIRS ALL LCA and use these results to obtain an upper bound of $O(n^{1+\omega})$ for ALL-PAIRS ALL LCA. Then, we show how this can be improved to $O(n^{\omega(2,1,1)})$ by applying rectangular matrix multiplication. We conclude the section and the chapter by improving the upper bounds for sparse dags.

4.2 Preliminaries

Let $A, B \in \mathbb{R}^{n\times n}$ be two matrices with real-valued entries. The entry $C[i,j]$ of the matrix product $C = A \cdot B$ is defined as

$$C[i,j] = \sum_{k=1}^{n} A[i,k] \cdot B[k,j]. \tag{4.1}$$

Naively, a matrix product can be computed in time $O(n^3)$ using Equation (4.1). In this work we use *Boolean* matrix multiplication as fundamental algorithmic tool in the design of common ancestor algorithms. Formally, let $A, B \in \{0,1\}^{n\times n}$ be two Boolean matrices with entries from $\{0,1\}$. The entry $C[i,j]$ of the Boolean matrix product $C = A \cdot B$ is given by

$$C[i,j] = \bigvee_{k=1}^{n} A[i,k] \wedge B[k,j]. \tag{4.2}$$

Boolean matrix multiplication can be easily reduced to matrix multiplication over $\mathbb{R}$. We use this reduction implicitly in the rest of this work. That is, we transfer upper bounds for the general matrix multiplication problem to the Boolean case. However, we note that for Boolean matrix products a combinatorial improvement over the naive upper bound is given by Avlazarov et al. [ADKF70]. The so-called *four-Russian-algorithm*, named after its four Russian inventors, computes the Boolean matrix product in $O(n^3/\log n)$ time. The algorithm was later improved to $O(n^3/\log^2 n)$ by Rytter [Ryt84]. For more details on these algorithms, we refer to [Man89].

We introduce basic results and notational concepts. Let ω denote the exponent of square matrix multiplication, i.e., ω corresponds to the smallest constant such that two $n \times n$ matrices can be multiplied in time $O(n^\omega)$. The first subcubic algorithm with running time $O\left(n^{2.81}\right)$ was given by Strassen [Str69]. Currently, the best bound is due to Coppersmith and Winograd [CW90] and implies $\omega < 2.376$ (actually even $\omega \leq 2.376 - \delta$ for a constant $\delta > 0$, compare Section 2.2).

Some of the algorithms presented in this work rely on fast *rectangular* matrix multiplication. Let $\omega(a,b,c)$ denote the exponent of the multiplication of an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix over the reals. That is, given matrices $A \in \mathbb{R}^{a\times b}$ and $B \in \mathbb{R}^{b\times c}$ it is possible to compute $C = A \cdot B$ with $O\left(n^{\omega(a,b,c)}\right)$ arithmetic operations. For $a = b = c = 1$, $\omega(a,b,c)$ corresponds to the exponent of the multiplication of two square matrices. Let in the following α be a constant such that

$$\alpha = \sup\{0 \leq r \leq 1 : \omega(1,r,1) = 2 + o(1)\}. \tag{4.3}$$

Coppersmith [Cop97] has shown that $\alpha > 0.294$. The following proposition is a direct consequence of this result; a formal proof can be found in [HP98].

Proposition 4.4. *Let* $\omega = \omega(1,1,1) < 2.376$ *and let* $\alpha = \sup\{0 \le r \le 1 : \omega(1,r,1) = 2+o(1)\}$. *Then,*

$$\omega(1,r,1) \le \begin{cases} 2+o(1) & \text{if } 0 \le r \le \alpha \\ 2+\frac{\omega-2}{1-\alpha}(r-\alpha)+o(1) & \text{if } \alpha \le r \le 1. \end{cases} \tag{4.4}$$

The next proposition is again given by Huang and Pan [HP98] by extending the solutions in [Cop97].

Proposition 4.5. *Let* $\omega(2,1,1)$ *be the exponent of the multiplication of an* $n^2 \times n$ *and an* $n \times n$ *matrix. Then,* $\omega(2,1,1) < 3.334$.

4.3 Representative LCA

The solution for ALL-PAIRS REPRESENTATIVE LCA given by Kowaluk and Lingas [KL05] relies on computing maximum witnesses for Boolean matrix multiplication (Prop.4.3). Their algorithm for ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION has running time $O\left(n^{2+1/(4-\omega)}\right)$, which can be bounded by $O(n^{2.616})$ using current exponents. We describe how to improve their approach by using fast rectangular matrix multiplication for the maximum witness computations. In the following, we assume that we have already computed the adjacency matrix A_{clo} of the transitive closure G_{clo} of G in time $O(n^{\omega})$. Also, we assume implicitly that the rows and columns in A are ordered according to a topological sort of G's vertex set.

Let $\mu \in [0;1]$ be a parameter. In order to ease exposition, we assume in the following that n^{μ} and $n^{1-\mu}$ are integers. This does not affect the asymptotics of the results. We divide V into equal-sized sets $V_1, \ldots, V_r$ of consecutive vertices (with respect to the topological order), where $r = n^{1-\mu}$. Thus, the size of the sets is n^{μ}. Maximum witnesses for the Boolean matrix multiplication $A_{\text{clo}}^T \cdot A_{\text{clo}}$ are found in two steps:

1. For each (x,y), determine $l \in \{1,\ldots,r\}$ such that the maximum witness of (x,y) is in V_l.
2. For each (x,y) and respective l, search V_l exhaustively to find the maximum witness.

Step one can be solved by performing rectangular matrix multiplications. Before we describe the approach, we introduce the concept of *matrix sampling*, see also Figure 4.2.

Definition 4.6 (Matrix Sampling)**.** *Let* M *be an* $n \times m$ *matrix, and let* $I \subseteq \{1,2,\ldots,m\}$. *Then,* $M[*,I]$ *is defined to be the matrix that is composed of the columns of* M *whose indices belong to* I. *Conversely, for* $I \subseteq \{1,2,\ldots,n\}$, $M[I,*]$ *is the matrix composed of the rows of* M *whose indices belong to* I.

M

1	0	1	0
0	0	1	1
0	1	0	0
1	1	1	1

$M[*,I_1]$

1	1
0	1
0	0
1	1

$M[I_2,*]$

0	0	1	1
1	1	1	1

Figure 4.2: Example of matrix sampling with regard to a matrix M, $I_1 = \{1,3\}$ and $I_2 = \{2,4\}$.

Let in the following $C^{(i)} = A_{\text{clo}}^T[*,V_i] \cdot A_{\text{clo}}[V_i,*]$ for $1 \le i \le r$.

Observation 4.7. *A pair $\{x,y\}$ of vertices has a common ancestor $z \in V_i$ if and only if $C^{(i)}[x,y] = 1$.*

Hence, for $i \in \{1,\ldots,r\}$, we compute the $r = n^{1-\mu}$ (rectangular) matrix products $C^{(i)}$ in 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. Then, we choose for each (x,y) the maximum index l such that $C^{(l)}[x,y] = 1$, which can be done in $O\left(n^{1-\mu+2}\right)$. Since $\omega(1,\mu,1) \geq 2$ for all $\mu \in [0;1]$, the total time needed for step one is $O\left(n^{1-\mu+\omega(1,\mu,1)}\right)$.

For the second step, we simply search for each (x,y) and the corresponding index l the set V_l manually, that is, for each $z \in V_l$ in descending order until we find z such that both $z \rightsquigarrow x$ and $z \rightsquigarrow y$. This takes time $O(n^2 \cdot |V_l|) = O\left(n^{2+\mu}\right)$. We get the optimal complexity by balancing the costs of the first and second step, i.e., for μ satisfying $1-\mu+\omega(1,\mu,1) = 2+\mu$. Currently, the best known upper bounds for rectangular matrix multiplication [Cop97] imply $\mu < 0.575$. Throughout the rest of this work, we fix the meaning of μ in the above sense. That is, let μ be the constant satisfying $\omega(1,\mu,1) = 1+2\mu$.

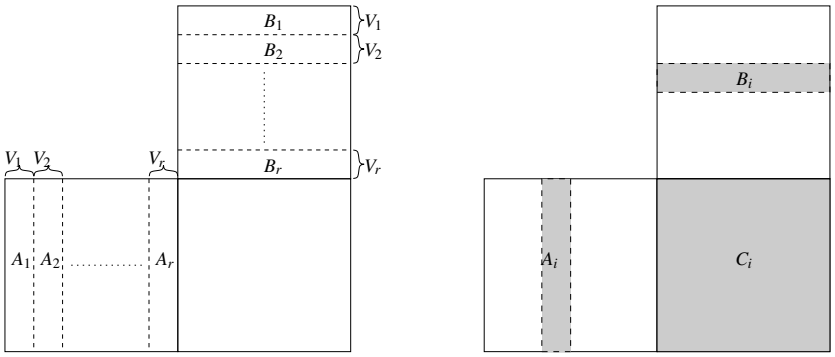


Figure 4.3: Idea for the decomposition of the square matrix product $C = A \cdot B$ into r rectangular matrix products $C_i = A_i \cdot B_i$ in the context of finding maximum witnesses. $C_i[x,y] = 1$ implies that $\mathrm{CA}\{x,y\} \cap V_i \neq \emptyset$.

Theorem 4.8. ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $O\left(n^{2+\mu}\right)$ on a dag G with n vertices, where μ satisfies $1+2\mu = \omega(1,\mu,1)$.*

4.4 All LCA

Solving ALL-PAIRS ALL LCA on a dag $G = (V,E)$ essentially corresponds to computing answers to the following kind of query for all $x,y,z \in V$: Is $z \in \mathrm{LCA}\{x,y\}$? Taking this perspective a natural subproblem can be described as follows. Compute answers to the above queries where one of the three input vertices is fixed, e.g., given $z \in V$, determine if $z \in \mathrm{LCA}\{x,y\}$ for all $x,y \in V$. This approach decomposes ALL-PAIRS ALL LCA into n subproblems. That is, fix vertex z for each $z \in V$ and solve the corresponding subproblem. An ALL-PAIRS ALL LCA solution can be easily obtained by combining the solutions to the subproblems.

To this end, we start by studying the *fixed-vertex variants* of ALL-PAIRS ALL LCA. Subsequently, we combine the solutions to the variants to obtain a tight upper bound for the general ALL-PAIRS ALL LCA problem. Furthermore, we proceed by exploiting sparseness of input instances in the two fixed-vertex variants in different ways. This leads to improved upper bounds for ALL-PAIRS ALL LCA for input dags having a sparse transitive reduction and/or sparse transitive closure.

4.4.1 Fixed-Vertex LCA Variants

We formally define the fixed-vertex LCA variants, which we call ALL-PAIRS FIXED-VERTEX LCA and FIXED-VERTEX-PAIRS ALL LCA respectively. Recall that we consider queries $z \in \text{LCA}\{x,y\}$ where one of the three vertices is fixed. We first fix z. The ALL-PAIRS FIXED-VERTEX LCA problem is defined as follows. Given a dag G and a fixed vertex z, find all pairs $\{x,y\}$ such that z is an LCA of $\{x,y\}$.

Problem:	ALL-PAIRS FIXED-VERTEX LCA (APFVLCA)
Input:	A dag $G=(V,E)$ and a vertex $z \in V$
Output:	A matrix L of size $n \times n$ such that for all vertices $x,y \in V$, $L[x,y]=1$ if and only if $z \in LCA\{x,y\}$ and $L[x,y]=0$ else.

The second kind of fixed-vertex LCA variant is derived from the queries $z \in \text{LCA}\{x,y\}$ by fixing either x or y. Observe, however, that fixing x is symmetric to fixing y. In the following we concern ourselves solely with fixing x. This gives rise to the problem FIXED-VERTEX-PAIRS ALL LCA. Given G and x, compute (the set) $\text{LCA}\{x,y\}$ for all $y \in V$.

Problem:	FIXED-VERTEX-PAIRS ALL LCA (FVPALCA)
Input:	A dag $G=(V,E)$ and a vertex $x \in V$
Output:	A matrix L of size $n \times n$ such that for all vertices $z,y \in V$, $L[z,y]=1$ if and only if $z \in LCA\{x,y\}$ and $L[z,y]=0$ else.

In fact, as we show below, both of the problems are computationally equivalent to BOOLEAN MATRIX MULTIPLICATION and hence to each other (Thm. 4.9). In the following we establish the respective theorem by proving the equivalence of BOOLEAN MATRIX MULTIPLICATION and ALL-PAIRS FIXED-VERTEX LCA (Lemmas 4.10 and 4.13) and, in a second step, the equivalence of BOOLEAN MATRIX MULTIPLICATION and FIXED-VERTEX-PAIRS ALL LCA (Lemmas 4.14 and 4.18).

Theorem 4.9. *The following problems are computationally equivalent:*

(i) BOOLEAN MATRIX MULTIPLICATION

(ii) ALL-PAIRS FIXED-VERTEX LCA

(iii) FIXED-VERTEX-PAIRS ALL LCA

All-Pairs Fixed-Vertex LCA

In the following let $z \in V$ be the fixed vertex. Let $\{x,y\}$, $x,y \in V$ be a pair of vertices such that $z \in \text{CA}\{x,y\}$. Recall that a witness to the fact that $z \notin \text{LCA}\{x,y\}$ is a vertex w such that $w \in \text{CA}\{x,y\}$

and $z \rightsquigarrow w$. At the high level a solution to the ALL-PAIRS FIXED-VERTEX LCA problem for z in G works as follows.

1. Determine all pairs $\{x,y\}$ such that $z \in \mathrm{CA}\{x,y\}$.
2. Test for each such pair if there exists a witness, i.e., a successor of z that is also a CA of this pair. To this end, it is enough to consider the children of z in G by Observation 2.9.

Lemma 4.10. ALL-PAIRS FIXED-VERTEX LCA *on a dag $G=(V,E)$ with n vertices can be reduced to* BOOLEAN MATRIX MULTIPLICATION *of two $n \times n$ matrices in time $O(n^\omega)$.*

Proof. We show that ALL-PAIRS FIXED-VERTEX LCA can be reduced to BOOLEAN MATRIX MULTIPLICATION by specifying an implementation to the above high-level description of a solution.

Let $G=(V,E)$ and $z \in V$ be the input to ALL-PAIRS FIXED-VERTEX LCA. We start by computing G_{clo} in time $O(n^\omega)$. Recall that computing G_{clo} and BOOLEAN MATRIX MULTIPLICATION are computational equivalent by Proposition 4.1. Let A_{clo} be again the adjacency matrix of G_{clo}. We initialize a *common ancestor matrix* $\mathrm{C}^{(z)}$, where $\mathrm{C}^{(z)}$ is an $n \times n$ matrix such that $\mathrm{C}^{(z)}[x,y]=1$ if and only if z is a CA of $\{x,y\}$. Obviously, $\mathrm{C}^{(z)}$ can be derived from A_{clo} in time $O(n^2)$.

Let $\mathrm{A}^{(z)}$ be a Boolean $n \times n$ matrix such that $\mathrm{A}^{(z)}[x,z']=1$ if and only if x is reachable from z' and z' is a child of z. Observe that $\mathrm{A}^{(z)}$ is similar to the rectangular matrix $A^T_{\mathrm{clo}}[*,N^{\mathrm{out}}(z)]$. The only difference is that columns not corresponding to children of z are not completely deleted from A^T_{clo}, but are replaced by all zero columns.

Let $\mathrm{W}^{(z)} = \mathrm{A}^{(z)} \cdot A_{\mathrm{clo}}$ be the *witness matrix* of z.

Claim 4.11. *Let $L^{(z)}$ be the solution matrix of* ALL-PAIRS FIXED-VERTEX LCA *on G and z. Then, $L^{(z)} = C^{(z)} \wedge \neg W^{(z)}$.*

Proof. Let $x,y \in V$ and assume $z \in \mathrm{LCA}\{x,y\}$. Clearly, $\mathrm{C}^{(z)}[x,y]=1$ by definition. On the other hand there exists no witness. Suppose that $\mathrm{A}^{(z)}[x,z'] \cdot A_{\mathrm{clo}}[z',y]=1$ for some z'. Then, by construction, z' is a child of z and z' reaches both x and y contradicting our assumption. That is, $\mathrm{A}^{(z)}[x,z] \cdot A_{\mathrm{clo}}[z,y]=0$ for all z and hence $\mathrm{W}^{(z)}[x,y]=0$. The other direction is analogous. ■

$\mathrm{W}^{(z)}$ can be computed by one matrix multiplication. $\mathrm{L}^{(z)}$ can be derived from $\mathrm{C}^{(z)}$ and $\mathrm{W}^{(z)}$ in time $O(n^2)$. Observe that the reduction from ALL-PAIRS FIXED-VERTEX LCA to BOOLEAN MATRIX MULTIPLICATION takes time $O(n^\omega)$ and is therefore valid in the sense of our reduction concept, see Section 2.2. ■

The matrix $\mathrm{A}^{(z)}$ used in the proof of Lemma 4.10 to compute the witness matrix W is structurally very similar to the rectangular matrix $A^T_{\mathrm{clo}}[*,N^{\mathrm{out}}(z)]$. Moreover, it is easy to see that W corresponds to a rectangular matrix product.

Observation 4.12. *Let $W^{(z)}$ be the witness matrix of z. Then,*

$$W^{(z)} = A^T_{\mathrm{clo}}[*,N^{out}(z)] \cdot A_{\mathrm{clo}}[N^{out}(z),*].$$

In particular, this implies that the witness matrix $\mathrm{W}^{(z)}$ can be computed significantly faster whenever the number of out-neighbors of z is small. However, it is necessary to compute the transitive closure

of G, i.e., A_{clo}, in order to derive the rectangular matrices. Therefore, we cannot bound the time needed to solve ALL-PAIRS FIXED-VERTEX LCA by $O\left(n^{\omega(1,\log_n(|N^{\text{out}}(z)|),1)}\right)$. On the other hand, Observation 4.12 is useful for some of the algorithmic solutions developed below.

Lemma 4.13. BOOLEAN MATRIX MULTIPLICATION *of two $n \times n$ matrices can be reduced to* ALL-PAIRS FIXED-VERTEX LCA *on a dag $G = (V,E)$ with $O(n)$ vertices in time $O(n^2)$.*

Proof. We show that matrix multiplication can be solved by solving ALL-PAIRS FIXED-VERTEX LCA.

Let A and B be two Boolean $n \times n$ matrices and let $C = A \cdot B$. We construct a dag $G = (V,E)$ from A and B as follows.

- $V = \{z\} \cup W \cup X \cup Y$ where z is a single vertex and $W = \{w_1, \ldots, w_n\}$, $X = \{x_1, \ldots, x_n\}$, and $Y = \{y_1, \ldots, y_n\}$ are vertex sets of cardinality n.
- $E = E_1 \cup E_2 \cup E_3$ where
  - $E_1 = (z, v)$ for all vertices $v \in W \cup X \cup Y$.
  - $E_2 = (w_k, x_i)$ for all $1 \leq i,k \leq n$ such that $A[i,k] = 1$.
  - $E_3 = (w_k, y_j)$ for all $1 \leq k, j \leq n$ such that $B[k,j] = 1$.

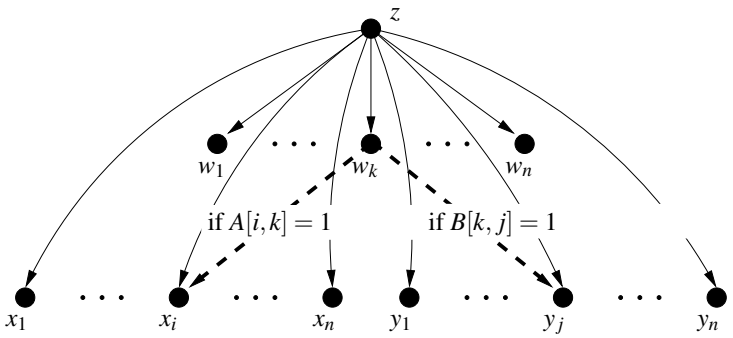


Figure 4.4: Construction of G from A and B. Observe that $z \notin \text{LCA}\{x_i, y_j\}$ if and only if there exists an index k such that $w_k \in \text{CA}\{x,y\}$. This, in turn implies $C[i,j] = 1$ by construction.

Observe that the construction of G from A and B takes time $O(n^2)$. The following observations are immediate.

1. $z \in \text{CA}\{x_i, y_j\}$ for all $1 \leq i,j \leq n$.
2. $w_k \in \text{CA}\{x_i, y_j\}$ if and only if $A[i,k] = B[k,j] = 1$, i.e., $A[i,k] \cdot B[k,j] = 1$.

From this, we can conclude

$$\begin{aligned} C[i,j] = 1 &\iff \exists k \in \{1,\ldots,n\} \text{ such that } A[i,k] \cdot B[k,j] = 1 \\ &\iff \text{there is a witness for } z \notin \text{LCA}\{x_i, y_j\} \\ &\iff z \notin \text{LCA}\{x_i, y_j\}. \end{aligned}$$

Hence, we can derive a solution to the matrix multiplication from a solution to ALL-PAIRS FIXED-VERTEX LCA in time $O\left(n^2\right)$ which concludes the proof.

■

Lemmas 4.10 and 4.13 establish the computational equivalence between BOOLEAN MATRIX MULTIPLICATION and ALL-PAIRS FIXED-VERTEX LCA.

Fixed-Vertex-Pairs All LCA

We proceed by establishing the equivalence between BOOLEAN MATRIX MULTIPLICATION and FIXED-VERTEX-PAIRS ALL LCA in a similar manner. Recall that the FIXED-VERTEX-PAIRS ALL LCA problem is to compute (the set) $\mathrm{LCA}\{x,y\}$ for a fixed vertex x and all $y \in V$.

Lemma 4.14. FIXED-VERTEX-PAIRS ALL LCA *on a dag $G=(V,E)$ with n vertices can be reduced to* BOOLEAN MATRIX MULTIPLICATION *of two $n \times n$ matrices in time $O(n^\omega)$.*

Proof. Let in the following $G=(V,E)$ be a dag and fix a vertex $x \in V$. Let $N^{\mathrm{in}}_{\mathrm{clo}}(x) \subseteq V$ denote the set of ancestors of x. We proceed similarly as in the proof of Lemma 4.10. Let $\mathrm{C}^{(x)}$ be a common ancestor matrix such that $\mathrm{C}^{(x)}[z,y]=1$ if and only if $z \in \mathrm{CA}\{x,y\}$. Observe that $\mathrm{C}^{(x)}$ can be derived from A_{clo} in time $O(n^2)$.

Let $z,y \in V$ such that $z \in N^{\mathrm{in}}_{\mathrm{clo}}(x)$ and z is a predecessor of y. Observe that this implies $z \in \mathrm{CA}\{x,y\}$. Moreover, the following is true for x,y,z.

Claim 4.15. *$z \in \mathrm{LCA}\{x,y\}$ if and only if there is no path (z,w,y) in G_{clo} such that $w \in N^{in}_{clo}(x)$.*

To see this, observe that w corresponds to a witness for $\{x,y\}$ and z. Similarly to the proof of Lemma 4.10, we compute a witness matrix $\mathrm{W}^{(x)}$ by exploiting the above claim. Let $\mathrm{A}^{(x)}$ be a Boolean $n \times n$ matrix such that $\mathrm{A}^{(x)}[z,z']=1$ if and only if $z,z' \in N^{\mathrm{in}}_{\mathrm{clo}}(x)$ and $(z,z') \in E_{\mathrm{clo}}$. Let $\mathrm{W}^{(x)} = \mathrm{A}^{(x)} \cdot A_{\mathrm{clo}}$. Observe that $\mathrm{W}^{(x)}[z,y]=1$ if and only if there exists a path (z,w,y) such that $z,w \in N^{\mathrm{in}}_{\mathrm{clo}}(x)$ in G_{clo}.

Claim 4.16. *Let L be the solution matrix of* FIXED-VERTEX-PAIRS ALL LCA *for G and x. Then, $L = C^{(x)} \wedge \neg W^{(x)}$.*

The matrix $\mathrm{A}^{(x)}$ can be derived from A_{clo} in $O(n^2)$ as well as L from $\mathrm{C}^{(x)}$ and $\mathrm{W}^{(x)}$. Hence, FIXED-VERTEX-PAIRS ALL LCA can be reduced to BOOLEAN MATRIX MULTIPLICATION in time $O(n^\omega)$. ■

Now consider again the square matrix product $\mathrm{W}^{(x)} = \mathrm{A}^{(x)} \cdot A_{\mathrm{clo}}$ used in the proof of Lemma 4.14. Recall that $\mathrm{W}^{(x)}[z,y]=1$ if and only if there exists a witness for z and $\{x,y\}$. Observe that $\mathrm{W}^{(x)}[z,y]=0$ for $z \notin N^{\mathrm{in}}_{\mathrm{clo}}(x)$.

Observation 4.17. *Let $W^{(z)}$ be the witness matrix of z. Then,*

$$W^{(x)}[N^{in}_{clo}(x),*] = A_{\mathrm{clo}}[N^{in}_{clo}(x),N^{in}_{clo}(x)] \cdot A_{\mathrm{clo}}[N^{in}_{clo}(x),*].$$

This observation implies that the witness matrix computation can be reduced to a rectangular matrix problem, i.e., multiplying an $|N^{\mathrm{in}}_{\mathrm{clo}}(x)| \times |N^{\mathrm{in}}_{\mathrm{clo}}(x)|$ matrix by an $|N^{\mathrm{in}}_{\mathrm{clo}}(x)| \times n$ matrix. The complexity of this approach depends on the number of ancestors of x. Again, since the reduction depends on computing the transitive closure in $O(n^\omega)$, this does not result in an improved upper bound for FIXED-VERTEX-PAIRS ALL LCA but is useful in the context of ALL-PAIRS ALL LCA.

Lemma 4.18. BOOLEAN MATRIX MULTIPLICATION *of two $n \times n$ matrices can be reduced to* FIXED-VERTEX-PAIRS ALL LCA *on a dag $G = (V,E)$ with $O(n)$ vertices in time $O(n^2)$.*

Proof. Let A and B be two arbitrary Boolean $n \times n$ matrices and let $C = A \cdot B$. We construct a dag $G = (V,E)$ from A and B as follows.

- $V = \{x\} \cup Z \cup W \cup Y$ where x is a single vertex and $Z = \{z_1, \ldots, z_n\}$, $W = \{w_1, \ldots, w_n\}$, and $Y = \{y_1, \ldots, y_n\}$ are vertex sets of cardinality n.
- $E = E_1 \cup E_2 \cup E_3 \cup E_4$ where
  - $E_1 = (v,x)$ for all vertices $v \in Z \cup W$.
  - $E_2 = (z,y)$ for all vertices $z \in Z$ and $y \in Y$.
  - $E_3 = (z_i, w_k)$ for all $1 \leq i,k \leq n$ such that $A[i,k] = 1$.
  - $E_4 = (w_k, y_j)$ for all $1 \leq k, j \leq n$ such that $B[k,j] = 1$.

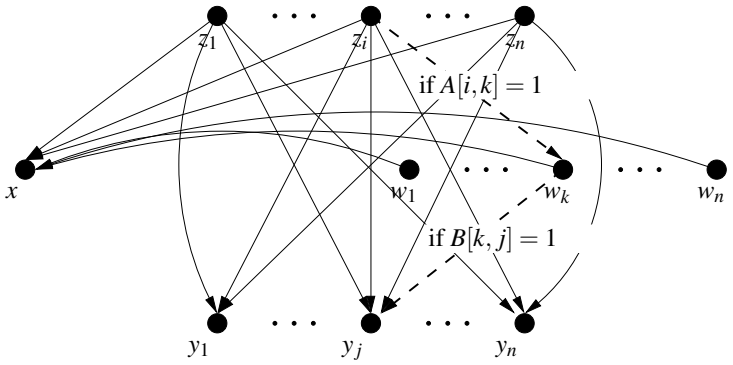


Figure 4.5: Construction of G from A and B. Observe that $z_i \notin \mathrm{LCA}\{x, y_j\}$ if and only if there exists an index k such that $w_k \in \mathrm{CA}\{x, y_j\}$. This, in turn implies $C[i,j] = 1$ by construction.

Observe that G has $3n+1$ vertices and that G can be constructed from A and B in time $O\left(n^2\right)$. We observe:

1. $z_i \in \mathrm{CA}(x, y_j)$ for all $1 \leq i, j \leq n$.
2. $w_k \in \mathrm{CA}(x, y_j)$ if and only if $B[k,j] = 1$.
3. w_k is a witness for z_i and (x, y_j) if and only if $w_k \in \mathrm{CA}(x, y_j)$ and $A[i,k] = 1$, that is, $A[i,k] \cdot B[k,j] = 1$.

Now we conclude as follows.

$$\begin{aligned} C[i,j] = 1 &\iff \exists k \in \{1, \ldots, n\} \text{ such that } A[i,k] \cdot B[k,j] = 1 \\ &\iff \text{there is a witness for } z_i \notin \mathrm{LCA}(x, y_j) \\ &\iff z_i \notin \mathrm{LCA}(x, y_j) \end{aligned}$$

■

Hence, by Lemmas 4.14 and 4.18, BOOLEAN MATRIX MULTIPLICATION and FIXED-VERTEX-PAIRS ALL LCA are computationally equivalent. This, in turn, combined with Lemmas 4.10 and 4.18, establishes Theorem 4.9.

4.4.2 All-Pairs All LCA

Theorem 4.9 implies that ALL-PAIRS FIXED-VERTEX LCA and FIXED-VERTEX-PAIRS ALL LCA can be solved in time $O(n^\omega)$. Furthermore, this immediately leads to an $O\left(n^{1+\omega}\right)$ algorithm for ALL-PAIRS ALL LCA – simply solve one of the two fixed-vertex LCA variants for all vertices $v \in V$. However, the application of fast rectangular matrix multiplication yields even stronger upper time bounds.

General Upper Bound

In the following we establish an upper time bound of $O\left(n^{\omega(2,1,1)}\right)$ for ALL-PAIRS ALL LCA on general dags. We use the decomposition into n ALL-PAIRS FIXED-VERTEX LCA subproblems. Observe, however, that the same bound can be obtained by using a decomposition with respect to FIXED-VERTEX-PAIRS ALL LCA.

Let in the following $\mathrm{W}^{(v)}$, $\mathrm{C}^{(v)}$, and $\mathrm{A}^{(v)}$ denote the witness matrix, the common ancestor matrix, and the children matrix of v with respect to the ALL-PAIRS FIXED-VERTEX LCA problem, i.e., as described in the proof of Lemma 4.10. As elaborated in the proof, $\mathrm{W}^{(v)}$ is computed in time $O(n^\omega)$ by multiplying $\mathrm{A}^{(v)}$ and A_{clo}. For n instances of ALL-PAIRS FIXED-VERTEX LCA, n witness matrices are constructed in time $O\left(n^{1+\omega}\right)$ by this approach. The total time needed for the construction of the witness matrices can be reduced by applying fast rectangular matrix multiplication as follows. Compute the witness matrices $\mathrm{W}^{(v)}$ for all vertices $v \in V$ in one step by multiplying an $n^2 \times n$ and an $n \times n$ matrix in time $O\left(n^{\omega(2,1,1)}\right)$. Let $V = \{v_1, \ldots, v_n\}$.

$$\begin{pmatrix} \mathrm{A}^{(v_1)} \\ \mathrm{A}^{(v_2)} \\ \vdots \\ \mathrm{A}^{(v_n)} \end{pmatrix} \cdot A_{\text{clo}} \quad = \quad \begin{pmatrix} \mathrm{W}^{(v_1)} \\ \mathrm{W}^{(v_2)} \\ \vdots \\ \mathrm{W}^{(v_n)} \end{pmatrix} \tag{4.5}$$

Theorem 4.19. ALL-PAIRS ALL LCA *can be solved in time* $O\left(n^{\omega(2,1,1)}\right)$ *on a dag* G *with* n *vertices.*

Proof. The full algorithm works as follows:

1. Compute the transitive closure G_{clo} of G.
2. Compute the common ancestor matrices $\mathrm{C}^{(v)}$ for all $v \in V$. Since G_{clo} is known, this can be done in time $O\left(n^3\right)$.
3. Compute the witness matrices $\mathrm{W}^{(v)}$ for all $v \in V$ using Equation (4.5). This takes total time of $O\left(n^{\omega(2,1,1)}\right)$.
4. Let $\mathrm{L}^{(v)} = \mathrm{C}^{(v)} \wedge \neg \mathrm{W}^{(v)}$ for each $v \in V$. Then, for each pair $\{x, y\}$, all LCAs can be read from the entries $\mathrm{L}^{(v)}[x, y]$ for each v. That is, $z \in \mathrm{LCA}\{x, y\}$ if and only if $\mathrm{L}^{(z)}[x, y] = 1$. This step takes a total of $O\left(n^3\right)$ time.

■

Hence, ALL-PAIRS ALL LCA reduces to the multiplication of a Boolean $n^2 \times n$ matrix by a Boolean $n \times n$ matrix. Currently, upper bounds for rectangular matrix multiplication imply $\omega(2,1,1) <$

3.334, see Proposition 4.5. Extending the idea of the proof of Lemma 4.13 does not directly yield an equivalence between the two problems. More specifically, we can only reduce rectangular matrix multiplication instances with a certain property to ALL-PAIRS ALL LCA on a dag with $O(n)$ vertices. The general problem on the other hand can be reduced to the problem of computing all LCAs of $\Theta(n^2)$ pairs in a dag with $\Theta(n^2)$ vertices.

Corollary 4.20. *Let A be a Boolean $n^2 \times n$ matrix and let B be a Boolean $n \times n$ matrix. Then, computing $C = A \cdot B$ can be reduced to* ALL-PAIRS ALL LCA *if*

(i)

$$A = \begin{pmatrix} A^{(1)} \\ A^{(2)} \\ \vdots \\ A^{(n)} \end{pmatrix}$$

such that $A^{(1)}, \ldots, A^{(n)}$ are Boolean $n \times n$ matrices.

(ii) There exists a Boolean $n \times n$ matrix D such that for all $1 \leq i,k \leq n$ either $A^{(i)}[*,k] = D^{(i)}[*,k]$ or $A^{(i)}[*,k] = 0$. Recall that $A^{(i)}[*,k]$ corresponds to the kth column of matrix $A^{(i)}$.

Sparse Transitive Reduction

The upper bound given in Theorem 4.19 applies to general dags. For sparse dags, more specifically for dags such that $m_{\text{red}} = O\left(n^{1.92}\right)$, we can improve the bound by exploiting Observation 4.12.

Theorem 4.21. ALL-PAIRS ALL LCA *on a dag G with n vertices can be solved in time $O\left(n^{\omega(1,r,1)+1}\right)$, where $r = \log_n\left(m_{\text{red}}/n\right)$ and m_{red} is the number of edges in the transitive reduction of G.*

Proof. The cost for computing the witness matrices $\mathrm{W}^{(v)}$ by using independent rectangular matrix multiplication is given by

$$\sum_{v \in V} O\left(n^{\omega\left(1,\log_n\left(|N^{\text{out}}_{\text{red}}(v)|\right),1\right)}\right).$$

We need Jensen's inequality to bound the above sum.

Proposition 4.22 (Jensen's Inequality)**.** *Let ϕ be real convex function. Then,*

$$\phi\left(\frac{\sum_{i=1}^{n} x_i}{n}\right) \leq \frac{\sum_{i=1}^{n} \phi(x_i)}{n}.$$

The inequality is reversed if ϕ is concave.

We observe that the function $f(x) = n^{\omega(1,x,1)-2}$ is concave for $0 \leq x \leq 1$. Hence, we can conclude that $n^{\omega\left(1,\log_n\left(|N^{\text{out}}_{\text{red}}(v)|\right),1\right)-2}$ is concave. Thus, we obtain:

$$\begin{aligned}
\sum_{v\in V} n^{\omega\left(1,\log_n\left(|N^{\text{out}}_{\text{red}}(v)|\right),1\right)} &= n^2 \sum_{v\in V} n^{\omega\left(1,\log_n\left(|N^{\text{out}}_{\text{red}}(v)|\right),1\right)-2} \\
&= n^2\cdot n\cdot \sum_{v\in V} \frac{n^{\omega\left(1,\log_n\left(|N^{\text{out}}_{\text{red}}(v)|\right),1\right)-2}}{n} \\
&\leq n^2\cdot n\cdot n^{\omega\left(1,\log_n\left(\frac{\sum_{v\in V}|N^{\text{out}}_{\text{red}}(v)|}{n}\right),1\right)-2} \\
&= n^2\cdot n\cdot n^{\omega(1,\log_n(m_{\text{red}}/n),1)-2} \\
&= n\cdot n^{\omega(1,\log_n(m_{\text{red}}/n),1)}
\end{aligned}$$

Hence, we have proved that the witness matrices can be computed in $O\left(n^{1+\omega(1,\log_n(m_{\text{red}}/n),1)}\right)$. Now recall the algorithm given in the proof of Theorem 4.19. Steps 1,2, and 4 can obviously be bounded by $O(n^3)$ which concludes the proof. ■

As a consequence of Theorem 4.21, the general upper time bound given in Theorem 4.19 can be tightened for sparse dags.

Corollary 4.23. *Let G be a dag with n vertices such that m_{red} is the number of edges in the transitive reduction of G.*

(i) If $m_{\text{red}} = O\left(n^{1.294}\right)$, then ALL-PAIRS ALL LCA *can be solved in $O\left(n^3\right)$.*

(ii) If $m_{\text{red}} = O\left(n^{1.92}\right)$, then, there exists a constant $\delta > 0$ such that ALL-PAIRS ALL LCA *on G can be solved in time $O\left(n^{3.334-\delta}\right)$.*

Proof. By Proposition 4.4:

$$\omega(1,r,1) \leq \begin{cases} 2+o(1) & \text{if } 0\leq r\leq \alpha \\ 2+\frac{\omega-2}{1-\alpha}(r-\alpha)+o(1) & \text{if } \alpha\leq r\leq 1. \end{cases}$$

Let now $m_{\text{red}}/n = n^r$. Obviously, for $r \leq 0.294$ we have $n^{\omega(1,r,1)+1} = O\left(n^3\right)$. Moreover, for $r > 0.294$ we have:

$$\begin{aligned}
\omega(1,r,1)+1 &\leq 3.334-\delta \\
r &\leq \left(0.334+\frac{(\omega-2)\alpha}{1-\alpha}\right)\frac{1-\alpha}{\omega-2}-\delta
\end{aligned}$$

Plugging in the current value for ω and α, i.e., 2.376 and 0.294 respectively, we find $r \leq 0.9208-\delta$ for a constant $\delta > 0$. ■

Sparse Transitive Closure

We proceed by considering ALL-PAIRS ALL LCA on dags with a (moderately) small transitive closure. More specifically, we show that if the average number of ancestors of a vertex v in a dag G is bounded by $O\left(n^{0.96}\right)$, we can improve the general upper bound of Theorem 4.19. This result is achieved by decomposing ALL-PAIRS ALL LCA into n FIXED-VERTEX-PAIRS ALL LCA subproblems and using fast rectangular matrix multiplication to compute the corresponding witness matrices.

Theorem 4.24. ALL-PAIRS ALL LCA *on a dag G with n vertices can be solved in time $O\left(n^{\omega(s,s,1)+1}\right)$, where $s = \log_n(m_{\text{clo}}/n)$ and m_{clo} is the number of edges in the transitive closure of G.*

Proof. Obviously, ALL-PAIRS ALL LCA can be solved by solving FIXED-VERTEX-PAIRS ALL LCA for each vertex $v \in V$. By Observation 4.17, the witness matrix for a fixed vertex v can be computed by rectangular matrix multiplication in time $O\left(n^{\omega\left(\log_n\left(|N^{\text{in}}_{\text{clo}}(v)|\right),\log_n\left(|N^{\text{in}}_{\text{clo}}(v)|\right),1\right)}\right)$. Hence, we can be bound the total time needed to compute the witness matrices by

$$\sum_{v\in V} O\left(n^{\omega\left(\log_n\left(|N^{\text{in}}_{\text{clo}}(v)|\right),\log_n\left(|N^{\text{in}}_{\text{clo}}(v)|\right),1\right)}\right).$$

The above sum can be bounded by Jensen's inequality in the same way as in the proof of Theorem 4.21. Hence, we get the following upper bound for ALL-PAIRS ALL LCA:

$$O\left(n^{\omega(\log_n(m_{\text{clo}}/n),\log_n(m_{\text{clo}}/n),1)+1}\right)$$

■

Corollary 4.25. *Let G be a dag with n vertices and let m_{clo} be the number of edges in the transitive closure of G.*

(i) If $m_{\text{clo}} = O\left(n^{1.72}\right)$, then ALL-PAIRS ALL LCA *on G can be solved in time $O(n^3)$.*

(ii) If $m_{\text{clo}} = O\left(n^{1.96}\right)$, then, there exists a constant $\delta > 0$ such that ALL-PAIRS ALL LCA *on G can be solved in time $O\left(n^{3.334-\delta}\right)$.*

Proof. Let $0 < r \leq 1$ such that $r = \log_n(m_{\text{clo}}/n)$. By using square matrix multiplication we have

$$\omega(r,r,1) = 1 - r + r\omega.$$

By Theorem 4.24 the total time needed can thus be bounded by

$$O\left(n^{2-r+r\omega}\right).$$

Now simply solve the following inequality

$$2 - r + r\omega < \lambda$$

for $\lambda = 3$ and $\lambda = 3.334$ to get $r < 0.726$ and $r < 0.969$ respectively. ■

Remark 4.26. *The upper bounds for the last corollary can be improved by using the slightly tighter upper bound for the rectangular matrix multiplication exponent given in the paper of Huang and Pan [HP98]. For example, plugging in $q = 7$ and $\beta = 0.0336$ in the right-hand side of (7.1) yields $r < 0.7329$.*

4.5 Conclusion

Similar as in the case of many thoroughly studied graph problems like ALL-PAIRS SHORTEST DISTANCES, computing the transitive closure, or finding maximum matchings, matrix multiplication turns out to be an indispensable ingredient in the design of efficient solutions to all-pairs LCA problems. The up to this point best upper time bounds for both ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA presented in this chapter rely crucially on fast (rectangular) matrix multiplication. This in turn implies that any improvement on the constants ω and/or α gives a more accurate picture of the actual numerical exponents, which we now bound by 2.575 for ALL-PAIRS REPRESENTATIVE LCA and 3.334 for ALL-PAIRS ALL LCA. Although the results presented in this chapter widen the understanding of the underlying problems significantly, intriguing open questions persist. A first line of future research concerns the ultimate classification of the problems within the hierarchy of well studied related graph problems. Is it coincidental that the current exponents of ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS SHORTEST DISTANCES in unweighted dags correspond to each other? Can we eventually show a computational equivalence? Similarly, can we show that ALL-PAIRS ALL LCA and the multiplication of an $n^2 \times n$ by an $n \times n$ matrix is equivalent? Closely connected to questions of this kind are research efforts that aim at virtually improving the exponents of ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA, i.e., improvements that are not derived from improvements of the matrix multiplication constants. It seems, however, that purely matrix-multiplication-based methods are exploited. A more promising step into this direction is given in Chapter 5, where matrix multiplication is combined with techniques that use other combinatorial properties.

CHAPTER

5

PATH COVER TECHNIQUES

5.1 Introduction

Unless the exponents of matrix multiplication are improved, algorithms of the kind presented in the previous chapter can hardly be expected to provide significant improvement for the considered LCA problems. The approach taken in this chapter is fundamentally different. We apply a path cover technique to decompose the problem of finding LCAs in the input dag to several LCA problems in trees, where the trees are derived from the respective path cover. It turns out that this approach is efficient on dags with low width. In Czumaj et al. [CKL07], an algorithm for computing ALL-PAIRS REPRESENTATIVE LCA that exploits low depth of the input dag was presented. Although this algorithm uses matrix multiplication, the basic observation, namely that a maximum depth CA is a lowest common ancestor, dissolves the necessity for computing maximum CAs, which is common to the purely matrix-multiplication-based algorithms. In some sense, the approach taken in this chapter is dual to the algorithm in [CKL07] and both can be combined in a natural way. In particular, the results presented in this chapter can be summarized as follows:

1. We elaborate in-depth on using path cover techniques for the solution to LCA problems in dags. To this end, we present a versatile decomposition technique that can be applied to a variety of LCA problems.
2. We apply our technique to the ALL-PAIRS REPRESENTATIVE LCA problem improving recently developed solutions for dags of small width [KL07]. Our result implies an upper time bound of $O(n^2 \mathrm{w}(G) \log n)$ and improves the result in [KL07] for dags with width $\mathrm{w}(G)$ bounded by $O(n^{\omega-2-\delta})$ for a constant $\delta > 0$. Similarly, the application of our approach to the ALL-PAIRS ALL LCA problem yields an upper time bound of $O(n^2 \mathrm{w}(G)^2 \log^2 n)$. This improves the general upper bound of $O(n^{\omega(2,1,1)})$ (Thm. 4.19) for $\mathrm{w}(G) = O(n^{\frac{\omega(2,1,1)-2}{2}-\delta})$. This also improves the upper time bound of $O(n\, m_{\mathrm{red}} \min\{\left(\mathrm{w}(G)^2 + \mathrm{w}(G)\log n\right), n\})$, which can be derived from Theorem 3.15, whenever the size of the transitive reduction of G cannot be bounded by

$\widetilde{O}(n)$.

3. We show that it is possible to combine the path cover approach with the efficient method for low depth dags given in [CKL07]. Our algorithm has the same asymptotic worst case time complexity as the currently fastest algorithm for this problem, i.e., $O(n^{2+\mu})$, see Theorem 4.8, up to polylogarithmic factors. However, the class of dags for which the algorithm needs $\Omega(n^{2+\mu})$ time is considerably limited (Thm. 5.22).
4. Finally, we describe an application of our approach. We improve the general upper bound for the ALL-K-SUBSETS REPRESENTATIVE LCA problem (i.e., compute representative LCAs for each vertex subset of size k) for $k = 3$, which was recently established by Yuster [Yus]. Another application of the path cover approach is within the realm of space-efficient LCA algorithms, however, the description of this application is deferred to Chapter 6.

We start by reviewing work related to the construction of path covers of minimum size (Section 5.2). Next, in Section 5.3, we describe an algorithm for ALL-PAIRS ALL LCA that can be considered to be the historical precursor of path-cover-based LCA algorithms. We note that the very same algorithm also was the first algorithm with general upper bound of $o(n^4)$ for ALL-PAIRS ALL LCA.

In Section 5.4, we describe the more generalized version of the path cover technique and derive its main properties. This technique is then combined with the method for dags of low width in Section 5.5. Finally, we conclude by applying the combined algorithm to improve the upper time bound for ALL-K-SUBSETS REPRESENTATIVE LCA for $k = 3$.

5.2 Related Work

Recall that the width $\mathrm{w}(G)$ of a dag G corresponds to the cardinality of a maximum antichain, or to the cardinality of a minimum path (chain) cover of G (G_{clo}). Recall also that path covers and chain covers are in one-to-one correspondence, see Section 2.1 for more details.

Let in the following $G = (V,E)$ be a dag with vertex set $V = \{v_1, \ldots, v_n\}$, m edges, and transitive closure $G_{\mathrm{clo}} = (V, E_{\mathrm{clo}})$. The path cover technique described in this chapter relies crucially on efficient methods for constructing small path covers for a given dag G. In the following we review algorithmic approaches for constructing minimum chain covers for transitive closures of dags; however, these algorithms can also be used for constructing path covers in G by the following observation.

Observation 5.1. *Given a chain cover for G_{clo} with cardinality r, a path cover for G with cardinality r can be constructed in time $O(rm)$.*

Proof. We observe that a given chain $c = (v_1, \ldots, v_k)$ in G_{clo} can be expanded to a path in G by performing a depth first search in G in time $O(m)$. To this end, the corresponding stack is initialized with the vertices $v_1, \ldots, v_k$ (in this order); hence the start vertex of the DFS is v_k. It is not difficult to see that the resulting DFS tree contains a path from v_1 to v_k. ■

A minimum chain cover for G_{clo} is usually constructed by using a reduction to bipartite matching given by Ford and Fulkerson [FF62]. We briefly sketch this idea below. Compute the transitive closure G_{clo} of G and construct a bipartite (undirected) graph $G' = (X \uplus Y, E')$ where X and Y are copies of V and $E' = \{\{x_i, y_j\} \mid x_i \in X,\, y_j \in Y \text{ and } (v_i, v_j) \in G_{\mathrm{clo}}\}$. Let M be a maximum bipartite matching in G'. The edges in M, interpreted as directed edges, define a chain cover of G_{clo}. More specifically, the

graph G^M_{clo} that is induced by the edges of M corresponds to a chain cover of G_{clo}. It is not difficult to see that the number of chains is equal to the number of unmatched vertices in X (Y). On the other hand, any chain cover of G_{clo} induces a matching of G'; hence, the chain cover induced by the maximum matching in G' is of minimum cardinality. Given a matching M, the chains can be constructed using, e.g., a breadth first search of G_{clo} restricted to the edges of M.

A maximum bipartite matching M in G' can be found in time $O(m'\sqrt{n}) = O(n^{2.5})$ applying the algorithm by Hopcroft and Karp [HK73][1]. Observe that m' corresponds to $|E'| = |E_{\text{clo}}|$, i.e., the number of comparable vertex pairs in G. Recently, Mucha and Sankowski [MS04] gave a randomized algorithm for the maximum bipartite matching problem with running time $O(n^\omega)$. Benczúr et al. [BFK99] modify the original reduction described above to obtain algorithms with running times $O(nm)$ and $O(m\sqrt{n\log n})$ respectively. Observe that this improves significantly on the $O(m_{\text{clo}}\sqrt{n})$ upper time bound whenever $m \ll m_{\text{clo}}$. Closer inspection of the algorithms and Proposition 3.2 show that both bounds can even be improved to $O(n\,m_{\text{red}})$ and $O(m_{\text{red}}\sqrt{n\log n})$ respectively. We summarize the results in the following proposition.

Proposition 5.2. *Given a dag G with n vertices, a minimum chain cover for G_{clo} can be constructed in time $O(\min\{n^{2.5}, n\,m_{\text{red}}\})$.*

5.3 A First Non-Trivial All-Pairs All LCA Solution

Before we embark upon the description of a general, multi-purpose decomposition technique, we study an ALL-PAIRS ALL LCA algorithm that reduces ALL-PAIRS ALL LCA to several ALL-PAIRS REPRESENTATIVE LCA computations. Historically, this approach yielded the first ALL-PAIRS ALL LCA algorithm [BEG+07] with running time $o(n^4)$, but was later improved in [EMN07], see Chapter 4. Below, we describe the underlying idea.

Suppose we are given a vertex z and want to determine all pairs $\{x,y\}$ for which z is an LCA, i.e., solve ALL-PAIRS FIXED-VERTEX LCA for z. (In fact, the basic idea of the reduction is also used in Section 4.4.)

To this end, we employ an ALL-PAIRS REPRESENTATIVE LCA algorithm on G instead of using Lemma 4.10. Hence, the rough idea is to manipulate the ALL-PAIRS REPRESENTATIVE LCA algorithm such that it returns a fixed vertex z as a representative LCA for $\{x,y\}$ whenever $z \in \text{LCA}\{x,y\}$. Since most ALL-PAIRS REPRESENTATIVE LCA algorithms return maximum CAs, i.e., make use of Proposition 2.10, this can be achieved by maximizing the topological number of z.

For a dag $G = (V,E)$ and a vertex $z \in V$, let $top^m(z)$ denote the maximum number of z in any topological order of G. It is easily seen that a topological order top satisfies $top(z) = top^m(z)$ if and only if $top(z) \leq top(x)$ implies $z \rightsquigarrow x$ for all $x \in V$. This immediately leads to a linear time algorithm for finding a corresponding order.

Lemma 5.3. *A topological order realizing $top^m(z)$ for any vertex z in a dag G with n vertices and m edges 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

[1] This bound was mildly improved to $O(n^{2.5}/\sqrt{\log n})$ in [ABMP91].

the dag induced by the removed vertices arbitrarily (via DFS), and concatenate both vertex lists each of which is sorted in ascending order with respect to the topological numbers. ■

If we fix a vertex z's number maximizing topological order, then z is the maximum CA of all vertex pairs $\{x,y\}$ such that $z \in \mathrm{LCA}\{x,y\}$. Now clearly, our strategy is to iterate for each $v \in V$ over the orders that maximize $top^m(v)$. Note that the $o(n^3)$ algorithms in [BFCP+05, KL05, CKL07] as well as the algorithms described in Sections 3 and 4 naturally return the vertex z with the highest number $top(z)$ (for a fixed topological order) among all LCAs of any pair $\{x,y\}$. This leads to Algorithm 6.

Algorithm 6: All LCA Using Representative LCAs

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.

```
1 begin
2     foreach z ∈ V do
3         Compute a topological ordering top such that top(z) is maximal.
4         Solve ALL-PAIRS REPRESENTATIVE LCA using any algorithm that returns the maximum CAs
          with respect to top and get matrix R.
5         foreach {x,y} with R[x,y] = z do A[x,y] ← A[x,y] ∪ {z} (by multiset-union)
6     end
7     Remove elements of multiplicity greater than one from A[x,y] for all x,y ∈ V.
8 end
```

The runtime of the algorithm is clearly $O(n^{3+\mu})$ for general dags, which is inferior to the $O(n^{\omega(2,1,1)})$ bound given in Theorem 4.19. However, a key observation is that an algorithm that outputs maximum CAs does it for all vertices $v \in V$ with $top(v) = top^m(v)$ in parallel. Hence, we aim at maximizing topological numbers simultaneously for as many vertices as possible. This can easily be achieved for vertices on chains.

Lemma 5.4. *A topological ordering maximizing $top^m(z)$ for all vertices z on a chain c in the transitive closure G_{clo} of a dag G simultaneously can be computed in time $O(n+m)$.*

Proof. For a chain $c = (v_1, \ldots, v_k)$, iteratively use the algorithm described in the proof of Lemma 5.3 starting at v_k and going to vertex v_1. ■

This lemma implies that, given such an ordering, it is possible to process a chain c in only one iteration of the algorithm, i.e., only one call of an ALL-PAIRS REPRESENTATIVE LCA algorithm. Thus, we can reduce the running time if we minimize the number of chains to be processed. This leads to the improved Algorithm 7.

The time bound for the improved approach follows immediately. Recall that a minimum chain cover can be computed in time $O(\min\{n^{2.5}, n m_{\mathrm{red}}\})$, see Proposition 5.2. The additional term of $n^2 \mathrm{w}(G) \log \mathrm{w}(G)$ is needed for the removal of non-unique elements. This can be done by sorting and scanning the multisets in time $O(n^2 \mathrm{w}(G) \log \mathrm{w}(G))$. Observe that bucket sort could be used in the case that $\mathrm{w}(G) \log \mathrm{w}(G) > n$.

Theorem 5.5. *Algorithm 7 solves* ALL-PAIRS ALL LCA *on a dag G with n vertices and width $\mathrm{w}(G)$ in $O\left(n^2 \mathrm{w}(G) \log \mathrm{w}(G) + \mathrm{w}(G) \cdot \min\{n^{2+\mu}, n m_{\mathrm{red}}\}\right)$, where m_{red} is the number of edges in the transitive reduction of G.*

Algorithm 7: All LCA Using Representative LCAs (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.

```
1  begin
2      Compute a transitive closure G_clo of G.
3      Compute a minimum chain cover 𝒞 of G_clo.
4      foreach c ∈ 𝒞 do
5          Compute a topological ordering top such that top(z) is maximal for all vertices of c.
6          Solve ALL-PAIRS REPRESENTATIVE LCA with respect to top and get matrix R.
7          foreach {x,y} with R[x,y] = z and z ∈ C do
8              A[x,y] ← A[x,y] ∪ {z} (by multiset-union)
9          end
10     end
11     Remove elements of multiplicity greater than one from A[x,y] for all x,y ∈ V.
12 end
```

5.4 Generalized Path Cover Approach

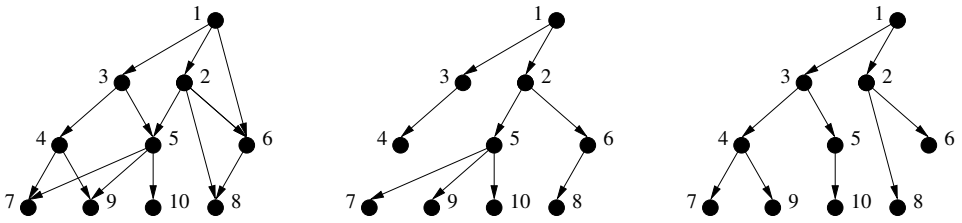
We continue by presenting an alternative, but natural and versatile approach to computing LCAs in dags based on using path covers. The method can be regarded as a generalization of ideas used in the previous section. Again, the efficiency of our decomposition technique depends mainly on the width $\mathrm{w}(G)$ of the underlying dag G.

We start by giving an intuitive description of our approach. Let $\{x,y\}$ be any vertex pair in G and let $z \in \mathrm{LCA}\{x,y\}$ be a lowest common ancestor of x and y. Suppose now that we start a depth first search (DFS) in G at vertex z. Let T_z be the corresponding DFS tree [CLRS01]. Then, it is not difficult to verify that $\mathrm{LCA}_{T_z}\{x,y\} = z$. Observe that the partial order induced by the tree is a suborder of G_{clo}. Moreover, for all vertices $w \in T_z$ it holds that w is reachable from z. Hence, since $z \in \mathrm{LCA}\{x,y\}$, the only common ancestor of x and y in T_z is z. Recall again, that the vertex with the highest topological number among $\mathrm{CA}\{x,y\}$ is a lowest common ancestor. This leads to a first naive solution to computing representative LCAs based on decomposing the problem:

1. For all $v \in V$, compute DFS trees T_v and preprocess the trees for constant time LCA queries.
2. Upon an $\mathrm{LCA}_G\{x,y\}$ query, execute $\mathrm{LCA}_{T_v}\{x,y\}$ queries for all $v \in V$. Observe that the answer to such a query might be NULL.
3. Let $Z = \bigcup_{v \in V} z_v$, where $z_v = \mathrm{LCA}_{T_v}\{x,y\}$, be the answers to the LCA queries in the DFS trees. Return z, where z is the vertex with the maximum topological number in Z.

The correctness of the above approach follows from the fact that the maximum CA z is returned at least once by the n LCA queries in the trees, i.e., by the query $\mathrm{LCA}_{T_z}\{x,y\}$. Moreover, since all returned vertices z_v are common ancestors of $\{x,y\}$, the topological number of z is maximal among the vertices in Z. Recall that trees can be preprocessed in linear time and space for constant time LCA queries. Hence, this first naive approach results in a preprocessing time of $O(nm)$. Subsequent queries for LCAs in G can be answered in time $O(n)$ implying an upper time bound of $O(n^3)$ for the ALL-PAIRS REPRESENTATIVE LCA problem. However, we show below that only $\mathrm{w}(G)$ DFS trees have to be considered.

Definition 5.6 (Special DFS Trees)**.** *For a path $p = (v_1, \ldots, v_l)$ in G, T_p is a special DFS tree obtained by starting the DFS at vertex v_1 and first exploring the edges along the path p.*



(a) input dag G (b) $(1,2,5,10)$ (c) $(1,3,4,7)$

Figure 5.1: Special DFS tree for the paths $(1,2,5,10)$ and $(1,3,4,7)$ in a dag G with 10 vertices.

In the following we denote by $T_p(v_i)$ the subtree of T_p rooted at vertex v_i for some vertex $v_i \in p$.

Lemma 5.7. *For a path $p = (v_1, \ldots, v_l)$ in G, $T_p(v_i)$ corresponds to a DFS tree T_{v_i} for all $1 \leq i \leq l$.*

Proof. Consider again the special DFS, i.e., the DFS that results in T_p, starting at v_1. The following is true for all $v_i \in p$. When v_i is visited for the first time by the DFS, all vertices that have been explored up to that point are predecessors of v_i in G. This follows from the fact that the DFS proceeds directly down the path p. This, in turn, implies that no successor of v_i has been reached since G is acyclic. The claim now follows from the recursive nature of depth first traversals. ■

The above lemma implies that given a path cover $\mathscr{P}$ of G, only $|\mathscr{P}|$ DFS trees have to be considered. We recapitulate our decomposition technique:

1. Compute a path cover $\mathscr{P} = \{p_1, \ldots, p_r\}$ of G.
2. Compute DFS-trees $T_{p_1}, \ldots, T_{p_r}$.
3. Preprocess the DFS-trees $T_{p_1}, \ldots, T_{p_r}$ for constant time LCA queries in trees.
4. Compute $Z = \bigcup z_i$ where $z_i = \mathrm{LCA}_{T_{p_i}}\{x,y\}$ for $1 \leq i \leq r$.
5. Derive the solution of $\mathrm{LCA}_G\{x,y\}$ from Z.

Lemma 5.8. *Let $\mathscr{P} = \{p_1, \ldots, p_r\}$ be a path cover of G. Next, let $Z = \bigcup z_i$, where $z_i = LCA_{T_{p_i}}\{x,y\}$ for $1 \leq i \leq r$. Then, the following chain of inclusion holds:*

$$\mathrm{LCA}\{x,y\} \subseteq Z \subseteq \mathrm{CA}\{x,y\}$$

Proof. The second inclusion follows again from the fact that each tree T_{p_i} induces a suborder on the poset induced by G. Hence, a common ancestor of $\{x,y\}$ in a tree is also a common ancestor of $\{x,y\}$ in G. Let z be an LCA of x and y. Let p_i be chosen such that $z \in p_i$. Since by Lemma 5.7, $T_{p_i}(z)$ corresponds to (some) DFS tree T_z, by the previous discussion we have $\mathrm{LCA}_{T_{p_i}}\{x,y\} = z$. ■

Remark 5.9. *By Lemma 5.8 both answers to representative LCA queries and all LCA queries in a dag $G = (V,E)$ can be derived from the set Z. For representative LCAs, we simply have to find the vertex with the maximum number in Z in time $O(|Z|)$. In order to find all LCAs, it is necessary to identify the set of vertices without outgoing edges in the subdag induced by Z. This can be achieved in time $O(|Z|^2)$. To this end, we assume that we have a mapping from V to $\mathscr{P}$ such that for each $v \in V$ a path P_i including v is known. With this mapping reachability queries of two vertices $z_1, z_2 \in Z$ can be answered in constant time by querying the LCA of $\{z_1, z_2\}$ in the corresponding tree.*

The following lemma follows immediately from the previously established facts.

Lemma 5.10. *For a path cover $\mathscr{P} = \{p_1, \ldots, p_r\}$ of a dag $G = (V,E)$ with n vertices and m edges,* ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $O(\mathscr{T}_{\mathrm{PC}}(G,r) + rm + rn^2)$, where $\mathscr{T}_{\mathrm{PC}}(G,r)$ is the time needed to compute $\mathscr{P}$.*

Proof. The special DFS trees for $p_1, \ldots, p_r$ can be constructed in time $O(rm)$. The preprocessing of the trees for LCA-queries takes time $O(nr)$ since the size of the trees is bounded by $O(n)$. Finally, since the resulting data structure supports representative LCA queries in $O(r)$, computing the solutions for all pairs takes $O(n^2 r)$ time. ■

Kowaluk and Lingas [KL07] have recently established the following.

Proposition 5.11. *For a dag $G = (V,E)$ with n vertices,* ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $\widetilde{O}(n^\omega + \mathrm{w}(G)n^2)$*

Furthermore, this bound is improved to $\widetilde{O}(\mathrm{w}(G)n^2)$ in [KL07] by using randomization. We continue by showing that the decomposition technique can be applied to obtain a simple algorithm for ALL-PAIRS REPRESENTATIVE LCA with deterministic upper time bound $\widetilde{O}(\mathrm{w}(G)n^2)$.

For a given minimum path cover of size $r = \mathrm{w}(G)$, the Lemma 5.10 immediately yields the claimed result. However, it is not known how to compute a minimum path cover in time $\widetilde{O}(n^2\mathrm{w}(G))$. Nonetheless, we show how to compute a path cover $\mathscr{P}$ of G of size $O(\mathrm{w}(G)\log n)$ in time $O(n^2\mathrm{w}(G)\log n)$, i.e., $\mathscr{P}$ is minimal up to a logarithmic factor. This, in turn, is used to improve the upper bounds of the ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA problems on dags of small width.

Recall that the best known time bounds for computing a minimum path cover of a dag G are $O(n^{2.5})$ and $O(nm_{\mathrm{red}})$ for deterministic algorithms and $O(n^\omega)$ for randomized algorithms, see Section 5.2. On the other hand, Felsner et al. [FRS03] showed that it is possible to recognize a poset P (which corresponds to a transitive dag) of width at most k in time $O(n^2 k)$. In their approach, the parameter k is given beforehand and – in the case that the width of P is bounded by k – their algorithm can be extended to output a chain cover of size k. However, the approach works only for transitive dags and cannot be directly applied to general dags.

We proceed by describing an algorithm that effectively yields a path cover of size $O(\mathrm{w}(G)\log n)$ for any dag G. It is based on the ideas in [FRS03]. To this end, we define the concept of a *greedy path cover*. Again, we stress that this concept is not equivalent to the approach taken in Section 3.2 in the context of computing a greedy chain cover for constructing the transitive closure of a dag.

Definition 5.12 (Greedy Path Cover)**.** *Let $G = (V,E)$ be a dag. A greedy path cover of G is obtained by iteratively finding paths $p_1, \ldots, p_r$ with $\bigcup_{1 \le i \le r} p_i = V$ such that for all p_i, the value $|p_i \setminus \bigcup_{k \le i-1} p_k|$, i.e., the number of vertices on p_i that are not covered by any of the paths $p_1, \ldots, p_{i-1}$, is maximized.*

We decompose the dag into paths by iteratively finding paths that contain as much uncovered vertices as possible. In [FRS03], a similar approach is used to cover a partial order P by chains in a greedy manner. In their approach, all covered vertices are removed from the partial order. Since we do not want to construct the poset G_{clo}, we cannot discard vertices. However, the following lemma establishes a one-to-one correspondence between covering G and G_{clo}. By slight abuse of notation, we use set notation in conjunction with paths and chains, which actually are defined as sequences. For example, $p \setminus V'$ denotes the path p with vertices $v \in V'$ deleted. The relative order of the remaining vertices is unchanged.

Lemma 5.13. *In a dag $G = (V,E)$, $\mathscr{P} = \{p_1, \ldots, p_r\}$ is a greedy path cover of G if and only if $\mathscr{C} = \{c_1, \ldots, c_r\}$ is a greedy chain cover of G_{clo}, where $c_i = p_i \setminus \bigcup_{k \leq i-1} p_k$.*

Proof. We proof the lemma by induction on i. For $i = 1$, we obviously have $p_1 = c_1$. Suppose now the claim is true for $k = 1, \ldots, i-1$. Let c_i be a chain of maximum size in G_{clo} restricted to $V \setminus \bigcup_{k \leq i-1} p_k$. Then, by definition, there exists a path p_i in G such that the number of uncovered vertices on p_i is equal to $|c_i|$. On the other hand, suppose that p_i is a path in G such that the number of uncovered vertices is strictly larger than $|c_i|$. Then, again, it is easy to see that this implies an existence of a chain of size strictly larger than $|c_i|$ in G_{clo} restricted to $V \setminus \bigcup_{k \leq i-1} p_k$ contradicting our assumption. ■

The above lemma effectively implies that a greedy path cover is minimal up to a logarithmic factor.

Corollary 5.14. *For $\mathscr{P} = \{p_1, \ldots, p_r\}$ be a greedy path cover of a dag G with n vertices and width $\mathrm{w}(G)$, $r \leq \mathrm{w}(G) \log n$.*

Proof. In [FRS03], it is shown that the size of a greedy chain cover of a partial order P is at most $\mathrm{w}(P) \log n$. The corollary now follows from Lemma 5.13 and the fact that $\mathrm{w}(G_{\text{clo}}) = \mathrm{w}(G)$. ■

Lemma 5.15. *A greedy path cover $\mathscr{P} = \{p_1, \ldots, p_r\}$ of a dag $G = (V,E)$ with m edges can be computed in time $O(mr)$.*

Proof. The (weighted) single source longest path problem can be solved in time $O(n+m)$ by using standard algorithms for solving the single source shortest path problem in dags, see, e.g., [CLRS01]. Observe that arbitrary edge weights are possible. We assume without loss of generality that G is equipped with a single source s, otherwise we simply add a super source. We initialize each edge in G with weight 1. Then, we iteratively solve the single source longest path problem for the source s. After the ith iteration we set the weight of each edge $e = (v,w)$ such that $w \in p_i$ to 0. It is easy to check that the weight of p_i is equal to the number of uncovered vertices on p_i. Hence, we obtain a greedy path cover. It is also easy to see that each step can be implemented in time $O(m)$. ■

Corollary 5.14, and Lemma 5.15 imply the main result of this section.

Theorem 5.16. ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $O(n^2 \mathrm{w}(G) \log n)$ and* ALL-PAIRS ALL LCA *can be solved in time $O(n^2 \mathrm{w}(G)^2 \log^2 n)$ on a dag G with n vertices and width $\mathrm{w}(G)$.*

This result improves the upper bound of ALL-PAIRS REPRESENTATIVE LCA (Prop. 5.11) for dags with $\mathrm{w}(G) = O(n^{\omega-2-\delta})$ and the general upper time bound for ALL-PAIRS ALL LCA (Thm. 4.19) for dags with $\mathrm{w}(G) = O(n^{\frac{\omega(2,1,1)-2}{2}-\delta})$ for a constant $\delta > 0$. We note that results on the expected value of the width of $G_{n,p}$ random dags [BE84, Sim88] imply that the average case complexity of this approach can be bounded by $\widetilde{O}(n^2)$ both for ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA in the case that p is a constant.

5.5 Combining Small Width and Low Depth

The path cover technique described in the previous section can be naturally used together with the solution for dags with low depth given in [CKL07]. Recall that the depth of a vertex v, denoted by $\mathrm{dp(v)}$, is defined as the length of longest path from a source vertex to v. The depth of a dag G, $\mathrm{dp(G)}$, is given by $\mathrm{dp(G)} = \max_{\mathrm{v}\in\mathrm{V}}\{\mathrm{dp(v)}\}$. The following proposition is due to Czumaj et al. [CKL07].

Proposition 5.17. *On a dag $G = (V,E)$ be with n vertices and depth $\mathrm{dp(G)} = \mathrm{n^q}$,* ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $\widetilde{O}(n^{\omega} + n^{q+\omega(1,1-q,1)})$.*

This result is achieved by exploiting that a common ancestor of maximum depth in a dag G is a lowest common ancestor. That is, $z = \mathrm{argmax}_{z'\in\mathrm{CA}\{x,y\}}\mathrm{dp(z')}$ implies $z \in \mathrm{LCA}\{x,y\}$. Observe that a possible witness would have greater depth by definition. The following lemma can be derived from the above proposition.

Lemma 5.18. *For a dag $G = (V,E)$ with n vertices and depth $\mathrm{dp(G)} = \mathrm{n^q}$, the* ALL-PAIRS REPRESENTATIVE LCA *problem can be solved in time $\widetilde{O}(n^{2+\mu-\delta})$ if $q \leq 1-\mu-\delta$ for an arbitrary small constant $\delta > 0$.*

Proof. $\widetilde{O}(n^{q+\omega(1,1-q,1)})$ is an upper time bound for ALL-PAIRS REPRESENTATIVE LCA in G by Proposition 5.17. By Proposition 4.4, we have

$$\omega(1,1-q,1) \leq \begin{cases} 2+o(1) & \text{if } 0 \leq 1-q \leq \alpha \\ 2+\frac{\omega-2}{1-\alpha}(1-q-\alpha)+o(1) & \text{if } \alpha \leq 1-q \leq 1. \end{cases}$$

Let in the following $\beta = \frac{\omega-2}{1-\alpha}$. We aim at solving the following inequality for q:

$$q+\omega(1,1-q,1) \leq 2+\mu-\delta \tag{5.1}$$

Recall that μ satisfies $\omega(1,\mu,1) = 1+2\mu$, i.e.,

$$\mu = \frac{1-\beta\alpha}{2-\beta}. \tag{5.2}$$

Now we plug (5.2) into (5.1) and solve for q:

$$\begin{aligned}
q+\omega(1,1-q,1) &\leq 2+\mu-\delta \\
2+q(1-\beta)+\beta(1-\alpha) &\leq 2+\frac{1-\beta\alpha}{2-\beta}-\delta \\
q &\leq \frac{1-\beta\alpha-\beta(1-\alpha)(2-\beta)}{(2-\beta)(1-\beta)}-\delta \\
q &\leq \frac{1+\beta\alpha-2\beta+\beta^2-\beta^2\alpha}{(2-\beta)(1-\beta)}-\delta \\
q &\leq \frac{(1-\beta)(1-\beta+\beta\alpha)}{(2-\beta)(1-\beta)}-\delta \\
q &\leq \frac{1-\beta+\beta\alpha}{2-\beta}-\delta \\
q &\leq 1-\mu-\delta
\end{aligned}$$

■

Assume first that we have already computed G_{clo}. On a high level, the combination of the above result with our path cover technique works as follows:

1. Construct a partial chain cover $\mathscr{C} = \{c_1, \ldots, c_r\}$ of G_{clo} greedily. That is, search for chains of maximum size until a termination criterion (to be specified below) is satisfied. Note that we do not require that the chains cover all vertices of G_{clo}. Note further that in general $r \neq \mathrm{w}(G)$.
2. Construct the special DFS trees associated with the chains as described in the previous chapter and prepare them for constant time LCA queries.
3. Reduce G_{clo} along the chains. That is, let $V_C = \bigcup_{1\leq i\leq r} c_i$. Then, remove all edges (v,w) such that $v \in V_C$. Observe that all edges (v,w) with $v \notin V_C$ are retained. The resulting graph is called the *reduced dag* denoted by $G^R = (V, E^R)$.
4. Compute maximum depth common ancestors in G^R. The maximum depth CAs are either lowest common ancestors in G or all of their witnesses are in the set V_C. Thus, in a second step we search for possible witnesses by querying the special DFS trees. If witnesses exist, i.e., common ancestors that are successors of the maximum depth CAs, we output the witness with highest label, which corresponds to a lowest common ancestor.

The reasoning behind this approach is as follows. By decomposing the graph along the longest chains we reduce successively the depth of G_{clo}. As soon as we reach a certain threshold depth, we can apply the algorithm given in [CKL07] to efficiently compute maximum depth CAs in the reduced dag. Moreover, if a maximum depth CA in the reduced dag G^R is not a lowest common ancestor in the original dag G, we know that all its witnesses are covered by the chains. Observe that outgoing edges of non-covered vertices are not removed. Hence, if z' is a witness of z and $\{x,y\}$ and we suppose $z' \notin V_C$, we have $z \rightsquigarrow z'$ and $z' \rightsquigarrow x,y$ in the reduced dag contradicting the fact that z is a maximum depth CA in G^R.

In the following we refer to the algorithm described in this section as the *combined algorithm*. Before proving the key properties of the approach we give a formal description of the decomposition in Algorithm 8.

Algorithm 8: Preprocessing

Input: A dag $G = (V,E)$ (and threshold parameters W and H).
Output: Special DFS trees $T_{c_1}, \ldots, T_{c_r}$ that are prepared for constant LCA queries and a reduced dag $G^R = (V, E^R)$.

```
begin
    Compute G_clo and initialize an empty partial chain cover 𝒞.
    Set G^R ← G_clo.
    repeat
        Find a chain c of maximum size in G^R.
        Add c to 𝒞 and remove all edges (v,w) from G^R such that v ∈ C.
    until termination criterion
    Compute special DFS trees T_{c_1}, ..., T_{c_r} on G_clo as described in the previous section.
    Prepare the special DFS trees for constant time LCA queries.
end
```

Lemma 5.19. *Let z_h be a maximum depth common ancestor of a pair $\{x,y\}$ in G^R. Then, either $z \in \mathrm{LCA}_G\{x,y\}$ or all witnesses of z_h, i.e., vertices z' such that $z_h \rightsquigarrow z'$ and $z' \in \mathrm{CA}_G\{x,y\}$ are in the set V_C.*

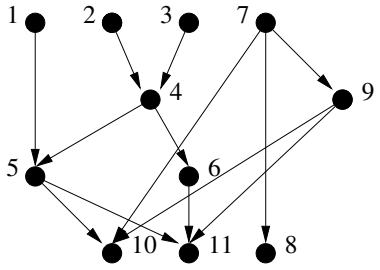
Proof. Suppose that z_h is not an LCA of $\{x,y\}$ in G. In that case there exists a witness z' such that $z' \in \mathrm{LCA}_G\{x,y\}$ and z' is a successor of z. This immediately implies $z' \in V_C$. Indeed, suppose that $z' \notin V_C$. This implies (i) $\mathrm{dp}(\mathrm{z}') > \mathrm{dp}(\mathrm{z})$ since z' is a successor of z and (z,z') is not removed by the decomposition and (ii) $z \in \mathrm{CA}_{G^R}\{x,y\}$. Moreover, (i) and (ii) contradict the fact that z is the maximum depth CA of $\{x,y\}$ in G^R. ■

Observe that we have constructed special DFS trees for the chains in G_{clo}. However, in general it is also possible to use an approach that allows constructing the trees in G. This involves computing a partial greedy path cover of G instead of the partial greedy chain cover of G_{clo}. In any case, however, the reduced dag G^R results from reducing G_{clo}. A formal description of the method for finding representative LCAs is given in Algorithm 9.

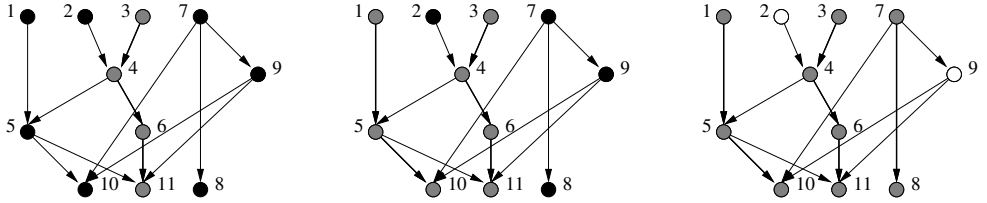
Algorithm 9: Algorithm for Representative LCA

Input: The reduced dag $G^R = (V, E^R)$ and special DFS trees $T_{c_1}, \ldots, T_{c_r}$ that are prepared for constant LCA queries (output of Algorithm 8).
Output: All-pairs representative LCA matrix.

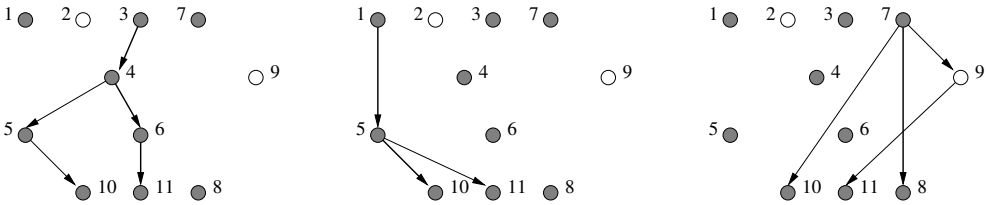
```
begin
    Compute maximum depth CAs on for all vertex pairs in G^R using the algorithm described in [CKL07].
    foreach vertex pair {x,y} do
        Let z_h be the maximum depth CA of {x,y} with respect to G^R.
        Initialize an empty result set Z.
        foreach chain C_i, 1 ≤ i ≤ r do
            Query LCA{x,y} in T_{c_i} and add the result to Z.
        end
        Remove all vertices from Z that are not successors of z_h.
        Return the maximum vertex in Z and z_h if Z is empty.
    end
end
```



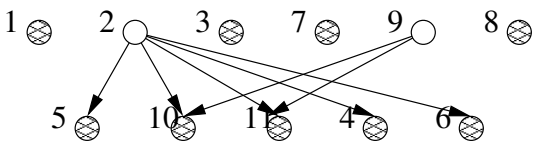
(a) input dag G



(b) computing a partial chain cover $\mathscr{C}$ in a greedy manner, $\mathscr{C} = \{(3,4,6,11),(1,5,10),(7,8)\}$



(c) the special DFS trees corresponding to the partial chain cover, the result set Z for queries for LCA$\{10,11\}$ in the trees is given by $Z = \{4,5,7\}$



(d) the reduced dag G^R, both 2 and 9 correspond to maximum depth CAs of $\{10,11\}$. However, only one of them is returned, e.g., 2.

Figure 5.2: Example for the combined algorithm on a dag with 11 vertices. Suppose that the maximum depth CA of $\{10,11\}$ in G^R that is returned is 2. Then, the maximum vertex in Z (7) is not an LCA. However, restricting the candidate vertices to successors of 2 excludes 7. In consequence 5 is returned and $5 \in \text{LCA}\{10,11\}$.

An example of the combined algorithm is given in Figure 5.5. The correctness of Algorithm 9 is a consequence of Lemma 5.19 and the results of Section 5.4. We turn our attention to the time complexity of this approach. Obviously, the running time depends on (i) r, the cardinality of the partial chain cover, and (ii) $\mathrm{dp}(\mathrm{G^R})$, the depth of the reduced dag. Our goal is an algorithm that does not exceed the worst case complexity for the general problem of $O(n^{2+\mu})$ but performs better in as many cases as possible. To this end, one can specify an implementation of the termination criterion in Algorithm 8 as follows: Pick threshold parameters W and H and terminate whenever $r \geq W$ or $\mathrm{dp}(\mathrm{G^R}) \leq \mathrm{H}$. Any reasonable choice of H should satisfy $H \leq n^{1-\mu}$ according to Lemma 5.18. Observe that any choice of W such that $W \leq n^{\mu}$ is sufficient to guarantee a worst case upper bound of $O(n^{2+\mu})$. Simply modify the combined algorithm such that it uses the general solution whenever $r \geq W$ (which implies $\mathrm{dp}(\mathrm{G^R}) > \mathrm{H}$). However, there is a more elegant way to find the optimal decomposition. The idea is to look for the intersection point of the functions $n^{q+\omega(1,1-q,1)}$ and rn^2, i.e., the functions that describe the asymptotic behavior of the two approaches. The respective termination criterion becomes (neglecting polylogarithmic factors)

$$rn^2 > n^{\log_n(\mathrm{dp}(\mathrm{G^R}))+\omega(1,1-\log_n(\mathrm{dp}(\mathrm{G^R})),1)}. \tag{5.3}$$

Theorem 5.20. *The time complexity of the combined algorithm on a dag G with n vertices can be bounded by $\widetilde{O}(n^{2+\mu})$, where μ satisfies $\omega(1,\mu,1) = 1+2\mu$.*

Proof. Let in the following $\beta = \frac{\omega-2}{1-\alpha}$. Let r be the cardinality of the path cover produced by Algorithm 8 and let n^q be the depth of the the reduced graph after step $r-1$. Since the reduction does not terminate after step $r-1$, by Equation (5.3) we have

$$r-1 \leq n^{q+\omega(1,1-q,1)-2}.$$

Moreover, observe that $\widetilde{O}(n^2 r)$ is obviously an upper time bound for the combined algorithm. We can safely assume that $q < 1-\alpha$. To see this, suppose first that $q \geq 1-\alpha$. Since $r-1$ paths cover at least n^q vertices, we have $r-1 \leq n^{1-q} \leq n^{\alpha}$. This in turn implies an upper bound of the time complexity of $O(n^{\omega})$ and we are done. We apply Proposition 4.4 and get

$$r-1 \leq n^{q+2+\beta(1-q-\alpha)-2+o(1)}.$$

This simplifies to

$$r-1 \leq n^{q(1-\beta)+\beta(1-\alpha)+o(1)}. \tag{5.4}$$

On the other hand, since we know that each of the $r-1$ paths covers at least n^q vertices we have

$$q \leq \log_n \frac{n}{r-1}. \tag{5.5}$$

Plugging (5.5) into (5.4) yields

$$r-1 \leq n^{\log_n \frac{n}{r-1}(1-\beta)+\beta(1-\alpha)+o(1)}.$$

Since we have $n^{\log_n \frac{n}{r-1}(1-\beta)+\beta(1-\alpha)} = \left(\frac{n}{r-1}\right)^{1-\beta} \cdot n^{\beta(1-\alpha)}$, this simplifies to

$$r-1 \leq n^{\frac{1-\beta\alpha}{2-\beta}+o(1)}. \tag{5.6}$$

On the other hand, recall that μ satisfies $\omega(1,\mu,1) = 1+2\mu$. Again, we use Proposition 4.4 to obtain

$$\mu = \frac{1-\beta\alpha}{2-\beta}. \tag{5.7}$$

Equations (5.6) and (5.7) conclude the proof. ■

As an immediate consequence of the above theorem we get the following corollary.

Corollary 5.21. *The time complexity of the combined algorithm for solving* ALL-PAIRS REPRESENTATIVE LCA *is $O(n^{2.575})$ on a dag $G = (V,E)$ with n vertices.*

Observe that the average case bound of $\widetilde{O}(n^2)$ is still valid under the assumption that the input space is distributed according to the $G_{n,p}$ model with constant edge probability p. To see this, recall that the transitive closure of a random dag in the $G_{n,p}$ model for arbitrary values of p can be computed in average case time $\widetilde{O}(n^2)$ by Proposition 3.2.

The combination of these two techniques narrows down the classes of dags for which no solution faster than $O(n^{2+\mu})$ is known considerably. Indeed, recall that for dags G of depth $\mathrm{dp(G)} \leq \mathrm{n}^{1-\mu-\delta}$ the ALL-PAIRS REPRESENTATIVE LCA problem can be solved in time $\widetilde{O}(n^{2+\mu-\delta})$ by Lemma 5.18.

Theorem 5.22. *For a dag G with n vertices and an arbitrary constant $\mu \geq \delta > 0$* ALL-PAIRS REPRESENTATIVE LCA *can be solved in time $\widetilde{O}(n^{2+\mu-\delta})$ if G does not contain a subgraph H that contains at least $n^{\mu-\delta}$ (vertex-disjoint) chains of length at least $n^{1-\mu-\delta}$.*

Observe that this is a significant restriction for bad dag classes, namely an almost linear-sized, i.e., $n^{1-2\delta}$ for an arbitrary small constant δ, subdag of extremely regular structure.

All-k-Subsets Representative LCA

The ALL-K-SUBSETS REPRESENTATIVE LCA problem is an extension of the ALL-PAIRS REPRESENTATIVE LCA problem. Given a constant $k \geq 2$, compute a representative LCA for each k-subset of vertices in G.

Problem:	ALL-K-SUBSETS REPRESENTATIVE LCA
Input:	A dag $G = (V,E)$
Output:	A matrix R of size n^k such that for all k-subsets of vertices $\{x_1,\ldots,x_k\}$, $R[x_1,\ldots,x_k] \in \mathrm{LCA}\{x_1,\ldots,x_k\}$; if $\mathrm{LCA}\{x_1,\ldots,x_k\} = \emptyset$ for some vertex k-subset $\{x_1,\ldots,x_k\}$, then $R[x_1,\ldots,x_k] =$ NIL.

The problem has been considered recently by Yuster [Yus]. He shows that the ALL-K-SUBSETS REPRESENTATIVE LCA problem can be solved in time $O(n^{3.575})$ for $k = 3$ and $O(n^{k+1/2})$ for $k \geq 4$. We improve slightly upon the bound for $k = 3$ by using our combined approach.

Theorem 5.23. *The* ALL-K-SUBSETS REPRESENTATIVE LCA *problem on a dag G with n vertices can be solved in time $O(n^{3.5214})$ for $k = 3$ and $\widetilde{O}(n^{k+\frac{1}{2}})$ for $k \geq 4$.*

Proof. Let $G = (V,E)$ be a dag of depth $\mathrm{dp(G)} = \mathrm{n^q}$. Combining the ideas in [Yus] and [CKL07] for computing LCAs in dags of small depth, ALL-K-SUBSETS REPRESENTATIVE LCA can be solved by computing (arbitrary) witnesses for Boolean matrix products MM^T for each level of the dag. More specifically, let l_i be the number of vertices on level L_i, then the matrix M corresponding to level L_i is an $n^{\binom{n}{k/2}} \times n^{l_i}$ matrix. Further, by Jensen's inequality (Prop. 4.22), the time complexity of the algorithm is maximized if the levels of G are of equal size. That is, the running time of this approach can be bounded by $\widetilde{O}(n^{q+\omega(1,\frac{2(1-q)}{k},1)\frac{k}{2}})$. The additional polylogarithmic factor results again from the witness computations.

On the other hand, we note that the LCA of a k-subset of vertices in a preprocessed tree can be computed in $O(1)$ since we assume that k is a constant. This implies that ALL-K-SUBSETS REPRESENTATIVE LCA can be solved in time $O(n^k \mathrm{w}(G))$ by extending the ideas in Section 5.4.

Now we combine these two approaches in an analogous way as described above. We get an algorithm with time complexity

$$\widetilde{O}(rn^k + n^{q+\omega(1,\frac{2(1-q)}{k},1)\frac{k}{2}}) \text{ for } \frac{2(1-q)}{k} > 0.294$$

and

$$\widetilde{O}(rn^k + n^{q+k}) \text{ for } \frac{2(1-q)}{k} \leq 0.294,$$

where $q \leq \log_n \frac{n}{r}$. Observe first that for $k \geq 4$ the respective function is minimized for $q = \frac{1}{2}$, which implies a time bound of $\widetilde{O}(n^{k+\frac{1}{2}})$.

Let now $k = 3$ and assume first that $\frac{2(1-q)}{3} \leq 0.294$. This implies $q \geq 0.559$ and hence an upper bound of $\widetilde{O}(n^{k+0.559})$. Suppose now that $\frac{2(1-q)}{3} > 0.294$. We solve the equality

$$rn^k = n^{q+\omega(1,\frac{2(1-q)}{k},1)\frac{k}{2}}$$

By using $q \leq \log_n \frac{n}{r}$ and $\beta = \frac{\omega-2}{1-\alpha}$ we find the following bound for r in a similar way as in the proof of Theorem 5.16.

$$r \leq n^{1-\frac{3}{2}\beta\alpha}$$

The claim follows now since $0.5214 > 1 - \frac{3}{2}\beta\alpha$. ■

Note that Theorem 5.23 slightly subsumes Yuster's upper time bound for $k = 3$ and matches his remaining upper bounds for $k \geq 4$ [Yus] ignoring polylogarithmic factors. Again, we observe that if the input dags are distributed according to the $G_{n,p}$ model for random dags such that p is a constant, ALL-K-SUBSETS REPRESENTATIVE LCA can be solved in time $\widetilde{O}(n^k)$ in the average case. Nonetheless, we can even state the following corollary which is valid for all choices of the edge probability p.

Corollary 5.24. ALL-K-SUBSETS REPRESENTATIVE LCA *can be solved in* $\widetilde{O}(n^k)$ *in the average case.*

Proof. We modify Algorithm 2. The following claim is a straightforward generalization of Lemma 3.6.

Claim 5.25. *Let $G = (V,E)$ be a dag and let $\{y_1,\ldots,y_k\}$ be a k-subset of vertices. Furthermore, let z be the maximum CA (MCA) of $\{y_1,\ldots,y_k\}$.*

1. *If y_1 is an ancestor of $y_2, \ldots, y_k$, then $z = y_1$.*
2. *If y_1 is not an ancestor of $y_2, \ldots, y_k$, then the following holds: Let $\{x_1, \ldots, x_l\}$ be the parents of y_1. Let $Z = \bigcup_{1 \leq i \leq l} MCA\{x_i, y_2, \ldots, y_k\}$. Then $MCA\{y_1, \ldots, y_k\}$ is the maximum vertex in Z.*

The above claim yields a dynamic programming algorithm that is analogous to Algorithm 2. By an analysis similar to the one given in the proof of Theorem 3.7, we can bound the running time by $O(m_{\text{red}}\, n^{k-1})$ which implies an average case complexity of $O(n^k \log n)$ by Proposition 3.4. ■

5.6 Conclusion

The width $\mathrm{w}(G)$ of a dag G represents a crucial parameter in the context of classifying the difficulty of LCA problems on certain dag classes. That is, die upper time bounds for ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA improve significantly on dags with small width. However, a second major contribution of the results presented in this chapter is the concise characterization of difficult dag classes as a consequence of the combined algorithm described in Section 5.5. This together with the results of the previous chapters might indicate a direction that ultimately yields an improved upper time bound for ALL-PAIRS REPRESENTATIVE LCA and possibly also ALL-PAIRS ALL LCA in the general case. To achieve this, it seems to be critical to dissolve from algorithmic solutions that are purely based on Proposition 2.10 as done in the case of the combined algorithm. The extremely regular structure of worst case dag classes, i.e., $n^{\mu-\delta}$ levels with $n^{1-\mu-\delta}$ vertices in each level, might provide exploitable combinatorial properties. Even if the upper time bound for ALL-PAIRS REPRESENTATIVE LCA can not be improved in the general case, the combined algorithm is the currently best solution in terms of worst case upper time bound and efficiency over the broad range of possible input dags since the purely matrix-multiplication-based algorithm presented in Section 4.3 takes $\Omega(n^{2+\mu})$ time in any case.

In practice, algorithms whose efficiency depends on small width do 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.

CHAPTER

6

ALGORITHMIC APPLICATIONS

6.1 Introduction

While the main focus in this work is on ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA in a fixed dag G, a variety of problem variants is motivated by applications and/or related theoretical questions. In this chapter we extend and combine the approaches developed for the two problem primitives in Chapters 3, 4, and 5 to obtain solutions to some natural variations. In particular, we consider LCA problems in weighted dags (Section 6.2), LCA problems in a dynamic setting (Section 6.3), and space-efficient LCA algorithms, i.e., LCA algorithms that use less than $\Theta(n^2)$ space (Section 6.4). To the best of our knowledge, we initiate the particular study of most of the problem variants considered in this chapter. Accordingly, we compare the quality of our solutions mainly to naive or trivial approaches. Introductory paragraphs are deferred to the beginnings of the respective sections along with a detailed description of the results. Since we have not yet given a concise description of the considered problems at this point, we outline the main results in this chapter in a more abstract way than in the previous chapters.

1. We present algorithms and computational equivalence results for (L)CA problems in edge- and vertex-weighted dags. The CA versions of the problems seem to be much easier than the LCA versions in general. To solve the LCA versions in full generality, our algorithms rely on ALL-PAIRS ALL LCA solutions, which implies a hard $\Omega(n^3)$ lower bound. It is not clear and an intriguing open question whether this can be improved to subcubic running time.
2. We improve update times of ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA in dynamic settings (fully and partially dynamic) slightly over naive recomputation from scratch. Essentially, the improvement in the fully dynamic case relies on reducing the key ingredients of the algorithms presented in Chapter 4, namely the (rectangular) matrix products, to transitive closure computations – a problem that is well understood in dynamic contexts. An improvement for an incremental problem version is obtained by combining several algorithmic approaches, i.e., the non-optimal ALL-PAIRS REPRESENTATIVE LCA algorithm given in [BFCP$^+$05], an

online topological order algorithm by Pearce and Kelly [PK06], the fully dynamic algorithm for ALL-PAIRS REPRESENTATIVE LCA, and a dynamic ALL-PAIRS SHORTEST DISTANCES algorithm by Thorup [Tho04].

3. We show that LCA computations in a dag G can be solved in a space-efficient way if the width $\mathrm{w}(G)$ of G is accordingly bounded. The approach is derived from the path cover technique presented in Chapter 5. Furthermore, by combining the path cover approach with a rather simple datastructure, we are able to show that space-efficient solutions exist in the average case.

6.2 (L)CA Problems in Weighted Dags

In many real-world scenarios accurate modeling of causality systems by dags requires the use of *weight functions*, i.e., a real-valued weight is associated with each edge, vertex, or both. Consider computing lowest common ancestors in phylogenetic networks as described in Section 1.1. In some cases it is possible to assign edge weights to such networks by considering speciation distances based on available evolutionary data. The distance between two vertices on a path in such networks is proportional to elapsed time, e.g., time between a speciation event and the present. If we consider and analyze phylogenetic networks with edge weights, naturally, we extend our notion of LCAs to questions of the following kind: What is the most recent speciation event between pairs of species? What is the first speciation event between pairs of species? On the other hand associating providers with a certain cost can be modeled by vertex weights in the Internet dag. Then, we make the transition from computing arbitrary top providers to computing cheapest (most expensive) top providers.

With respect to weighted dags the problem of computing representative LCAs is readily generalized to questions of the following flavor: What is the (L)CA that minimizes the *ancestral distance* of a vertex pair $\{x,y\}$? What is the (L)CA with minimum vertex weight of a vertex pair $\{x,y\}$?

Bender et al. [BPSS01] have shown that shortest ancestral (CA) distances in unweighted dags can be computed in time $O(n^{2.575})$. In the same work, the authors consider a restricted version of the problem of computing shortest LCA distances, i.e., only ancestral distances with respect to LCAs are considered. Here, additional information from genealogical data is used to rank candidate LCAs in pedigree graphs. This ranking implies that shortest distance CAs are LCAs. However, the general version seems to be much more difficult.

Let $G = (V,E)$ be dag. G is called *edge-weighted* if it is equipped with an edge weight function $w_e : E \to \mathbb{R}$. G is *vertex-weighted* if it is equipped with a vertex weight function $w_v : V \to \mathbb{R}$. In edge-weighted dags the distance between two vertices u and v, denoted by $\mathrm{d}(u,v)$, is defined as the minimum weight of an u-v-path, where the path weight is given by the sum of the weights of the edges on the path. Computing shortest distances and paths in graphs is a fundamental problem in algorithmic graph theory and has been studied extensively. In the context of this chapter, we restrict our attention to the ALL-PAIRS SHORTEST DISTANCES problem (APSD), i.e., the problem of computing shortest distances between all vertex pairs. ALL-PAIRS SHORTEST DISTANCES can be solved in time $O(nm)$ for arbitrary edge weights [CLRS01]. For special classes of edge weights, faster solutions are available, e.g., $O(n^{2+\mu})$ for integer weights of constant absolute value [Zwi02].

While in general the research focus is on edge-weighted graphs, a number of interesting algorithmic problems and solutions on vertex-weighted graphs have been studied recently. This includes, e.g., computing *bottleneck paths* [SYZ07] or finding triangles of maximum weight [CL07].

Suppose first that G is edge-weighted. Before formally defining corresponding common ancestor problems, we introduce the notion of *ancestral distance*.

Definition 6.1 (Ancestral Distance)**.** *Let $G = (V,E)$ be dag with edge weight function $w_e : E \to \mathbb{R}$. Let $\{x,y\}$ be a vertex pair. The ancestral distance of $\{x,y\}$, denoted by $\mathrm{AD}\{x,y\}$, is defined as*

$$\mathrm{AD}\{x,y\} = \begin{cases} \min\limits_{z \in \mathrm{CA}\{x,y\}} d(z,x) + d(z,y) & \text{if } \mathrm{CA}\{x,y\} \neq \emptyset \\ +\infty & \text{else.} \end{cases}$$

In the following we denote by $\mathrm{AD}_z\{x,y\}$ the ancestral distance of x and y with respect to vertex z, i.e., $\mathrm{AD}_z\{x,y\} = \mathrm{d}(z,x) + \mathrm{d}(z,y)$. Using the above definition we define the ALL-PAIRS SHORTEST DISTANCE CA problem in edge-weighted dags.

Problem:	ALL-PAIRS SHORTEST DISTANCE CA
Input:	A dag $G = (V,E)$ with weight function $w : E \to \mathbb{R}$.
Output:	A matrix *SDC* of size $n \times n$ such that for all vertices $x,y \in V$, $SDC[x,y] = \mathrm{argmin}_{z \in \mathrm{CA}\{x,y\}} \mathrm{AD}_z\{x,y\}$; if $\mathrm{CA}\{x,y\} = \emptyset$ for some vertex pair $\{x,y\}$, then $SDC[x,y] =$ NIL. (Ties are arbitrarily broken.)

Note that we want to compute the respective vertices and not only ancestral distances. Given a solution to ALL-PAIRS SHORTEST DISTANCES in G, the shortest ancestral distances can be readily derived from the matrix *SDC*.

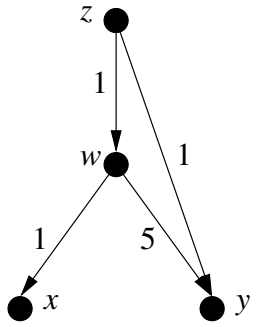


Figure 6.1: z is the shortest distance CA of $\{x,y\}$ but $z \notin \mathrm{LCA}\{x,y\}$.

Observe that a shortest distance CA does not necessarily correspond to an LCA as the example in Figure 6.1 shows. This motivates the ALL-PAIRS SHORTEST DISTANCE LCA problem.

Problem:	ALL-PAIRS SHORTEST DISTANCE LCA
Input:	A dag $G = (V,E)$ with weight function $w : E \to \mathbb{R}$.
Output:	A matrix *SDL* of size $n \times n$ such that for all vertices $x,y \in V$, $SDL[x,y] = \mathrm{argmin}_{z \in \mathrm{LCA}\{x,y\}} AD_z\{x,y\}$; if $\mathrm{CA}\{x,y\} = \emptyset$ for some vertex pair $\{x,y\}$, then $SDL[x,y] =$ NIL. (Ties are arbitrarily broken.)

We turn our attention to vertex-weighted dags. The problem definitions of (L)CA problems for vertex-weighted dags are analogous. Observe again that we distinguish the consideration of general common ancestors and the restriction to lowest common ancestors.

Problem:	ALL-PAIRS MINIMUM WEIGHT CA
Input:	A dag $G = (V,E)$ with (vertex) weight function $w : V \to \mathbb{R}$.
Output:	A matrix MWC of size $n \times n$ such that for all vertices $x,y \in V$, $MWC[x,y] = \operatorname{argmin}_{z \in \mathrm{CA}\{x,y\}} w(z)$; if $\mathrm{CA}\{x,y\} = \emptyset$ for some vertex pair $\{x,y\}$, then $MWC[x,y] =$ NIL. (Ties are arbitrarily broken.)

Problem:	ALL-PAIRS MINIMUM WEIGHT LCA
Input:	A dag $G = (V,E)$ with (vertex) weight function $w : V \to \mathbb{R}$.
Output:	A matrix MWL of size $n \times n$ such that for all vertices $x,y \in V$, $MWL[x,y] = \operatorname{argmin}_{z \in \mathrm{LCA}\{x,y\}} w(z)$; if $\mathrm{CA}\{x,y\} = \emptyset$ for some vertex pair $\{x,y\}$, then $MWL[x,y] =$ NIL. (Ties are arbitrarily broken.)

We summarize the results presented in this section.

1. We prove that it is possible to reduce ALL-PAIRS SHORTEST DISTANCE CA to ALL-PAIRS SHORTEST DISTANCES, up to a polylogarithmic factor. This in turn implies that the complexity of ALL-PAIRS SHORTEST DISTANCE CA essentially depends on the edge weight function under consideration. In the case of integral edge weights of small absolute value the result leads to subcubic algorithms. Moreover, we describe a dynamic programming approach with complexity $O(nm)$ for ALL-PAIRS SHORTEST DISTANCE CA.
2. We establish a computational equivalence of ALL-PAIRS MINIMUM WEIGHT CA and ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION. As a consequence ALL-PAIRS MINIMUM WEIGHT CA can be solved in $O(n^{2+\mu})$ time.
3. We derive upper bounds for ALL-PAIRS SHORTEST DISTANCE LCA and ALL-PAIRS MINIMUM WEIGHT LCA by using ALL-PAIRS ALL LCA solutions. Additionally, we show that ALL-PAIRS MINIMUM WEIGHT LCA can be solved substantially faster if the vertex weight function has benign properties, i.e., the function respects topological properties up to a certain error $f(n)$. In such cases ALL-PAIRS MINIMUM WEIGHT LCA can be solved in time $O(n^{2+\mu} + f(n)^2 n^2)$.

6.2.1 Common Ancestor Problem Variants

We start this section by establishing the close relationship between ALL-PAIRS SHORTEST DISTANCES and ALL-PAIRS SHORTEST DISTANCE CA. In particular, we show that ALL-PAIRS SHORTEST DISTANCE CA can be reduced to ALL-PAIRS SHORTEST DISTANCES, up to a polylogarithmic factor.

There is a close relationship between computing shortest-distance CAs and computing shortest ancestral distances. On the other hand, it is possible to reduce the problem of computing shortest ancestral distances to the ALL-PAIRS SHORTEST DISTANCES problem in dags. This reduction is inspired by ideas used for reducing ALL-PAIRS COMMON ANCESTOR EXISTENCE to transitive closure computation, see Chapter 4 and in particular Figure 4.1.

There are classes of dags on which ALL-PAIRS SHORTEST DISTANCES can be solved in time $o(n^3)$, e.g., dags in which the edge weights are small integer weights of absolute value $O(n^t)$ and $t < 3 - \omega$. However, it is not obvious how to derive the shortest distance CAs from shortest ancestral distances. Our approach is based on ideas in [Sei95, Zwi02] used for computing witnesses for shortest paths. A detailed description of the reduction and the identification of the shortest distance CAs is given in the proof of Theorem 6.2.

Theorem 6.2. ALL-PAIRS SHORTEST DISTANCE CA *can be reduced to* ALL-PAIRS SHORTEST DISTANCES*, up to a polylogarithmic factor.*

Proof. Let $G = (V,E)$ be a dag with weight function $w_e : E \to \mathbb{R}$. Let $V = \{v_1, \ldots, v_n\}$. We construct a dag $G' = (V',E')$ with edge weight function w'_e as follows:

- $V' = X \cup Z \cup Y$, where $X = \{x_1, \ldots, x_n\}$, $Z = \{z_1, \ldots, z_n\}$, and $Y = \{z_1, \ldots, z_n\}$.
- $E' = E_X \cup E_Y \cup E_Z$, where
  - $E_X = (x_i, x_j)$ for all $1 \leq i, j \leq n$ such that $(v_j, v_i) \in E$.
  - $E_Y = (y_i, y_j)$ for all $1 \leq i, j \leq n$ such that $(v_i, v_j) \in E$.
  - $E_Z = (x_i, z_i), (z_i, y_i)$ for all $1 \leq i \leq n$.

Furthermore, we define the following weight function $w'_e : E' \to \mathbb{R}$.

$$w'_e(u,v) = \begin{cases} w_e(v_j, v_i) & \text{if } u = x_i,\ v = x_j \text{ and } (v_j, v_i) \in E, \\ w_e(v_i, v_j) & \text{if } u = y_i,\ v = y_j \text{ and } (v_i, v_j) \in E, \\ 0 & \text{else.} \end{cases}$$

The graph G' (see Figure 6.2) is structurally similar to the graph given in Figure 4.1 in the context of the CA existence problem. However, we represent edges between vertex pairs (x_i, y_i) by additional vertices z_i and use edge weights. It is easy to see that

$$d_{G'}(x_i, y_j) = \min_{z \in \mathrm{CA}(v_i, v_j)} d_G(z, v_i) + d_G(z, v_j),$$

i.e., the distance between x_i and y_j in G' equals the ancestral distance between v_i and v_j in G for all $1 \leq i, j \leq n$. Obviously, the graph G' can be constructed in time $O(n+m)$.

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

$$D[v_i, v_j] = d_{G'}(x_i, y_j)$$

for all $v_i, v_j \in V$. The overall process takes time $O(\mathscr{T}_{\mathscr{A}}(3n, 2m + 2n))$, where $\mathscr{T}_{\mathscr{A}}(n,m)$ is the time needed by algorithm $\mathscr{A}$ on a dag with n vertices and m edges. However, we cannot directly read off

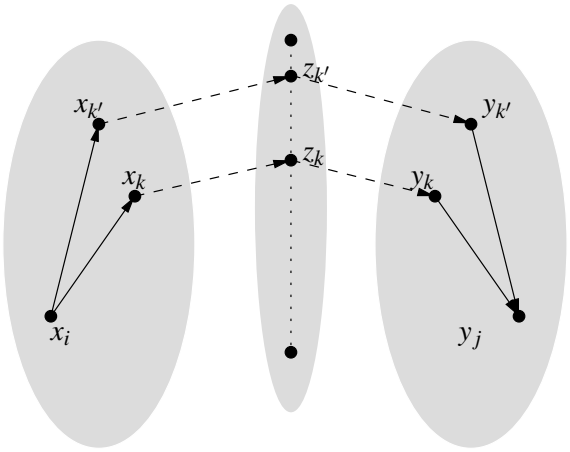


Figure 6.2: Solving ALL-PAIRS SHORTEST DISTANCE CA for a dag G with ALL-PAIRS SHORTEST DISTANCES in a dag G' that is constructed from G. Dashed lines have weight zero.

the shortest distance common ancestors themselves. Instead, we make use of techniques described in [Zwi02] to compute witnesses for distance products. This only adds a polylogarithmic factor to the overall running time. In the following, we describe the witness computation technique. Readers familiar with this approach can skip the rest of the proof.

The main idea behind the witness construction is the following: Suppose, G'' is a graph in which one of the vertices from V_Z, say the vertex z_k, $1 \leq k \leq n$, is contained, but all other vertices from V_Z and all their adjacent edges are deleted. Further, let D' and D'' be the distance matrices of G' and G'', respectively. Apparently, for all vertices v_i, v_j with $1 \leq i, j \leq n$, $d_{G''}(x_i, y_j) = d_{G'}(x_i, y_j)$ implies that z_k lies on some shortest path from x_i to y_j in G' and, by construction of G', that v_k corresponds to a shortest distance CA of $\{v_i, v_j\}$ in G. Thus, we could find all pairs $\{v_i, v_j\}$ such that v_k is a shortest distance CA of v_i and v_j by solving the ALL-PAIRS SHORTEST DISTANCES problem on G'' and comparing D'' with D', element by element. Alas, trying all vertices from $V_Z = \{z_1, \ldots, z_n\}$ one by one yields $O(n)$ ALL-PAIRS SHORTEST DISTANCES computations in order to derive the shortest distance CAs.

In the following we outline a randomized approach. Assume that a pair $\{v_i, v_j\}$ has exactly c shortest distance CAs. We first describe how to sample a subset $V'_Z \subseteq V_Z$ (which represents the candidates for shortest distance CAs in G) such that there is only one candidate in V'_Z for the pair $\{v_i, v_j\}$ 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'_Z as a multiset of size d uniformly at random from $\{z_1, \ldots, z_n\}$ with repetition. The probability of choosing exactly one of the c ancestors into V'_Z is greater than $\frac{1}{2e}$. This holds for each pair which has c shortest distance 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'_Z of size d each. Then, the probability that in none of these sets there is exactly one of the c shortest distance 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 shortest distance CAs at least once with the desired probability.

Next we show how to find all unique shortest distance CAs in a candidate set V'_Z. For all pairs $\{v_i, v_j\}$ such that there is exactly one shortest distance CA v_k in V'_Z, we can construct v_k deterministically as follows: For all $1 \leq \ell \leq \lceil \log n \rceil$, let I_ℓ be the set of all vertices z_k in V'_Z 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 $v \in V'_Z \cap I_\ell$, but with all other vertices from V_Z 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) shortest distance CA of v_i and v_j in G must have its ℓ-th bit equal to one, too. Hence all unique shortest distance CA in V'_Z 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 DISTANCES 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 [AN96] for more details). The derandomized approach takes $O((\log n)^6)$ ALL-PAIRS SHORTEST DISTANCES computations and the theorem follows. ■

Theorem 6.2 and the results in [Zwi02] imply the following corollary.

Corollary 6.3. *Let G be a dag with n vertices.* ALL-PAIRS SHORTEST DISTANCE CA *can be solved in time $\widetilde{O}(n^{2+\mu})$ if the edge weights are restricted to integers of constant absolute value.* ALL-PAIRS SHORTEST DISTANCE CA *can be solved in time $o(n^3)$, if the edge weights are restricted to integers of absolute value M such that $M < n^{3-\omega}$.*

From the above-mentioned, currently best bound for μ, we obtain an $O(n^{2.575})$ algorithm for ALL-PAIRS SHORTEST DISTANCE CA for constant integral edge weights.

Dynamic programming techniques can also be applied to solve weighted common ancestor problems. We extend the ideas developed in Chapter 3 to solve the ALL-PAIRS SHORTEST DISTANCE CA problem in time $O(nm)$. Recall again that we focus explicitly on computing shortest distance common ancestors instead of only computing shortest ancestral distances.

A naive solution is as follows:

1. Compute the all-pairs shortest distance matrix D of G.
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 [CLRS01] 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 $O(n^3)$. This can be improved to $O(nm)$ by applying dynamic programming. Observe, however, that it is in general not possible to use transitive reduction as a speed-up trick in common ancestor problems involving distances. We give an $O(nm)$ dynamic programming solution to ALL-PAIRS SHORTEST DISTANCE CA, which is similar to Algorithm 2. The following lemma describes the structure of the dynamic programming matrix.

Lemma 6.4. *Let $G = (V,E)$ be a weighted dag and let $\{x,y\}$ be a vertex pair such that $\mathrm{CA}\{x,y\} \neq \emptyset$. Furthermore, let $\{x_1,\ldots,x_l\}$ be the parents of x and let $Z = \{z_1,\ldots,z_l\}$ be the set of the corresponding shortest distance CAs of $\{x_i,y\}$ for $1 \leq i \leq l$. Then, for the shortest distance CA z of x and y it holds that $z = \mathrm{argmin}_{z' \in Z \cup \{x\}} \mathrm{AD}_{z'}\{x,y\}$.*

Proof. The proof is similar to the proofs of Lemmas 3.6 and 3.11. To ease exposition, we assume that the shortest distance CAs (SDCAs) are unique. Let z be the unique SDCA of $\{x,y\}$. Suppose for the sake of contradiction that $z \notin Z \cup \{x\}$. Consider a path $p = (z,\ldots,x)$ that minimizes the distance $\mathrm{d}(z,x)$. Since $z \neq x$, p has the form $z,\ldots,x_k,x$ for some $1 \leq k \leq l$ and x_k is a parent of x. Obviously $z \in \mathrm{CA}\{x_k,y\}$. Let $z' = \mathrm{SDCA}\{x_k,y\}$. We have

$$\begin{aligned} \mathrm{AD}_z\{x,y\} &= \mathrm{d}(x_k,x) + \underbrace{\mathrm{d}(z,x_k) + \mathrm{d}(z,y)}_{\mathrm{AD}_z\{x_k,y\}} \\ &< AD_{z'}\{x,y\} \\ &= \mathrm{d}(x_k,x) + \underbrace{\mathrm{d}(z',x_k) + \mathrm{d}(z',y)}_{\mathrm{AD}_{z'}\{x_k,y\}}, \end{aligned} \tag{6.1}$$

since z is the unique SDCA of $\{x,y\}$. On the other hand, Equation (6.1) implies

$$\mathrm{AD}_z\{x_k,y\} < \mathrm{AD}_{z'}\{x_k,y\}$$

contradicting $z' = \mathrm{SDCA}(x_k,y)$. ■

Algorithm 10: DP-Algorithm for Shortest Distance CA

Input: A dag $G = (V,E)$ with a weight function $w : E \to \mathbb{R}$.
Output: An array SDC of size $n \times n$ where $SDC[x,y]$ is a minimum-weight CA of $\{x,y\}$.

```
1  begin
2      Compute the all-pairs shortest-distance matrix D of G.
3      Compute a topological ordering top.
4      foreach {x,y} with D[x,y] < ∞ do
5          SDC[x,y] ← x
6      end
7      foreach x ∈ V in ascending order of top(v) do
8          foreach (z,x) ∈ E do
9              foreach y ∈ V do
10                 if AD_z{x,y} < SDC[x,y] then
11                     SDC[x,y] ← SDC[z,y]
12                 end
13             end
14         end
15     end
16 end
```

Theorem 6.5. *In a dag G with n vertices and m edges, the dynamic programming algorithm (Algorithm 10) solves* ALL-PAIRS SHORTEST DISTANCE CA *in time $O(nm)$.*

Proof. The analysis is similar to the proof of Theorem 3.7. The correctness follows from Lemma 6.4. Since vertices are visited in ascending topological order, the shortest distance CA of $\{z,y\}$ has already been determined at the first visit of x. Finally, the running time is clearly $O(nm)$, because Line 2 can be achieved in time $O(nm)$, as mentioned above. ■

We turn our attention to the respective problem in vertex-weighted dags, i.e., the ALL-PAIRS MINIMUM WEIGHT CA problem. As our main result we establish a computational equivalence between ALL-PAIRS MINIMUM WEIGHT CA and ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION below.

Observe that the relationship between ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION given by Proposition 4.3 is only one-sided, i.e., ALL-PAIRS REPRESENTATIVE LCA can be reduced to ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION, but the opposite direction is open. On the other hand, this reduction implicitly makes use of a vertex weight function, namely a topological ordering. That is, ALL-PAIRS REPRESENTATIVE LCA is in fact reduced to the problem of finding *maximum* CAs with respect to a weight function that implies that a maximum weight CA corresponds to an LCA.

Theorem 6.6. ALL-PAIRS MINIMUM WEIGHT CA *and* ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION *are computationally equivalent.*

Proof. We reduce the problem of finding minimum weight CAs to the problem of finding maximum weight CAs by simply negating the vertex weight function w_v.

1. ALL-PAIRS MINIMUM WEIGHT CA to ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION: This follows from Proposition 4.3. Order the vertices according to the negated vertex weight function and compute maximum witnesses for $A_{\text{clo}}^T \cdot A_{\text{clo}}$. Here, A_{clo} corresponds again to the adjacency matrix of the transitive closure of the respective dag.
2. ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION to ALL-PAIRS MINIMUM WEIGHT CA: Let $A \cdot B$ be the considered Boolean matrix product. Construct a dag $G = (X \cup Z \cup Y, E_X \cup E_Y)$ with vertex weight function w_v such that:
   - $X = \{x_1, \ldots, x_n\}$, $Z = \{z_1, \ldots, z_n\}$, and $Y = \{y_1, \ldots, y_n\}$.
   - $E_X = (z_i, x_j)$ for all $1 \leq i, j \leq n$ such that $A[j,i] = 1$.
   - $E_X = (z_i, y_j)$ for all $1 \leq i, j \leq n$ such that $B[i,j] = 1$.
   - $w_v(z_i) = -i$ for all $1 \leq i \leq n$ and $w_v(v) = 0$ for all other vertices.

   Then, the minimum weight CA of $\{x_i, y_j\}$ in G corresponds to the maximum witness of (i,j) with regard to the matrix product $A \cdot B$.

■

Corollary 6.7. *For a dag G with n vertices, there exists a dynamic programming algorithm for* ALL-PAIRS MINIMUM WEIGHT CA *with running time $O(n\,m_{\text{red}})$ and average case running time $O(n^2 \log n)$, where m_{red} is the number of edges in the transitive reduction of G.*

Proof. It is not difficult to verify that Algorithm 2 can be modified such that it computes minimum CAs with respect to an arbitrary vertex weight function in time $O(n\,m_{\text{red}})$ (Thm. 3.7), which is $O(n^2 \log n)$ in the average case. ■

6.2.2 Lowest Common Ancestor Problem Variants

While Theorems 6.2 and 6.6 give a fairly accurate complexity classification of the common ancestor problem variants, algorithmic solutions with respect to lowest common ancestors, i.e., for ALL-PAIRS SHORTEST DISTANCE LCA and ALL-PAIRS MINIMUM WEIGHT LCA, seem to be far more difficult. In particular, the best upper time bounds for the LCA variants both in edge- and vertex-weighted dags rely on solving ALL-PAIRS ALL LCA. This in turn bounds the complexity in the worst case by $\Omega(n^3)$ from below, see Theorem 2.12. Furthermore, the best upper bound for ALL-PAIRS ALL LCA is $O(n^{\omega(2,1,1)})$ by Theorem 4.19 although we have identified benign input classes. It is not clear whether the solutions to one or both of the problems studied below can be solved without solving ALL-PAIRS ALL LCA. We continue by describing the solutions leading to the general upper time bounds.

A generic solution to finding shortest distance lowest common ancestors can be described as follows:

1. Solve ALL-PAIRS SHORTEST DISTANCES 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 $z \in \mathrm{LCA}\{x,y\}$ that minimizes the ancestral distance $d_G(z,x)+d_G(z,y)$ and set $SDL[x,y]=z$.

From solutions to ALL-PAIRS SHORTEST DISTANCES and ALL-PAIRS ALL LCA, the shortest distance LCAs can be readily derived in time $O\big(\sum_{x,y\in V}\ |\mathrm{LCA}\{x,y\}|\big)$. The following theorem is an immediate consequence of the generic solution given above.

Theorem 6.8. *Let G be a dag with n vertices and m edges. Let $\mathscr{A}$ be an algorithm that solves* ALL-PAIRS ALL LCA *in time $\mathscr{T}_{\mathscr{A}}(n,m)$. Let $\mathscr{B}$ be an algorithm that solves* ALL-PAIRS SHORTEST DISTANCES *in time $\mathscr{T}_{\mathscr{B}}(n,m)$. Then,* ALL-PAIRS SHORTEST DISTANCE LCA *can be solved in time $O(\mathscr{T}_{\mathscr{A}}(n,m)+\mathscr{T}_{\mathscr{B}}(n,m))$.*

Recall that in the worst case ALL-PAIRS ALL LCA is lower bounded by $\Omega(n^3)$, see Theorem 2.12. On the other hand, ALL-PAIRS SHORTEST DISTANCES can be solved in time $O(nm)$ in dags with arbitrary edge weights [CLRS01]. Hence, in the worst case the complexity of the ALL-PAIRS ALL LCA algorithm subsumes the complexity of the ALL-PAIRS SHORTEST DISTANCES algorithm. Nevertheless, there are classes of dags for which ALL-PAIRS ALL LCA can be solved more efficiently. For example, consider dense dags with arbitrary (large) real-valued edge weights with a sparse transitive reduction and unique LCAs. In such dags the ALL-PAIRS SHORTEST DISTANCES computation becomes the bottleneck. For dags with integral edge weights of constant absolute value we get the following corollary.

Corollary 6.9. *In an edge-weighted dag G with n vertices and m edges such that the edge weights are integers of absolute value bounded by a constant* ALL-PAIRS SHORTEST DISTANCE LCA *can be solved in time $O(\min\{n^2m, nm(\kappa^2+\kappa\log n\}) + n^{2+\mu})$, where κ is the maximum cardinality of all LCA sets.*

The next theorem establishes the general upper bound for ALL-PAIRS MINIMUM WEIGHT LCA and is straightforward. Since we do not have to solve ALL-PAIRS SHORTEST DISTANCES, upper bounds for ALL-PAIRS ALL LCA directly transfer to ALL-PAIRS MINIMUM WEIGHT LCA.

Theorem 6.10. ALL-PAIRS MINIMUM WEIGHT LCA *in a dag G can be reduced to* ALL-PAIRS ALL LCA *in G.*

In particular, Theorems 6.8 and 6.10 imply that the current best possible upper bound for ALL-PAIRS SHORTEST DISTANCE LCA and ALL-PAIRS MINIMUM WEIGHT LCA in full generality is $O(n^{\omega(2,1,1)})$. It is an interesting open question whether this can be improved or proved to be optimal.

While the general upper time bound for ALL-PAIRS MINIMUM WEIGHT LCA relies on the solution of ALL-PAIRS ALL LCA, we can identify classes of distance functions that allow to solve ALL-PAIRS MINIMUM WEIGHT LCA directly, thereby circumventing the $\Omega(n^3)$ lower bound for ALL-PAIRS ALL LCA.

To ease exposition we consider the ALL-PAIRS MAXIMUM WEIGHT LCA problem instead of the ALL-PAIRS MINIMUM WEIGHT LCA problem in the following. Observe that these two problems are in one-to-one correspondence. It is sufficient to negate the vertex weights in order to switch from one problem to the other.

Let now $w_v : V \to \mathbb{R}$ be an arbitrary vertex weight function and let $ord : V \to \{1,\ldots,n\}$ be a function that orders the vertices with respect to w_v. That is, $ord(u) \leq ord(v)$ implies $w_v(u) \leq w_v(v)$ for all $u,v \in V$.

Obviously, if *ord* corresponds to a valid topological order of V, ALL-PAIRS MAXIMUM WEIGHT LCA reduces again to ALL-PAIRS MAXIMUM WITNESSES FOR BOOLEAN MATRIX MULTIPLICATION and can hence be solved in time $O(n^{2+\mu})$. However, even if *ord* is not a valid topological order, but only *close* to one, we can derive substantially better upper bounds than in the general case.

Theorem 6.11. *Let G be a vertex-weighted dag with n vertices. Let the vertex weight function w be such that $ord(u) \leq ord(v) + f(n)$ for all $u,v \in V$ such that u is a predecessor of v. Here, f is an arbitrary positive function of n. Then,* ALL-PAIRS MAXIMUM WEIGHT LCA *can be solved in time $O\left(n^{2+\mu} + f(n)^2 n^2\right)$.*

Proof. We start by solving ALL-PAIRS MAXIMUM WEIGHT CA on G with respect to *ord* in time $O(n^{2+\mu})$. Let now z_c be the maximum weight CA of a pair $\{x,y\}$ and let z_l be the maximum weight LCA of $\{x,y\}$.

Claim 6.12. $ord(z_l) \in \{ord(z_c) - f(n), \ldots, ord(z_c)\}$.

Proof. For $z_c = z_l$ the claim readily follows. If $z_c \neq z_l$, we have $z_c \notin \mathrm{LCA}\{x,y\}$ and hence there exists a witness w for z_c such that $w \in \mathrm{LCA}\{x,y\}$ (Obs. 2.9) and w is a successor of z_c. This implies $ord(z_c) \leq ord(w) + f(n)$ and hence $ord(w) \geq ord(z_c) - f(n)$. Since w is an LCA, we get

$$ord(z_c) \geq ord(z_l) \geq ord(w) \geq ord(z_c) - f(n)$$

which concludes the proof of the claim. ■

Let now $Z \subseteq V$ be a vertex subset such that $\bigcup_{z \in Z} ord(z) = \{ord(z_c) - f(n), \ldots, ord(z_c)\}$. Observe that Z can be constructed in linear time assuming we have a reverse mapping from $ord(v)$ to v for all $v \in V$. By the above claim $z_l \in Z$. As a consequence of the vertex weight property we can determine for each vertex in $z \in Z$ whether it is an LCA of $\{x,y\}$ or not in $O(f(n))$ time. Recall that the maximum order of a possible witness for z is less than $ord(z_c)$ but greater or equal than $ord(z) - f(n)$. ■

Observe that, e.g., for $f(n) = O(n^{\mu/2})$, ALL-PAIRS MAXIMUM WEIGHT LCA can be solved in $O(n^{2+\mu})$ and in subcubic time for $f(n) = o(n^{0.5})$.

Again, we stress the fact that the bound for ALL-PAIRS MAXIMUM WEIGHT LCA also applies to ALL-PAIRS MINIMUM WEIGHT LCA. Note that weight functions that are close to topological orderings appear in situations where the vertex weights are related to causal dependencies modeled by the dag. Consider for example an assignment of costs to providers in the Internet dag. Obviously, one expects decreasing costs with decreasing level, i.e., the costs of providers on the lowest level are expected to be minimal. However, the data used might be faulty or contradicting such that it might make sense to drop strict topological requirements. On the other hand, the assumption $ord(u) \leq ord(v) + f(n)$ for $u \rightsquigarrow v$ is more restrictive than, for example, assuming $w_v(u) \leq w_v(v) + f(n)$. Observe that in the latter case there might be a linear fraction of the vertices with weights between $w_v(v)$ and $w_v(u)$. In the first case we assume that only $f(n)$ vertices have weights in between, hence we implicitly assume that the vertex weights are nicely distributed in some sense.

Note that it can be checked in time $O(n^2)$ if a given vertex weight function is *good* in the sense of Theorem 6.11 if the transitive closure of the input graph is known and the vertex weights are unique. Simply fix the unique ordering and compute $\max_{(u,v)\in E_{\text{clo}}} ord(u) - ord(v)$.

6.3 Dynamic Algorithms

The consideration of graph algorithms in a dynamic context is motivated by many applications such as communication networks, graphical applications, information systems, assembly planning, or VLSI design. In these applications graphs are subject to changes like insertions or deletions of vertices and edges. The design of specialized solutions for dynamic situations aims at preventing to recompute the problem of interest from scratch. Interest in dynamic graph algorithms has been growing tremendously in the last two decades. Progress has been made for many classical problems such as transitive closure [San04, DI05] or all-pairs shortest paths [DI04, Tho04]. In the context of lowest common ancestor computations, Cole and Hariharan [CH05] have recently obtained an improved dynamic solution for the special case of trees. The results in this section are a first step in establishing dynamic solutions for the general case of acyclic digraphs. We describe the following results with regard to a dag G with n vertices, which improve over naive recomputation:

- The update time for ALL-PAIRS REPRESENTATIVE LCA can be bounded by $O(n^{2.5})$ under vertex-centered updates. Using a similar approach the result can be extended to $O(n^3)$ update time for ALL-PAIRS ALL LCA. We note that the result for ALL-PAIRS REPRESENTATIVE LCA requires preprocessing of $O(n^{2.876})$. The main technique used to achieve this result is a reduction from the rectangular matrix products to transitive closure computations.
- For an incremental version of ALL-PAIRS REPRESENTATIVE LCA, i.e., the edges of the dag are sequentially inserted, we obtain an amortized update complexity of $\widetilde{O}(\min\{n^4/m, n^{2.5}\})$. This improves over the fully dynamic case for dense dags with $m = \omega(n^{1.5})$. This result is achieved by a non-trivial combination of an earlier solution [BFCP$^+$05] to ALL-PAIRS REPRESENTATIVE LCA based on shortest path computations with dynamic ALL-PAIRS SHORTEST DISTANCES algorithms [Tho04] and an online topological order algorithm [PK06].

Throughout this chapter we assume that no update operation in a dag G causes a cycle.

6.3.1 Fully Dynamic Lowest Common Ancestors

We start by considering LCA problems in a *fully dynamic* setting, i.e., update operations involve deletions and insertions. In contrast to that, a problem is called *partially dynamic* if only one type of operation is allowed. An algorithm that handles insertions only is usually called *incremental*, whereas *decremental* algorithms are restricted to deletions. We consider an incremental problem in Section 6.3.2. Throughout this section, we focus on update operations with respect to the edge set. That is, the set V is fixed, but in every step, a subset of edges might be deleted from or added to the edge set E of a dag $G = (V,E)$. We consider *vertex-centered* updates.

Definition 6.13 (Vertex-Centered Update)**.** *Let $G = (V,E)$ be a dag and let $E^u = E^+ \cup E^-$ be the set of edges to be updated in an update operation. Here, E^+ corresponds to a set of edges to be added to the dag, and E^- corresponds to a set of edges to be deleted. The update operation is called vertex-centered if there exists a vertex $v \in V$ such that every edge in E^u is incident to v.*

Let A be the adjacency matrix of $G = (V,E)$. Observe that a vertex-centered update with respect to a vertex v updates exactly the row and column with index v in A. On the other hand, the addition or deletion of a single edge can change up to $\Theta(n^2)$ entries in the adjacency matrix A_{clo} of G_{clo}.

Recall the matrix-multiplication-based solutions to ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA in the static case described in Chapter 4. For both problems we compute matrix products in order to answer questions of the following type: Is there a vertex $z \in S$ that is a common ancestor of a pair $\{x,y\}$? Here, $S \subseteq V$ is some subset of the vertex set. In the case of the ALL-PAIRS REPRESENTATIVE LCA problem, we use answers to these questions in order to determine the subset of vertices that contain the maximum CA. In the ALL-PAIRS ALL LCA solution, the products correspond to witness existence queries. Naturally, extending these approaches to a dynamic setting we need to maintain the matrix products under row- and column-updates of the adjacency matrix A of G. However, the products involve A_{clo} rather than A so that the updates are not restricted to a single row and column.

In the following, we slightly extend our notion of matrix sampling given in Definition 4.6. Let M be an $n \times n$ matrix and let $I \subseteq \{1,\dots,n\}$. We denote by $M[*,I,0]$ an $n \times n$ matrix in which the columns whose indices belong to I correspond to the columns with the same index in M and all other columns correspond to 0-vectors. We call $M[*,I,0]$ a *column-submatrix* of M. $M[I,*,0]$ is called a *row-submatrix* of M and is defined analogously, see Figure 6.3.

M

1	0	1	0
0	0	1	1
0	1	0	0
1	1	1	1

$M[*,I]$

1	1
0	1
0	0
1	1

$M[*,I,0]$

1	0	1	0
0	0	1	0
0	0	0	0
1	0	1	0

Figure 6.3: Example of extended matrix sampling with regard to a matrix M and $I = \{1,3\}$.

Observe that all matrices involved in the matrix multiplication steps of the solutions to ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA can be viewed as column-submatrices of A_{clo}^T

or row-submatrices of A_{clo}. The following lemma is essential for obtaining non-trivial solutions for all-pairs LCA problems under vertex-centered updates.

Lemma 6.14. *Let A_{clo} be the adjacency matrix of the transitive closure G_{clo} of a dag $G = (V,E)$ with n vertices. Next, let A be a column-submatrix of A_{clo}^T and B be a row-submatrix of A_{clo} and let $C = A \cdot B$. Then, the computation of C can be reduced to the computation of the transitive closure G'_{clo} of a dag $G' = (V',E')$ with the following properties:*

(i) The number of vertices n' of G' is linear in n, more specifically, $n' = |V'| = 3n$.

(ii) The result of $A \cdot B$ can be directly read off the adjacency matrix A_{clo} of G'.

(iii) A vertex-centered update operation in G incurs at most 2 column-updates and at most 2 row-updates in G'.

Proof. We start by describing the reduction of the matrix multiplication $A \cdot B$ to the transitive closure computation in a corresponding dag G'. Let in the following $G = (V,E)$ where $V = \{v_1, \ldots, v_n\}$. By assumption, A is a column-submatrix of A_{clo}^T and B is a row-submatrix of A_{clo}. That is, there exist vertex-subsets $V_A \in V$ and $V_B \in V$ such that $A = A_{\text{clo}}^T[*, V_A, 0]$ and $B = A_{\text{clo}}[V_B, *, 0]$. We construct $G' = (V',E')$ as follows:

- $V' = X \cup W \cup Y$, where $X = \{x_1, \ldots, x_n\}$, $W = \{w_1, \ldots, w_n\}$, and $Y = \{y_1, \ldots, y_n\}$ are vertex sets of cardinality n.
- $E' = E_X \cup E_Y \cup E_A \cup E_B$ where
  - $E_X = (x_i, x_j)$ for all $1 \leq i, j \leq n$ such that $(v_j, v_i) \in E$.
  - $E_Y = (y_i, y_j)$ for all $1 \leq i, j \leq n$ such that $(v_i, v_j) \in E$.
  - $E_A = (x_i, w_i)$ for all $1 \leq i \leq n$ such that $v_i \in V_A$.
  - $E_B = (w_i, y_i)$ for all $1 \leq i \leq n$ such that $v_i \in V_B$.

This construction is similar to constructions used in earlier chapters, e.g., the construction of the graph used for the reduction of ALL-PAIRS COMMON ANCESTOR EXISTENCE to BOOLEAN MATRIX MULTIPLICATION, see Section 4. The difference here is that we restrict connections between the two parts of the graph to vertices in the sets V_A and V_B (actually $V_A \cap V_B$).

We get the following claim.

Claim 6.15. *$C[i,j] = 1$ if and only if y_j is reachable from x_i in G'.*

Proof. Indeed, observe that every path from a vertex x_i to y_j contains a vertex w_k. The following is easy to check.

1. $v_k \in V_A$ and $v_k \in V_B$.
2. (v_k, v_i) and (v_k, v_j) are both edges in G_{clo}.

On the other hand, $C[i,j] = 1$ if and only if there exists an index k such that $v_k \in V_A$, $v_k \in V_B$ and $A[i,k] \cdot B[k,j] = 1$. It is not difficult to check that this implies $x_i \rightsquigarrow y_j$ in G'. ■

This establishes property (ii) of the reduction. Observe that $C = A_{\text{clo}}[X,Y]$. Property (iii), i.e., a vertex-centered update in G incurs only 2 vertex centered updates in G', also follows from the

construction of G'. The edges in G' are either edges in G, reverse edges in G, or connecting edges. Hence, one vertex-centered update in G incurs two vertex-centered updates in G'. ■

The following is due to Sankowski [San04].

Proposition 6.16. *Let G be a directed acyclic graph with n vertices. Let A_{clo} be the adjacency matrix of G_{clo}. Then, A_{clo} can be maintained under vertex-centered updates with $O(n^{\omega})$ initialization and $O(n^2)$ update time.*

Lemma 6.14 and Proposition 6.16 yield the following theorem.

Theorem 6.17. *For a dag G with n vertices, there exists an algorithm for dynamic* ALL-PAIRS REPRESENTATIVE LCA *under vertex-centered updates with $O(n^{1-\lambda+\omega}+n^{2+\lambda})$ initialization, $O(n^{1-\lambda+2}+n^{2+\lambda})$ update, and $O(1)$ query time, for some $\lambda \in [0;1]$.*

Proof. Recall that for the static ALL-PAIRS REPRESENTATIVE LCA solution we divide up the vertices into $n^{1-\lambda}$ sets of size n^{λ} for some $\lambda \in [0;1]$. To ease exposition, we assume again in the sequel that n^{λ} and $n^{1-\lambda}$ are integers. For each vertex set V^i, $1 \leq i \leq n^{1-\lambda}$, one rectangular matrix product of an $n \times n^{\lambda}$ and an $n^{\lambda} \times n$ matrix is computed in order to identify all pairs for which a CA in the set V^i exists. The maximum CAs are then searched for in the vertex set with the largest index i. In order to improve the upper time bound for vertex-centered updates, we use the reduction from rectangular matrix product to transitive closure described in Lemma 6.14 for each of the matrix products.

We analyze the complexity of this approach. The initialization needs time $O(n^{1-\lambda+\omega}+n^{2+\lambda})$. The time needed to compute the transitive closures of the graphs $G^{(1)},\ldots,G^{(n^{1-\lambda})}$ corresponds to the time needed to compute the corresponding rectangular matrix products, i.e., $O((n^{1-\lambda+\omega(1,\lambda,1)})$. Additionally, however, we have to initialize the dynamic data structure (matrix inverse) for each transitive closure. The best known time bound for this is $O(n^{\omega})$, see Proposition 6.16. Deriving the maximum common ancestors for all pairs is again in $O(n^{2+\lambda})$. An optimal upper time bound for the initialization is obtained by choosing λ such that $1-\lambda+\omega = 2+\lambda$ is satisfied, which implies $\lambda < 0.688$.

The update complexity, on the other hand, is $O(n^{1-\lambda+2}+n^{2+\lambda})$ due to Proposition 6.16. Hence, we get a dynamic algorithm with $O(n^{1-\lambda+\omega}+n^{2+\lambda})$ initialization, $O(n^{1-\lambda+2}+n^{2+\lambda})$ update, and $O(1)$ query time. ■

Optimizing λ for updates yields $\lambda = 0.5$. The usage of transitive closures causes a slight overhead for the initialization, but improves the time bound for updates in the dynamic case.

Corollary 6.18. *There exists an algorithm for dynamic* ALL-PAIRS REPRESENTATIVE LCA *on a dag G with n vertices with $O(n^{2.876})$ initialization, $O(n^{2.5})$ update, and $O(1)$ query time.*

We apply the same technique to the ALL-PAIRS ALL LCA problem to achieve the next theorem.

Theorem 6.19. *There exists an algorithm for dynamic* ALL-PAIRS ALL LCA *on a dag G with n vertices with $O(n^{\omega+1})$ initialization, $O(n^3)$ update, and $O(1)$ query time.*

Proof. We reduce the n matrix products that are used to compute the witness matrices by dags $G^{(1)},\ldots,G^{(n)}$ according to the reduction described in Lemma 6.14. Since the common ancestor matrices $C^{(v)}$ can be computed in time $O\left(n^3\right)$ for all $v \in V$, the bounds follow from Proposition 6.16. ■

Observe that $O\left(n^3\right)$ update time is optimal in the worst case, since a single update can change up to $\Theta(n^3)$ entries.

6.3.2 Incremental LCA Algorithm

We proceed by considering an incremental LCA variant, which we also refer to as *online representative LCA* problem. That is, we are given a dag with vertex set V and an initially empty edge set E and incrementally add edges to the dag. The goal is to maintain a representative LCA matrix L after each edge addition. Using the dynamic solution presented in the previous section, L can clearly be maintained with $O(n^{2.5})$ update complexity. However, we show below that the amortized update complexity can be improved whenever the number of inserted edges is large. For example, for $m = \Theta(n^2)$, the amortized update complexity is only $O(n^2)$. The main motivation for online LCA problem variants stems from incremental compilation. Recall that in Section 1.1 we have described an application of LCA algorithms which is related to compilation and execution of object-oriented programming languages. A lot of modern compilers use incremental compilation, e.g., the *Eclipse* Java compiler. Roughly speaking, an incremental compiler maintains a dependency graph between modules to reduce the amount of needed recomputation work when updates occur or to enable fast compilation at the runtime level. Such a setting is readily modeled by considering respective computational problems incrementally. We note that incremental compilation is also the main motivation for the related *online topological order* problem, see below.

Before we describe our approach we review related work and introduce the concepts needed.

Maximum CA by Shortest Paths

There is a close relationship between finding arbitrary CAs and reachability on the one hand, and finding maximum CAs and shortest paths on the other hand. This was first noticed and used by Bender et al. [BFCP+05]. The reduction from finding arbitrary CAs to reachability has been already described in the context of ALL-PAIRS COMMON ANCESTOR EXISTENCE. Recall again the construction of the graph G' in the context of the reduction from ALL-PAIRS COMMON ANCESTOR EXISTENCE to BOOLEAN MATRIX MULTIPLICATION in Section 4.1 (Fig.4.1). We augment G' with the following edge-weight function $w_e : E' \to \mathbb{N}_0$.

$$w_e(e) = \begin{cases} 0 & \text{if } e \in E_X \cup E_Y \\ n-i & \text{if } e \in E'' \text{ and } e = (x_i, y_i) \end{cases}$$

We have the following immediate relationship between maximum CAs in $G = (V,E)$ and shortest distances in G' derived from G. For the lemma, we assume again that the vertex set $V = \{v_1,\ldots,v_n\}$ is in ascending order with respect to a topological ordering.

Lemma 6.20. *For a pair of vertices $\{v_i, v_j\}$ in a dag G, v_k is the maximum CA of $\{v_i, v_j\}$ in G if and only if $d_{G'}(x_i, y_j) = k$.*

Bender et al. [BFCP+05] used this relationship in order to develop the first truly subcubic ALL-PAIRS MAXIMUM CA algorithm. Since no $o(n^3)$ algorithm for ALL-PAIRS SHORTEST DISTANCES in digraphs with large (here, up to n) integer weights is known, they compute approximate shortest paths using an algorithm given by Zwick [Zwi98]. Zwick's algorithm computes shortest distances within a factor of $1+\varepsilon$ in time $\widetilde{O}((n^\omega/\varepsilon)\log(W/\varepsilon))$, where W is an upper bound for the largest edge weight. The approximate shortest distances narrow down the number of candidates for each pair to $O(\varepsilon n)$. Ignoring the polylogarithmic terms and equating $n^\omega/\varepsilon = n^3\varepsilon$ yields $\varepsilon = n^{\frac{\omega-3}{2}}$ and hence an upper time bound of $\widetilde{O}(n^{\frac{\omega+3}{2}})$.

Dynamic All-Pairs Shortest Distance

In the *dynamic all-pairs shortest distance* problem one is interested in maintaining a matrix storing the shortest distances between each vertex pair under update operations. The following proposition is due to Thorup [Tho04].

Proposition 6.21. *The dynamic all-pairs shortest distance problem under vertex-centered updates can be solved with amortized update time $O(n^2(\log\log n + \log n \log\log n))$ if the edge weights are integers.*

The above bound is based on earlier work by Demetrescu and Italiano [DI04] improving their result by logarithmic factors. We note that Thorup considers *vertex updates*, i.e., a vertex with incident edges is inserted or deleted in each update step. However, it is easy to see that vertex-centered updates correspond to at most 2 vertex updates: Simply remove the respective vertex and add a new vertex whose incident edges correspond to the result of the vertex-centered update.

Online Topological Ordering

The *online topological order* problem, i.e., maintaining a valid topological order under a sequence of edge insertions, has attained interest recently, [AHR+90, MSNR96, KB05, PK06, AFM06, AF07]. Achievements in this field have provided upper time bounds of $O(\min\{m^{1.5}\log n, m^{1.5}+n^2\log n\})$ [KB05] and $O(n^{2.75})$ [AFM06] for the insertion of m edges in an initially empty graph; the second algorithm outperforms the first on dense dags. The algorithm given by Pearce and Kelly [PK06] (PK) outperforms the other approaches empirically under sparse random edge insertions; recently an average case running time of $O(n^2\log^2 n)$ was shown for this algorithm in [AF07]. We summarize properties of the PK approach that are needed for our online LCA solution.

Proposition 6.22. *Let $G=(V,E)$ be a dag with n vertices and initially empty edge set E in which m edges are inserted incrementally. Suppose further that a valid topological sort is maintained after each insertion by using the PK algorithm. Then, the following holds:*

(i) The total running time on any sequence of m edge insertions is bounded by $O(n^3)$.

(ii) Let δ_i denote the number of vertices that are assigned a new topological number after the insertion of the ith edge. Then

$$\sum_{i=1}^{m} \delta_i = O(n^2). \tag{6.2}$$

We further note that the PK algorithm recognizes edges that close a cycle in G.

Online Representative LCA

In order to find a non-trivial upper time bound for maintaining a representative LCA matrix online, we now combine Lemma 6.20, Proposition 6.21, and Proposition 6.22.

Obviously, by Lemma 6.20, if we are able to maintain an all-pairs shortest distance matrix for G' after each edge insertion, we can read off the corresponding maximum CAs, i.e., representative LCAs. Observe however, that the correctness of Lemma 6.20 is crucially depending on the fact that the vertices $\{v_1,\ldots,v_n\}$ are topologically ordered. Now, edge insertions can invalidate previously used topological orders. To this end we use the online topological order algorithm by Pearce and Kelly [PK06], i.e., Proposition 6.22. That is, we assume that we have a black box giving us a valid topological ordering $top^{(i)}$ after the insertion of the ith edge for all $1 \leq i \leq m$. Now, let again δ_i denote the number of vertices whose topological number is changed by the insertion of edge i.

Observation 6.23. *The insertion of the ith edge leads to* 2 *edge insertions and δ_i edge-weight updates in G'.*

This is immediate from the construction of G' and the definition of δ_i. Now combining this observation with Lemma 6.20, Proposition 6.22, and Theorem 6.17 leads to the following theorem, which states a non-trivial update time bound for the online representative LCA problem.

Theorem 6.24. *The amortized update time of the online representative lowest common ancestor problem in a graph G with n vertices under m edge insertions can be bounded by*

$$\widetilde{O}(\min\{n^4/m, n^{2.5}\}).$$

Proof. We describe the respective algorithm. To simplify matters we suppose in the following that $n^{1.5}$ is an integer. We use the online topological order solution by Pearce and Kelly [PK06] in order to update the topological order after each edge insertion. For the update of the representative LCA matrix we use the result from Theorem 6.17 for the first $n^{1.5}$ edges. From the $(n^{1.5}+1)$st edge onward, we use a refined approach. We start by constructing the graph G' and computed an ALL-PAIRS SHORTEST DISTANCES matrix in time $O(n^3)$. Then, after each edge insertion, we update the edge weights in G' according to the new topological order and recalculate the ALL-PAIRS SHORTEST DISTANCES matrix using the approach of [Tho04]. Observe that the time needed for this step is bounded by $\widetilde{O}(n^2\delta_i)$. The LCA matrix can be updated in time $O(n^2)$ with knowledge of the ALL-PAIRS SHORTEST DISTANCES matrix. In order to analyze the amortized update time on m edge insertions, we distinguish two cases:

- $m \leq n^{1.5}$: The update cost per edge insertions is trivially bounded by $O(n^{2.5})$ in this case.
- $m > n^{1.5}$: The total update complexity $\mathscr{T}$ can be bounded as follows:

$$\begin{aligned}\mathscr{T} &\leq O(n^{1.5}\cdot n^{2.5}) + O(n^3) + \sum_{i=1}^{n} O(n^2(\log\log n + \log n \log\log n)\delta_i) \\ &= O(n^4(\log\log n + \log n\log\log n))\end{aligned}$$

Since $n^4/m = O(n^{2.5})$ for $m > n^{1.5}$, the theorem follows.

■

6.4 Space-Efficient LCA Computations

With respect to possible applications, the consideration of "all-pairs" LCA problems is fairly specific. Often, one is not interested in LCA information associated with every possible vertex pair in a dag. Rather, LCAs are only queried for a subset of the vertex pairs. Precomputing the answers to all possible queries and storing these answers in a matrix is brute force in this context. Considering the huge memory requirements of this approach, it is desirable to design more space-efficient solutions. More specifically, we give a generalized formulation of LCA problems as follows: Preprocess G such that a query of the form LCA$\{x,y\}$ can be answered as quickly as possible. The query LCA$\{x,y\}$ can be any variant of LCAs in dags, e.g., representative, all, or minimum weight. The quality of solutions to the above problems is measured in terms of preprocessing time, space requirements, and query time. We refer to these parameters – in dependence of the number of vertices n – by a triple $\langle p(n), s(n), q(n)\rangle$, i.e., using an ALL-PAIRS REPRESENTATIVE LCA solution we get $\langle O(n^{2.575}), O(n^2), O(1)\rangle$. This point of view is naturally taken in the context of LCA queries in trees, i.e., LCAs in trees can be solved in $\langle O(n), O(n), O(1)\rangle$ [BFCP$^+$05].

In the following we concern ourselves solely with the representative LCA problem. Observe however, that the results can be generalized to the all LCA problem. Our goal is to study data structures with subquadratic space requirement, i.e., $s(n) = o(n^2)$. A lot of work has been devoted to the space-efficient encoding of posets, see, for example, [Fal98] and the references therein. Here, one is interested in encoding a partial order in a space-efficient way such that reachability queries, i.e., is vertex v reachable from vertex u, can be answered quickly. For general posets there is – to the best of our knowledge – no solution that achieves $o(n^2)$ space and $o(n)$ query time in the worst case. Since reachability queries can be answered via representative LCA queries [BFCP$^+$05], the lack of a solution to the general poset encoding problem imposes a possible theoretical obstacle to space-efficient and quick LCA solutions.

In this section we give two natural approaches, which yield substantially space-efficient solutions on certain classes of dags. We analyze the according parameters, i.e., again, the width $\mathrm{w}(G)$ of G and the number of edges m_{clo} in the transitive closure of G on random dags in the $G_{n,p}$ model showing that for every choice of p there exists a solution with space requirement $o(n^2)$ and query time $o(n)$ in the average case. Recall that we consider average case running times and space requirements under the assumption that the input space is distributed according to the $G_{n,p}$ model.

For the following theorem we apply the path cover technique described in Chapter 5 in a straightforward manner. Let $\mathscr{T}_{\mathrm{PC}}(G)$ be the time needed to construct a minimum path cover for a dag G with n vertices and m edges. Then, the representative LCA problem in dags can be solved within $\langle O(\mathscr{T}_{\mathrm{PC}}(G) + \mathrm{w}(G)m), O(\mathrm{w}(G)n), O(\mathrm{w}(G))\rangle$. Using results from Section 5.4 we get the following theorem.

Theorem 6.25. *The representative LCA problem on a dag G with n vertices and width $\mathrm{w}(G)$ can be solved with $O(n^2\mathrm{w}(G)\log n)$ preprocessing time, $O(n\mathrm{w}(G)\log n)$ space, and $O(\mathrm{w}(G)\log n)$ query time.*

Note, however, that $\mathrm{w}(G) = \Theta(n)$ in the worst case. In what follows, let G be dag in the $G_{n,p}$ model. Let $p = f(n)/n$. The width is in expectation benign for dense dags, e.g., for $f(n) = \Theta(n)$ and $0 < p < 1$ is a constant.

Lemma 6.26. *Let G be random dag in the $G_{n,p}$ model such that $p = f(n)/n$. Then,*

$$\mathbb{E}[w(G)] = \begin{cases} o(n) & \text{if } f(n) = \omega(1) \\ O(n) & \text{else.} \end{cases}$$

Proof. Simon [Sim88] has shown that there exists an algorithm that produces a (not necessarily minimum) path cover of G of size k such that $\mathbb{E}[k] = O(\frac{\log(pn)}{p})$. The width $\mathrm{w}(G)$ is trivially upper bounded by k. For $p = f(n)/n$ we get $\frac{\log(pn)}{p} = \frac{n\log(f(n))}{f(n)}$. From this we can conclude that $\mathbb{E}[w(g)] = o(n)$ for $f(n) = \omega(1)$. Since $\mathrm{w}(G) \leq n$, the lemma follows. ■

The above Lemma along with the results of the previous sections imply the following corollary.

Corollary 6.27. *Let G be a random dag in the $G_{n,p}$ model such that $p = f(n)/n$ and $f(n) = \omega(1)$. Then, there exists a solution to the representative LCA problem in dags with average case complexity $\langle o(n^3), o(n^2), o(n)\rangle$.*

We continue by giving a simple solution for sparse dags, in particular dags having a small transitive closure. Suppose we have a sorted predecessor list L_v stored with each node v of G. Then, upon a query $\mathrm{LCA}\{x,y\}$, a representative LCA of x and y can be found by simply searching the vertex with the greatest topological number that is included in both L_x and L_y. This step obviously takes time $O(|L_x| + |L_y|)$ since the lists are sorted. The lists can be created as follows. For each vertex $v \in V$, perform a DFS traversal on the reverse dag $\overline{G} = (V, \overline{E})$ starting at v. This takes a total of $O(nm)$. Sorting the list L_v costs $O(|L_v| \log |L_v|)$. Using this easy approach, we get.

Theorem 6.28. *Representative LCA in a dag G can be solved within $\langle O(nm + \sum_{v\in V} |L_v| \log |L_v|), O(\sum_{v\in V} |L_v|), O(|L_x| + |L_y|)\rangle$ for a query* $\mathrm{LCA}\{x,y\}$.

In the worst case the above bounds are $\langle O(n^3), O(n^2), O(n)\rangle$, which does not improve over naive solutions. However, again it is possible to show that if G is a random dag in the $G_{n,p}$ model the expected value of $s(n)$ is $o(n^2)$ for certain choices of the parameter p.

We note that the space requirement of the above data structure, $\sum_{v\in V} |L_v|$, is proportional to the size of the transitive closure of G, i.e., the number of 1s in the adjacency matrix of G_{clo}. Let in the following $\gamma(s)$ be the transitive, reflexive closure of vertex $s \in V$, where s is the vertex with topological number 1 in G. The following lemma can be derived from results presented in [SCC93].

Lemma 6.29. *Let G be a random dag in the $G_{n,p}$ model. Then,*

$$\mathbb{E}[|\gamma(s)|] = \begin{cases} o(n) & \text{if } f(n) = o(\log n) \\ \Theta(n) & \text{else.} \end{cases}$$

Proof. The quantity $|\gamma(s)|$ was analyzed in random dags in the $G_{n,p}$ model in [SCC93]. It is shown that

$$\lim_{n\to\infty} n - \mathbb{E}[|\gamma(s)|] = L(q), \tag{6.3}$$

where $L(q) = \sum_{i=1}^{\infty} \frac{q^i}{1-q^i}$ is the so-called Lambert series with parameter q and $q = 1 - p$. The Lambert series cannot be analytically resolved, but again in [SCC93], an approximation for the expected value

of interest is given. Let $g(n)$ be a function that approximates $\mathbb{E}[|\gamma(s)|]$ such that $\mathbb{E}[|\gamma(s)|] = O(g(n))$. Then,

$$g(n) < -\frac{\log(1-\frac{\log q}{q^n})}{\log q} + \frac{\log(1-2\log q)}{2\log q}\exp\left(\frac{1-q^n}{\log q}\right). \tag{6.4}$$

We consider

$$\lim_{n\to\infty} g(n) \\ < \lim_{n\to\infty} -\frac{\log(1-\frac{\log q}{q^n})}{\log q} + \frac{\log(1-2\log q)}{2\log q}\exp\left(\frac{1-q^n}{\log q}\right).$$

for $q = 1 - f(n)/n$. We split up the right-handed expression as follows:

$$\lim_{n\to\infty} \underbrace{-\frac{\log(1-\frac{\log q}{q^n})}{\log q}}_{a_n} + \underbrace{\frac{\log(1-2\log q)}{2\log q}}_{b_n}\underbrace{\exp\left(\frac{1-q^n}{\log q}\right)}_{c_n}$$

For the limit of a_n we get

$$\begin{aligned}\lim_{n\to\infty} a_n &= \lim_{n\to\infty} -\frac{\overbrace{\log\left(1-\frac{\log(1-f(n)/n)}{(1-f(n)/n)^n}\right)}^{\leq -\frac{\log(1-f(n)/n)}{(1-f(n)/n)^n}}}{\log(1-f(n)/n)} \\ &= \lim_{n\to\infty} ((1-f(n)/n)^n)^{-1} \\ &= e^{f(n)}.\end{aligned}$$

For b_n we use the following identity.

$$\lim_{x\to 0}\frac{\log(1+x)}{x} = 1 \tag{6.5}$$

We observe that $1 - f(n)/n$ approaches 1 as $n \to \infty$ since $f(n) = o(n)$. Hence, $\lim_{n\to\infty} -2\log(1 - f(n)/n) = 0$. By setting $x = -2\log(1-f(n)/n)$ and using Equation (6.5) we get $\lim_{n\to\infty} b_n = 1$.

Finally, we consider

$$\begin{aligned}\lim_{n\to\infty} c_n &= \lim_{n\to\infty}\exp\left(\frac{\overbrace{1-(1-f(n)/n)^n}^{\to 1-e^{-f(n)}}}{\underbrace{\log(1-f(n)/n)}_{\to 0(<0)}}\right) \\ &= 0.\end{aligned}$$

Altogether, since the limits of b_n and c_n exist, we have

$$\begin{aligned}\lim_{n\to\infty} g(n) &< \lim_{n\to\infty} a_n + \lim_{n\to\infty} b_n \lim_{n\to\infty} c_n \\ &= e^{f(n)}.\end{aligned}$$

Clearly, $e^{f(n)} = o(n)$ for $f(n) = o(\log(n))$ and $e^{f(n)} = \Omega(n)$ for $f(n) = \Omega(\log n)$. Since trivially, $\mathbb{E}[|\gamma(s)|] = O(n)$, the lemma follows. ■

Lemma 6.29 implies, e.g., that for sparse random dags with $p = c/n$ for some constant c, the number of vertices reached by any vertex v is in expectation also bounded by a constant, i.e., e^c. Dags originating from real-world applications are often sparse in this sense, that is, the average degree of the vertices can be bounded by a constant, see [EMN07] and Chapter 7. However, under practical considerations, the constant e^c might be quite large. Lemma 6.29 also implies that for $p = f(n)/n$ and $f(n) = \Omega(\log n)$, the above approach does not provide an asymptotic advantage over the naive approaches with regard to space requirements.

Corollary 6.30. *Let G be a random dag in the $G_{n,p}$ model where $p = f(n)/n$. Then, the representative LCA problem can be solved in $\langle O(n^3), o(n^2), o(n)\rangle$ in the average case if $f(n) = o(\log n)$.*

Proof. Let $v \in V$ be any vertex in G. The quantity $|L_v|$ corresponds to the size of the transitive closure of v in the reverse dag G'. It is easy to see that G' is also a random dag in $G_{n,p}$ with parameter p. For all $v \in V$ it holds that $\mathbb{E}[|\gamma(v)|] \leq \mathbb{E}[|\gamma(s)|]$. By linearity of expectation we have $\sum_{v \in V} |L_v| \leq no(n) = o(n^2)$. The same reasoning yields $\mathbb{E}[|L_x| + |L_y|] = o(n)$. ■

In particular, for $f(n) = \Theta(1)$, the above approach yields $\langle O(n^3), O(n), O(1)\rangle$ and is thus in the average case in terms of space requirements and query speed competitive to the respective solutions for trees. We conclude by combining Corollaries 6.27 and 6.30.

Corollary 6.31. *Let G be a random dag in the $G_{n,p}$ model. Then, there exist solutions to the representative LCA problem with space requirement $o(n^2)$ and expected query time $o(n)$ in the average case for any choice of the parameter p.*

6.5 Summary

We have considered natural and well-motivated LCA problem variants and have predominantly been able to improve naive approaches. However, we conjecture that our contributions are mostly first steps in the direction of understanding the problems and that significant improvements can be expected. We are left with intriguing open questions in all of the three considered domains (dynamic algorithms, weighted variants, space-efficient algorithms). Computational relationships of some of the problems considered in this work are schematically depicted in Figure 6.4.

- Making the transition from unweighted to weighted dags in the context of (L)CA problems has left us with reasonable accurate understanding of the common ancestor variants, but also with a complexity gap to the lowest common ancestor variants. Most intriguingly, the following questions are left open: Can one or both of the considered lowest common ancestor variants in weighted dags be solved without implicitly solving ALL-PAIRS ALL LCA? If yes, can the problems be solved in subcubic (neglecting the implications of the respective ALL-PAIRS SHORTEST DISTANCES problem) time? If no, is it possible to give a proof? Furthermore, it is not clear whether the level of difficulty of the two LCA variants is the same. Intuitively, one might lean towards a position that favors the opinion that the vertex-weighted variant is easier.

- With respect to dynamic LCA algorithms, it seems likely that the slight asymptotic improvements presented in this chapter can be significantly improved using more specialized approaches, e.g., to $\widetilde{O}(n^2)$ update time for dynamic ALL-PAIRS REPRESENTATIVE LCA. There is evidence for the fact that ALL-PAIRS REPRESENTATIVE LCA lies *in between* transitive closure computation and ALL-PAIRS SHORTEST DISTANCES as far as problem difficulty is concerned. For both of these problems solutions with update time $\widetilde{O}(n^2)$ have been described. This might indicate that a direct extension of the currently optimal static solutions to the dynamic setting is ultimately not the optimum approach. Observe that any approach that follows the idea of Section 4.3 needs $\Omega(n^{2.5})$ update time. Similarly, there is hope that the online version of ALL-PAIRS REPRESENTATIVE LCA can be improved by exploiting combinatorial relationships between vertices that obtain new topological numbers and vertex pairs for which the maximum CA has to be updated by these changes in a more subtle way.
- As stated above, the state of studies in the field of encoding partial orders might impose a possible theoretical obstacle to general space-efficient, yet reasonably fast LCA data structures. However, the arsenal of algorithmic approaches that yield good solutions for special dag classes and in practical scenarios might not be fully consumed. Can, for example, a bounded height or uniqueness of LCAs be used to obtain space-efficient data structures? Is it possible to combine benign dag attributes to narrow down the class of bad dags in a similar way as done in Section 5.5?

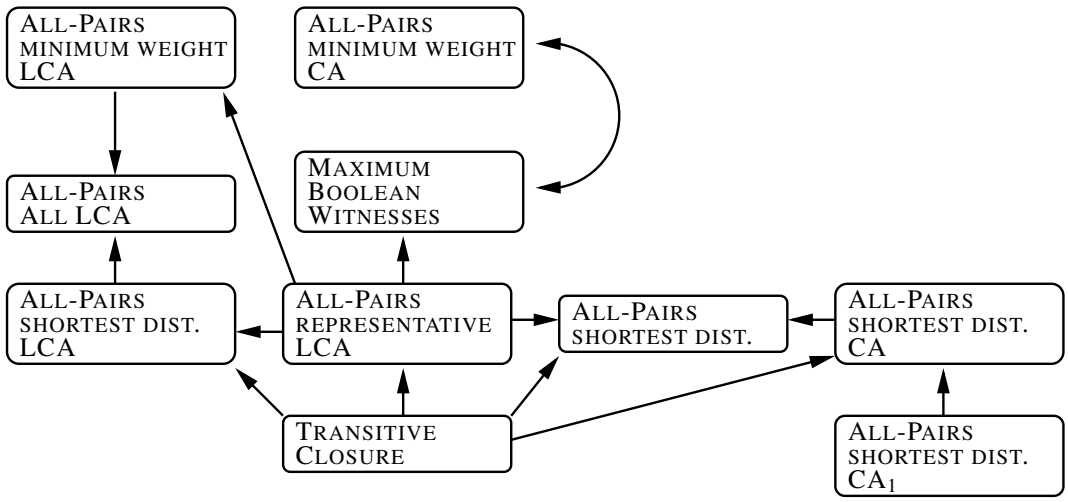


Figure 6.4: Schematic description of some of the currently established problem relationships. An arrow from A to B means that A can be reduced to B. Observe that we ignore polylogarithmic factors for reductions.

CHAPTER

7

EXPERIMENTAL ANALYSIS

7.1 Introduction

In this chapter we describe the results of an experimental analysis of LCA algorithms. In particular, we conduct a study of the dynamic-programming-based algorithms for ALL-PAIRS REPRESENTATIVE LCA and ALL-PAIRS ALL LCA. The benefits of the experimental analysis are twofold. A recent trend in algorithmics is *algorithm engineering*. Basically, algorithm engineering aims at bridging the gap between theoretical bounds and algorithmic solutions of practical relevance. Many of the algorithmic methods presented in Chapters 4 and 5 have limited practical relevance. This is particularly true for the methods that use fast matrix multiplication. Although there are reasonable efficient implementations of subcubic matrix multiplication algorithms [DN05, DN07], the methods yielding the best exponents [CW90] are currently considered to be impractical. To be a good candidate for a practically good solution, an algorithm should be reasonably easy to implement and reasonably efficient on typical problem instances. Here, typical means data that has characteristics of data that is expected in the respective applications. We close the gap between the established theoretical upper bounds and practically satisfying solutions by the experimental study presented in this chapter.

On the other hand, experimental analysis of algorithms can be used to derive asymptotic properties of algorithms where mathematical analysis fails or is too much of an effort to be conducted. The average case complexities of the dynamic programming algorithms given in Chapter 3 give explanation for the good performance of the algorithms on random dags in the $G_{n,p}$ model. However, this is not sufficient to conclude that the algorithms perform well on typical data. Therefore, experimental analysis of algorithms is used to extend the results of the average case analysis to more refined random models and real world instances.

In Section 7.2 we describe the experimental setup, i.e., the tested algorithms and respective implementations, the used input instances, and the evaluation approach. Subsequently, we present the results of the experimental study in Section 7.3. The main conclusions that can be drawn from the study are the following:

- ALL-PAIRS REPRESENTATIVE LCA can be solved by using the dynamic programming algorithm presented in Chapter 3 in time close to $O(n^2)$ in practice. In fact, an upper bound of $O(n^2)$ seems to hold except for dags in which the number of edges is close to $n \log n$. For such dags, the results indicate an upper time bound of $O(n^2 \log n)$.
- Similarly, algorithms based on dynamic programming solve ALL-PAIRS ALL LCA efficiently. Here, the improvement with regard to the best general upper bound is even more dramatic, from $O(n^{3.334})$ to approximately $O(n^2)$. As in the case of ALL-PAIRS REPRESENTATIVE LCA, instances having close to $n \log n$ edges appear to be the most difficult. Here, however, the additional penalty is roughly a factor of n rather than $\log n$.

7.2 Experimental Setup

7.2.1 Implementation

We have implemented Algorithm 2 (DP-LCA), Algorithm 3 (TC-APA), Algorithm 4 (DP-APA) described in Chapter 3, and Algorithm 6 (PC-APA) described in Chapter 5 in C++[1]. For comparison with two of the algorithms that were subject to the experimental study in [BFCP+05], we adapted the code provided by the authors to fit in our C++ framework. We followed the concept of making as few changes to the original code as possible in order to prevent bias against or for foreign methods. The third algorithm tested in [BFCP+05] was ruled out due to its inferior performance. This finding is in accordance with the results of the study presented in the respective paper. The first algorithm (RMQ) 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 $O(n^3)$. However, RMQ was by far the best performing algorithm in the experimental study in [BFCP+05]. The second algorithm (TC) is based on transitive closure look-ups. It computes G_{clo} first and then chooses the maximum CA of a pair $\{x,y\}$ by comparing the corresponding rows of the adjacency matrix A_{clo} of G_{clo}. The running time of the algorithm is $\Theta(n^3)$.

Additionally, we implemented the two transitive closure algorithms described in [Sim88]: the algorithm of Goralćíková and Koubek[GK79] (GK) with average case running time of $O(n^2 \log n)$ and the algorithm of Simon [Sim88] (Simon) with average case running time of $O(n^2)$ [Cri94]. Our preprocessing routines are complemented by the greedy algorithm for computing a chain cover of G with running time $O(n+m)$ [Sim88], see also Section 3.2, Algorithm 1. 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 widely believed to be not efficient in practice.

7.2.2 Test Data

We tested the algorithms on four families of dags:

- $G_{n,p}$ **random dags**: In order to mirror random dags of varying density we use the parameter p to control the expected number m of edges in the dag G. Observe that $\mathbb{E}[m] = p \cdot \binom{n}{2}$. We created the following random $G_{n,p}$ dags.
  - *sparse*: $p = 4/n$, i.e., $m \approx 2n$

[1] Our implementations and the used test data are available at http://wwwmayr.informatik.tu-muenchen.de/personen/nowakj/lca/

  - *medium*: $p = 2\log n/n$, i.e., $m \approx n \log n$
  - *dense*: $p = 0.5$, i.e., $m \approx 0.25n^2$

- $G_{n,m}$ **random dags**: $G_{n,m}$ random dags were created in an analogous way. Here, we fix the number of edges instead of the parameter p.
  - *sparse*: $m = 2n$
  - *medium*: $m = n \log n$
  - *dense*: $m = 0.25n^2$

- **Power law random dags:** Modeling real world data often leads to graphs in which the degrees of the vertices satisfy a power law. That is, the number of vertices of degree k is proportional to $k^{-\alpha}$ for some constant $\alpha > 1$. This is also true for dags, e.g., a citation graph compiled from crawling scientific literature obeys a power law with $\alpha \approx 1.7$ [AJM04]. There are numerous examples of large-scale networks that fall into the category of power law graphs, most prominently the *World-Wide-Web* [AJB99, KKR+99]. We generated random dags in which the expected out-degrees of the vertices follow a power law by slightly adapting the Chung-Lu model [ACL01, CL02]. First, target out-degrees of the vertices are generated according to the power law. Suppose that the vertices are sorted in decreasing order with respect to their target degrees. Then, for each vertex pair (i, j), $i < j$ (with respect to the order) an edge (i, j) is added with probability $p_{i,j} = \frac{\deg_i}{n-i}$, where $\deg_i$ is the target degree of vertex i. We generated dags for $\alpha = 3$ (sparse), $\alpha = 2$ (medium), and $\alpha = 1.5$ (dense), representing common exponents. It should be noted that the densities of power law dags are not directly comparable with their respective $G_{n,p}$ and $G_{n,m}$ counterparts. While the power law dags for $\alpha = 1.5$ are indeed the densest instances, the expected number of edges in such graphs is only slightly superlinear. The exponents for these models are chosen to reflect commonly observed exponents.

- **Real-world data sets**: Our real-world data sets include the *Internet dag*, the *citation dag*, and phylogenetic networks.

  1. The Internet dag is derived from business relationships of autonomous systems (ASes) in the Internet. Our data set is obtained from observable BGP routes. From these routes an acyclic AS graph is inferred with the method proposed in [KMT06]. We use the same data set as in [HK07], where a detailed description of the data collection and postprocessing steps can be found. The final dag has 22,218 vertices and 57,413 edges.
  2. The citation dag is compiled from Citeseer's OAI publicly available records[2]. Similar graphs have been studied and analyzed extensively in the past, see [AJM04] and references therein. Computing LCAs in such dags may provide valuable information, e.g., which papers are original to two or more papers on a particular topic. The dag has 716,772 vertices and 1,331,948 edges. The data provided by CiteSeer is automatically obtained by crawling the web and not free of errors. We first parsed the documents and created a citation graph, where an edge (x, y) was added whenever y references x. We then manually cleaned the data by ordering the vertices according to their timestamps (which are part of the data) and deleting edges that are not consistent with the partial order imposed by the timestamps.
  3. We considered two phylogenetic networks for which the data sets are included as examples in the SplitsTree4 [HB06] package. The first network represents evolutionary relationships

[2] http://citeseer.ist.psu.edu/oai.html

between mammals. It has 498 vertices and 889 edges. The second network represents evolutionary relationships between a single protein, namely myosin, a protein involved in muscle contraction, in different organisms. The network has 5462 vertices and 10,321 edges.

The Internet dag and the citation dag had to be sampled in order to be processed by our algorithms. We were interested in dense subgraphs in order to evaluate the effect of the transitive reduction in real-world dags. To achieve this, we sampled the graphs by performing breadth first searches starting at high-degree vertices. To sample a graph of size n, we choose the dag which is induced by the first n visited vertices. The sampling imposes a bias towards dense subgraphs. However, most of the instances generated are still *s*parse in the sense that the average degree of the vertices is a small constant, e.g., < 5 in the case of the citation graph samples. The densities of the used samples are plotted in Figure 7.1. These graphs have to be taken into account when comparing the results of the experimental study on the real-world data sets with the results on randomly generated instances in which the dag densities can be controlled. Although the effect of the transitive reduction on sparse dags is small, naturally, the $O(nm_{\text{red}})$ dynamic programming approach is by far the best choice for such dags.

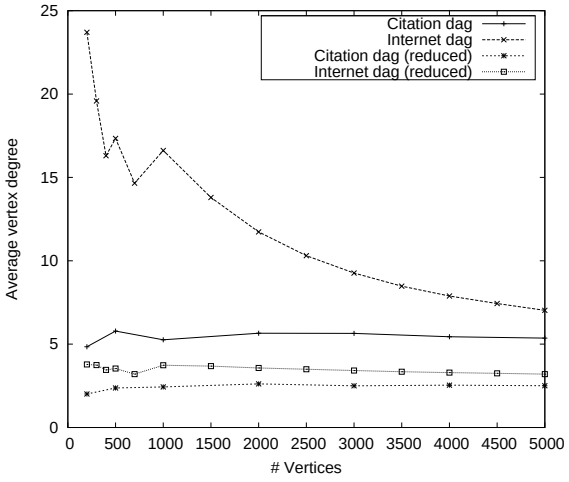


Figure 7.1: Densities of the real-world dags before and after application of transitive reduction.

7.2.3 Evaluation

We evaluated the performance of the algorithms on all input dag families, varying densities, and varying graph sizes. The performance was measured in terms of CPU time. On randomly created instances, the experiments were repeated several times in order to obtain an accurate average. However, we note at this point that the number of iterations had to be held small on some combinations of slow algorithms and large problem instances due to extremely long computation times (more than one hour for one iteration in such cases). Nevertheless, in all cases the variances of the running time were rather small so that as few as 10 iterations give an accurate estimate of the expected value.

An important goal of the experimental evaluation was to verify the results implied by the average case analyses given in Chapters 3 and 5 in models that are not directly covered by assuming an input space distribution according to the $G_{n,p}$ model, i.e., other random dag models and real-world instances.

Although constant factors matter in practice, we focus on the derivation of asymptotic behavior in the presentation of the experimental results below. However, in cases where competing algorithms appear to have the same asymptotics, we aim at indicating constant differences as well.

Measuring running times as *time per element*, i.e., dividing the running time by an adequate factor (e.g., per vertex, or per vertex pair) provides insights about the asymptotic behavior, see, e.g., [San00]. We make use of such representations of our results throughout this chapter. More specifically, suppose we evaluate results on a dag G with n vertices. We use the following terminology:

- *time per vertex*: total running time divided by n
- *time per vertex pair*: total running time divided by n^2

Note that we use n^2 as number of vertex pairs instead of $n(n-1)$ or $n(n-1)/2$. This does not affect the respective asymptotic conclusions.

All tests were performed on a system with an AMD Athlon 64 X2 6000+ dual core processor clocked at 3013 MHz with 2 GB RAM running on Linux.

7.3 Results

Transitive Reduction: We first compared the two methods for creating the transitive reduction of a dag. Simon clearly outperforms GK on medium and dense dags. Moreover, the space requirement for GK is significantly larger than for Simon imposing a limiting factor on large problem instances. As a result of the study Simon was used to create the transitive reductions in all further experiments.

Since all of the dynamic programming algorithms depend heavily on the number of edges in the transitive reduction of the input dag G, we measured the actual size of the reduction on the different input data sets, see Figure 7.2. Results for $G_{n,p}$ random dags follow the known results on the respective expectations (Chapter 3). The sizes of the transitive reductions of $G_{n,m}$ random dags seem to be close to equal to the sizes of the comparable $G_{n,p}$ dags; this indicates that the average case results of the previous chapters hold for an input space that is distributed according to the $G_{n,m}$ model. However, Proposition 3.3 cannot be applied to establish this mathematically, since m_{red} is not a monotonic graph property. The graphs depicted in Figure 7.2 give also a first indication to the fact that medium-sized instances turn out to be the most difficult. This is in accordance with the results presented below.

7.3.1 All-Pairs Representative LCA

We tested the effect of using transitive reduction along with the RMQ algorithm. The effect is very small compared to the dynamic programming algorithms, but improves the running time slightly. Subsequently, transitive reduction is used also for RMQ, whereas TC clearly does not benefit from transitive reduction. As a first immediate consequence of the results with respect to the tested ALL-PAIRS REPRESENTATIVE LCA algorithms, DP-LCA is far superior to both RMQ and TC on all tested input classes, where the advantage of DP-LCA over its closest competitor RMQ is minimized for sparse input dags. Figure 7.3 depicts a sample of the respective results.

We turn our attention to the evaluation of the asymptotics of the considered algorithms. Since the time complexity of TC is $\Theta(n^3)$, we exclude the respective results from this study. Close examination of the RMQ algorithm reveals that the running time depends effectively on the total size of the ancestor

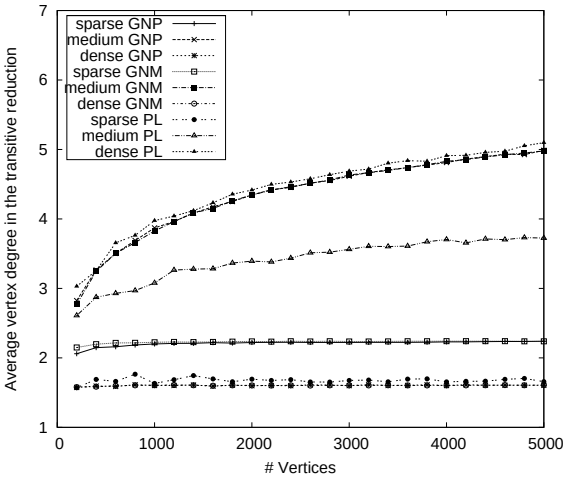
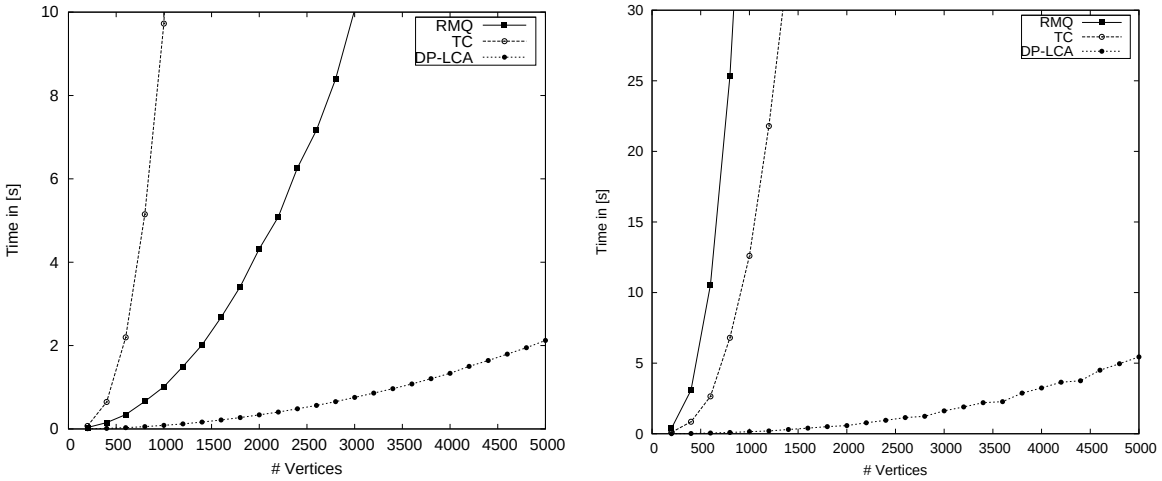


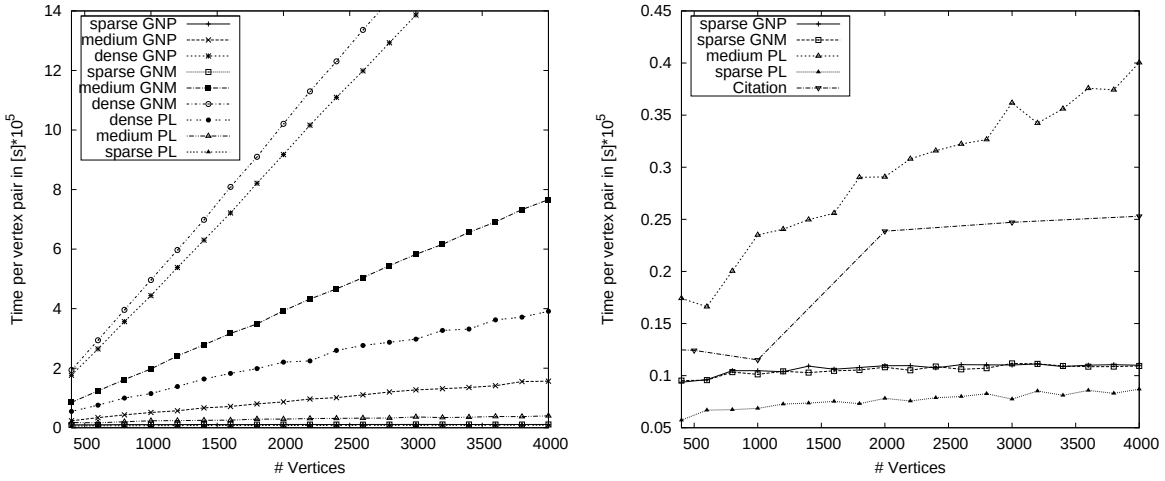
Figure 7.2: Average vertex degree after applying transitive reduction.



(a) sparse $G_{n,m}$ (b) dense $G_{n,m}$

Figure 7.3: Performance comparison of ALL-PAIRS REPRESENTATIVE LCA algorithms (sample).

lists and hence on the size of the respective transitive closures. The expected transitive closure sizes in random $G_{n,p}$ dags can be bounded by $O(n \cdot e^{f(n)})$, where $p = f(n)/n$, recall the proof of Lemma 6.29. This in turn implies that the running time of RMQ can be expected to be quadratic on sparse $G_{n,p}$ and, eventually, input instances with similar properties. Experimental evaluation supports this as can be seen in Figure 7.4. The running times for the majority of input classes are cubic, with exceptions given in Figure 7.4(b). Sparse $G_{n,p}$, sparse $G_{n,m}$, and sparse power law seem to be close to $O(n^2)$. Medium power law is clearly superquadratic, however, the additional factor is probably only polylogarithmic.



(a) all input classes

(b) input classes with close to quadratic running time

Figure 7.4: Asymptotic evaluation of RMQ on various input classes.

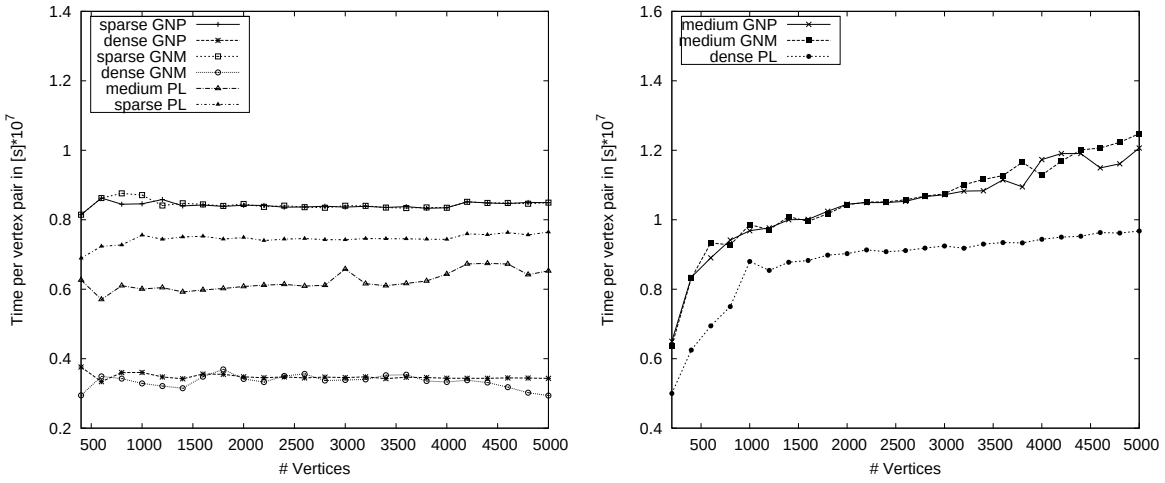
Figure 7.5 depicts the running times of DP-LCA measured as time per vertex pair. The results indicate that DP-LCA has running time close to $O(n^2)$ for all tested input classes. However, while an actual upper bound of $O(n^2)$ seems to hold for almost all classes, the graphs in Figure 7.5(b) show that the running times are slightly superquadratic for medium $G_{n,p}$, medium $G_{n,m}$, and dense power law dags. The additional factor is conjectured to be proportional to $\log n$.

We have not included most of the respective graphs for the real-world test instances in Figures 7.4 and 7.5. The reason for this is that the graphs appear wild and erratic, in particular in high resolution. Recall that the densities of these graphs are not controlled as in the case of the randomly created graphs, see Figure 7.1. However, the real-world dags behave much like sparse $G_{n,p}$ or $G_{n,m}$ random dags as the number of vertices grows, see Figure 7.6. Even the running time of RMQ seems to be quadratic. The finding that the real-world graphs appear to have the same asymptotic behavior as sparse $G_{n,p}$ and $G_{n,m}$ dags is supported by all of our conducted experiments. However, we usually do not print the respective graphs below in order to keep clarity.

We summarize the conclusions drawn from the experiments with regard to the asymptotic behavior of the tested algorithms in Table 7.3.1.

7.3.2 All-Pairs All LCA

We continue by considering ALL-PAIRS ALL LCA. The algorithms that were subject to the experimental evaluation are:



(a) quadratic (b) superquadratic

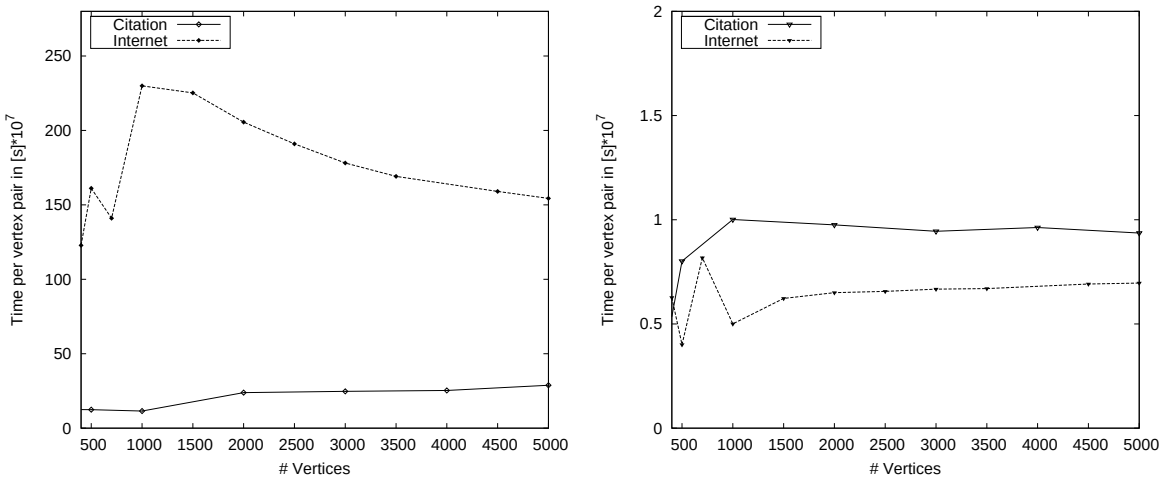
Figure 7.5: Performance evaluation of DP-LCA measured as CPU-time per vertex pair

	TC	RMQ	DP-LCA
sparse $G_{n,p}$	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium $G_{n,p}$	$O(n^3)$	$O(n^3)$	$O(n^2 \cdot \text{polylog(n)})$
dense $G_{n,p}$	$O(n^3)$	$O(n^3)$	$O(n^2)$
sparse $G_{n,m}$	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium $G_{n,m}$	$O(n^3)$	$O(n^3)$	$O(n^2 \cdot \text{polylog(n)})$
dense $G_{n,m}$	$O(n^3)$	$O(n^3)$	$O(n^2)$
sparse PL	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium PL	$O(n^3)$	$O(n^2\text{polylog(n)})$	$O(n^2 \cdot \text{polylog(n)})$
dense PL	$O(n^3)$	$O(n^3)$	$O(n^2 \cdot \text{polylog(n)})$
real-world	$O(n^3)$	$O(n^2)$	$O(n^2)$

Table 7.1: Predicted asymptotic behavior of tested ALL-PAIRS REPRESENTATIVE LCA algorithms.

1. TC-APA: the simple method (Algorithm 3) with running time $O(n^2 m_{\text{red}})$.
2. DP-APA-it: the dynamic programming method (Algorithm 4) using the iterative merging strategy with running time $O(n m_{\text{red}} \text{w}(G) \kappa)$.
3. DP-APA-lazy: the dynamic programming method (Algorithm 4) using the lazy merging strategy with running time $O(n m_{\text{red}}(\kappa^2 + \kappa \log n))$.
4. PC-APA: the method that uses ALL-PAIRS REPRESENTATIVE LCA solutions and the path cover technique (Algorithm 6) with running time $O(n m_{\text{red}} \text{w}(G))$.

Again, the performance of the considered algorithms depends on the number of edges in the transitive reductions of the input dags, see Figure 7.2. Comparing the performance of the tested algorithms, it is readily seen that the two dynamic programming algorithms (DP-APA-it and DP-APA-lazy) clearly outperform TC-APA and PC-APA on almost all input dag classes. The only exception to this finding is the good performance of PC-APA on very dense input classes, i.e., dense $G_{n,p}$ and $G_{n,m}$. The size of the path cover is sufficiently small only in these tests. Observe that this can be expected considering results on the expected width of random dags [Sim88, BE84].



(a) RMQ (b) DP-LCA

Figure 7.6: Performance evaluation of RMQ and DP-LCA on real-world instances

# vertices	RMQ	TC	DP-LCA	
Mammals	498	1.45	1.33	0.01
Myosin	5462	1540.95	1917.73	2.01

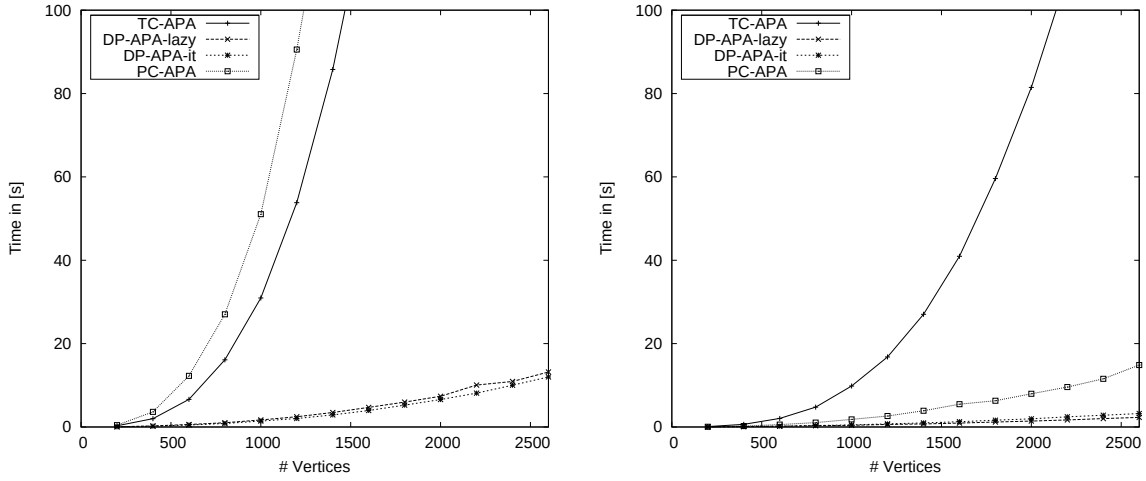
	n	TC-APA	PC-APA	DP-APA-it	DP-APA-lazy
Mammals	498	12.82	2.4	0.26	0.46
Myosin	5462	19652.2	1821.02	38.76	66.01

Table 7.2: Experimental results for phylogenetic networks.

The results for the two versions of DP-APA, i.e., using the iterative and lazy merging strategies, are very similar. The asymptotic behavior (Figures 7.8 and 7.9) of both algorithm variants seems to be equivalent on all tested input classes. Most interestingly, the running times are close to quadratic for all but two input classes (medium-sized $G_{n,p}$ and $G_{n,m}$ dags). Observe that this is an improvement over the best general upper bound (Thm. 4.19) of more than an order of magnitude. Furthermore, this indicates that the trivial $\Omega(n^3)$ lower bound is rarely attained. In contrast to this, the running times on the *bad* input classes, namely medium-sized $G_{n,p}$ and $G_{n,m}$ dags, is not subcubic. The gap between theses two classes and the rest of classes is roughly a factor of n. The reason for this complexity gap is that the LCA set sizes in the medium-sized random dags are considerably larger than for their sparse and dense counterparts. Since the running time scales quadratically with this quantity, our dynamic programming algorithms are vulnerable to such spikes.

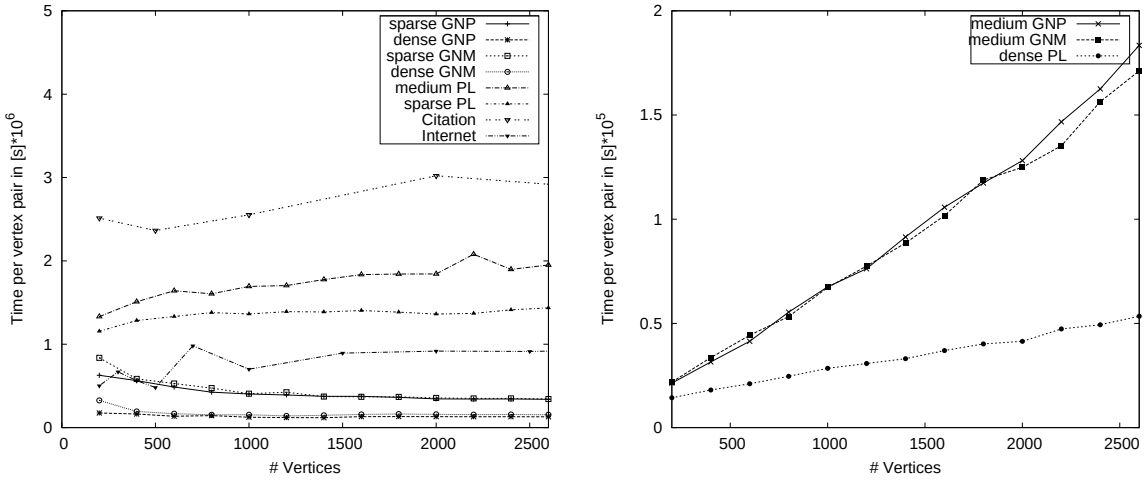
The running times of both TC-APA and PC-APA are roughly cubic on almost all input classes with the only exception being the results of PC-APA on dense $G_{n,p}$ and $G_{n,m}$ dags. For these dense instances, the running time of PC-APA seems to be close to quadratic, see Figure 7.10.

Interestingly, medium-sized dags turn out to be the most difficult problem instances both in terms of running time and maximum size of problem instances that can be handled by the main memory. We evaluated the size of the LCA sets in dependence of the dag densities with $n = 800$, see Figure 7.11. We conjecture that the values peak out in $G_{n,p}$ dags for $p \approx 8/n$. This is supported by Figure 7.11(b) which plots the expected average degree for the edge probability that maximizes the set sizes for various n. Figure 7.11(c) analyzes the p-values for which the average LCA set size is maximized. This quantity might be more suited for predicting the running time of the DP-APA algorithms. The p-



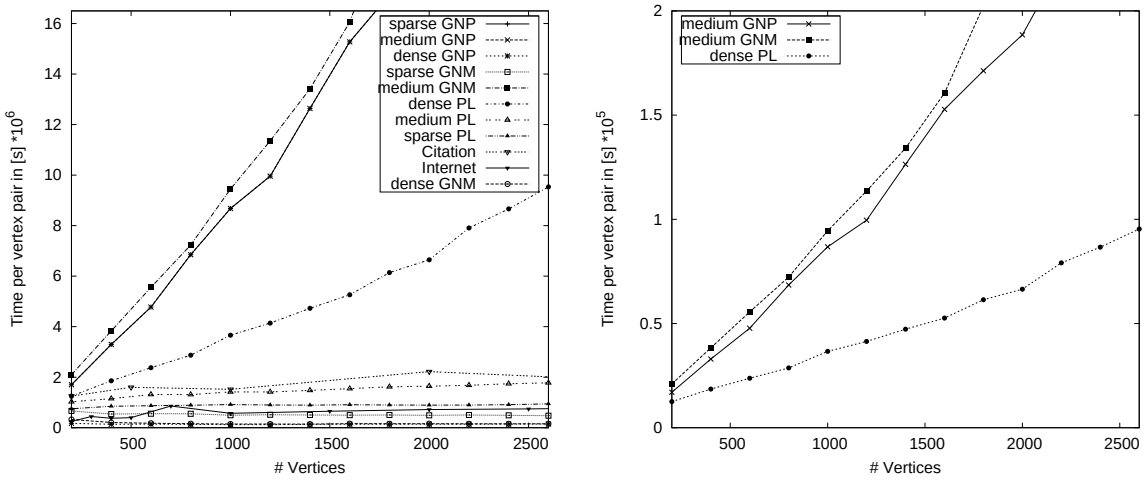
(a) medium power law (b) dense $G_{n,m}$

Figure 7.7: Performance comparison of ALL-PAIRS ALL LCA algorithms



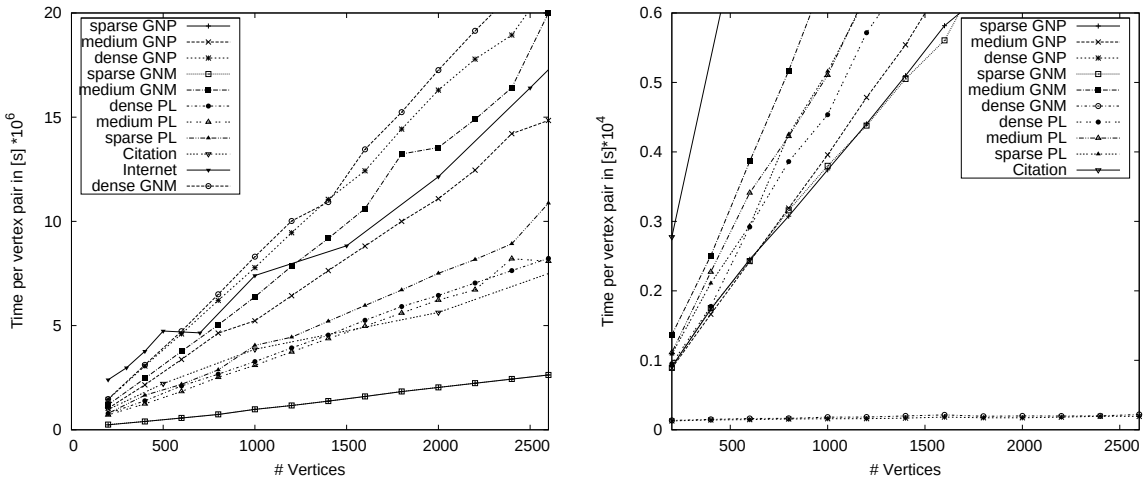
(a) input classes with quadratic running time (b) input classes with superquadratic running time

Figure 7.8: Asymptotic evaluation of DP-APA-lazy



(a) all input dag classes (b) superquadratic input dag classes

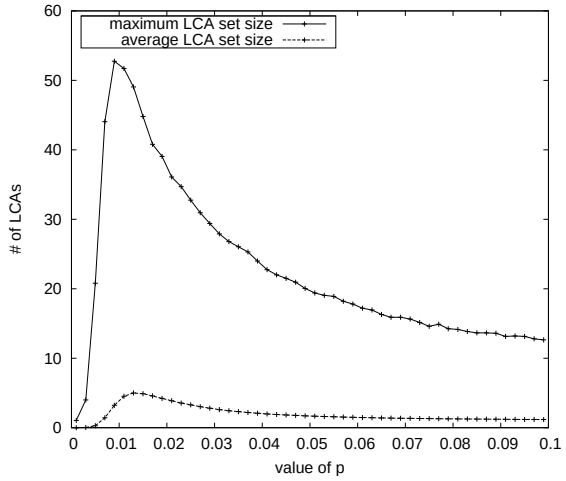
Figure 7.9: Asymptotic evaluation of DP-APA with iterative merging strategy.



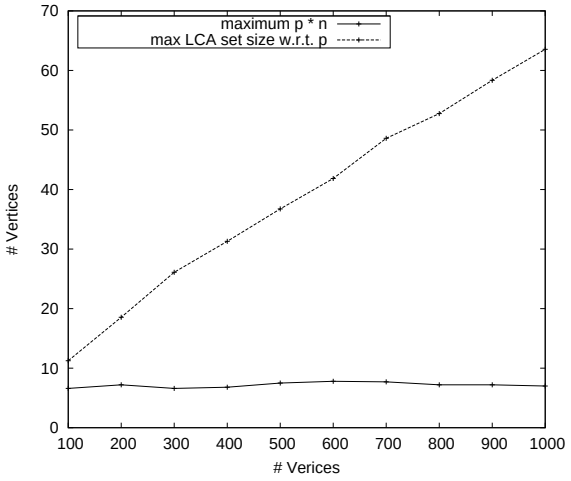
(a) TC-APA (b) PC-APA

Figure 7.10: Asymptotic evaluation of TC-APA and PC-APA.

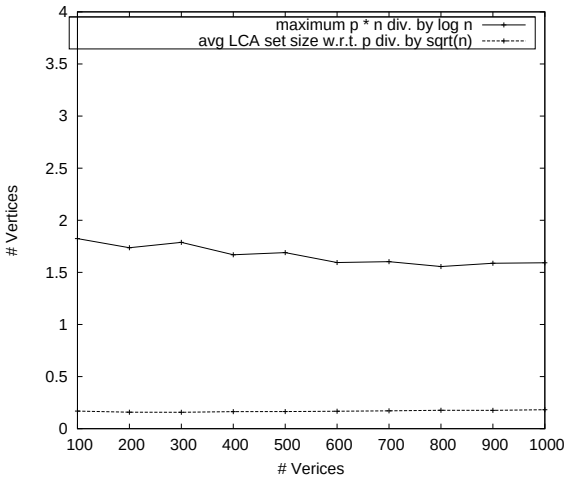
values seem to peak for $p \approx \log n/n$. The average LCA sizes are close to $\sqrt{n}$ for these p-values. This finding explains the inferior running time (close to $O(n^3)$) of the DP algorithms for such instances.



(a) maximum LCA set sizes for n=800



(b) maximum p w.r.t. maximum LCA set size



(c) maximum p w.r.t. average LCA set size

Figure 7.11: Evaluation of LCA set cardinalities in dependence of the parameter p in $G_{n,p}$ dags.

The conclusions drawn from the experimental study with regard to asymptotic properties of the ALL-PAIRS ALL LCA algorithms are summarized in Table 7.3.2.

	TC-APA	PC-APA	DP-APA-it	DP-APA-lazy
sparse $G_{n,p}$	$O(n^3)$	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium $G_{n,p}$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$
dense $G_{n,p}$	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
sparse $G_{n,m}$	$O(n^3)$	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium $G_{n,m}$	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$
dense $G_{n,m}$	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
sparse PL	$O(n^3)$	$O(n^3)$	$O(n^2)$	$O(n^2)$
medium PL	$O(n^3)$	$O(n^3)$	$O(n^3)$	$O(n^3)$
dense PL	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
real-world	$O(n^3)$	$O(n^3)$	$O(n^2)$	$O(n^2)$

Table 7.3: Predicted asymptotic behavior of tested ALL-PAIRS ALL LCA algorithms ignoring poly-logarithmic factors.

SUMMARY OF NOTATION

notation	meaning
G_{clo}	transitive closure of a (directed acyclic) graph G
m_{clo}	number of edges in G_{clo}
A_{clo}	adjacency matrix of G_{clo}
G_{red}	transitive reduction of a (directed acyclic) graph G
m_{red}	number of edges in G_{red}
A_{red}	adjacency matrix of G_{red}
$u \rightsquigarrow v$	there is a directed path from u to v in G
$\text{indeg}(v)$	number of incoming edges of vertex v
$\text{outdeg}(v)$	number of outgoing edges of vertex v
$N(v)$	the set of all vertices adjacent to v in G
$N^{\text{out}}(v)$	the set of children of vertex v in G
$N^{\text{in}}(v)$	the set of parents of vertex v in G
$N^{\text{out}}_{\text{clo}}(v)$	the set of successors of vertex v in G
$N^{\text{in}}_{\text{clo}}(v)$	the set of ancestors of vertex v in G
$\text{h}(v)$	height of a vertex v in G
$\text{dp}(\text{v})$	depth of a vertex v in G
$\text{dp}(\text{G})$	depth of a dag G
$\text{w}(G)$	width of a dag G
$top(v)$	topological number of a vertex v
$top^m(v)$	maximum topological number of a vertex v in any topological sort
$\text{CA}\{x,y\}$	set of all common ancestors of a vertex pair $\{x,y\}$ in G
$\text{LCA}\{x,y\}$	set of all lowest common ancestors of a vertex pair $\{x,y\}$ in G
$\text{MCA}\{x,y\}$	maximum CA of a vertex pair $\{x,y\}$ with respect to some topological order top
κ	maximum cardinality of an LCA set in G
$\overline{\kappa}$	average cardinality of the LCA sets in G
$\overline{V'}$	forbidden set w.r.t. V', i.e., all vertices $v \in V$ such that $v \rightsquigarrow v'$ for some $v' \in V$
$M[i,j]$	entry in row i and column j of a matrix M
$M[*,I]$	submatrix of columns of M whose indices belong to I

$M[I,*]$	submatrix of rows of M whose indices belong to I
$M[*,I,0]$	column-submatrix of M, consists of columns of M whose indices belong to I, other columns are 0-vectors
$M[I,*,0]$	row-submatrix of M, consists of rows of M whose indices belong to I, other rows are 0-vectors
M^T	transpose of the matrix M
ω	exponent for square matrix multiplication
$\omega(a,b,c)$	exponent of multiplication of an $n^a \times n^b$ by an $n^b \times n^c$ matrix
α	parameters satisfying $\alpha = \sup\{0 \leq r \leq 1 : \omega(1,r,1) = 2+o(1)\}$
β	constant satisfying $\beta = \frac{\omega-2}{1-\alpha}$
μ	parameter satisfying $1+2\mu = \omega(1,\mu,1)$
$\mathrm{d}_G(x,y)$	distance of a shortest path from x to y in a digraph G
$\mathrm{AD}\{x,y\}$	shortest ancestral distance of a vertex pair $\{x,y\}$
$\mathrm{AD}_z\{x,y\}$	ancestral distance of a vertex pair $\{x,y\}$ with respect to z
L_x	list of ancestors of a vertex x
DP	dynamic programming
G^R	(reduced) dag that results from a dag G_{red} by removing all edges incident to vertices covered by a partial chain cover
$\widetilde{O}(f(n))$	$O(f(n)\log^c n)$ for a constant $c > 0$

BIBLIOGRAPHY

[ABMP91] H. Alt, N. Blum, K. Mehlhorn, and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1.5}\sqrt{m/\log n})$. *Information Processing Letters*, 37(4):237–240, 1991.

[ACL01] William Aiello, Fan R. K. Chung, and Linyuan Lu. Random evolution in massive graphs. In *Proceedings of the* 42^{th} *Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 510–519. IEEE, 2001.

[ADKF70] V.L. Avlazarov, E.A. Dinic, M.A. Kronod, and I.A. Faradzev. On economical construction of the transitive closure of a directed graph. *Soviet Math. Doklady*, 11:1209–1210, 1970.

[AF07] Deepak Ajwani and Tobias Friedrich. Average-case analysis of online topological ordering. In *Proceedings of the* 18^{th} *International Symposium on Algorithms and Computation, (ISAAC 2007)*, volume 4835 of *Lecture Notes in Computer Science*, pages 464–475. Springer-Verlag, 2007.

[AFM06] Deepak Ajwani, Tobias Friedrich, and Ulrich Meyer. An $o(n^{2.75})$ algorithm for online topological ordering. In *Proceedings of the* 10^{th} *Scandinavian Workshop on Algorithm Theory (SWAT'06)*, volume 4059 of *Lecture Notes in Computer Science*, pages 53–64. Springer-Verlag, 2006.

[AGU72] 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.

[AHR+90] Bowen Alpern, Roger Hoover, Barry K. Rosen, Peter F. Sweeney, and F. Kenneth Zadeck. Incremental evaluation of computational circuits. In *Proceedings of the* 1^{st} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'90)*, pages 32–42. SIAM, 1990.

[AHU76] 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.

[AJB99] Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[AJM04] Yuan An, Jeannette Janssen, and Evangelos E. Milios. Characterizing and mining the citation graph of the computer science literature. *Knowledge and Information Systems*, 6(6):664–678, 2004.

[AKBLN89] 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.

[AN96] Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4–5):434–449, 1996.

[BE84] 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.

[BEG+06] 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.

[BEG+07] Matthias Baumgart, Stefan Eckhardt, Jan Griebsch, Sven Kosub, and Johannes Nowak. All-pairs common-ancestor problems in weighted directed acyclic graphs. In *Proceedings of the* 1^{st} *International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE'07)*, volume 4614 of *Lecture Notes in Computer Science*, pages 282–293, 2007.

[BFCP+05] 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.

[BFK99] András A. Benczúr, Jörg Förster, and Zoltán Király. Dilworth's theorem and its application for path systems of a cycle - implementation and analysis. In *Proceedings of the* 7^{th} *Annual European Symposium on Algorithms (ESA'99)*, volume 1643 of *Lecture Notes in Computer Science*, pages 498–509. Springer-Verlag, 1999.

[Bol01] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.

[BPSS01] Michael A. Bender, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Finding least common ancestors in directed acyclic graphs. In *Proceedings of the* 12^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 845–854. SIAM, 2001.

[BV94] Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *Journal of Computer and System Sciences*, 48(2):214–230, 1994.

[CH05] Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM Journal on Computing*, 34(4):894–923, 2005.

[CKL07] Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1-2):37–46, 2007.

[CL02] Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of Combinatorics*, 6(2):125–145, 2002.

[CL07] Artur Czumaj and Andrzej Lingas. Finding a heaviest triangle is not harder than matrix multiplication. In *Proceedings of the* 18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 986–994. SIAM, 2007.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2nd edition, 2001.

[Cop97] Don Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13(1):42–49, 1997.

[Cri94] Davide Crippa. *q-distributions and random graphs*. PhD thesis, ETH Zürich, 1994.

[CW90] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[DI04] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004.

[DI05] Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on dags: breaking through the $o(n^2)$ barrier. *Journal of the ACM*, 52(2):147–156, 2005.

[Dil50] Robert P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950.

[DN05] Paolo D'Alberto and Alexandru Nicolau. Adaptive strassen and atlas's dgemm: A fast square-matrix multiply for modern high-performance systems. In *Proceedings of the* 8^{th} *International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, page 45. IEEE Computer Society, 2005.

[DN07] Paolo D'Alberto and Alexandru Nicolau. Adaptive strassen's matrix multiplication. In *Proceedings of the* 21^{st} *annual international conference on Supercomputing (ICS'07)*, pages 284–292. ACM, 2007.

[EMN06] 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.

[EMN07] Stefan Eckhardt, Andreas Mühling, and Johannes Nowak. Fast lowest common ancestor computations in dags. In *Proceedings of the* 15^{th} *Annual European Symposium on Algorithms (ESA'07)*, volume 4698 of *Lecture Notes in Computer Science*, pages 705–716. Springer-Verlag, 2007.

[Fal98] Andrew Fall. The foundations of taxonomic encoding. *Computational Intelligence*, 14:598–642, 1998.

[FF62] Lester R. Ford and Delbert R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[FRS03] Stefan Felsner, Vijay Raghavan, and Jeremy Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003.

[Gao01] Lixin Gao. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking*, 9(6):733–745, 2001.

[GBT84] Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the* 16^{th} *Annual ACM Symposium*

on Theory of Computing (STOC 1984), pages 135–143. ACM, 1984.

[GK79] A. Goralćíková and Václav Koubek. A reduct-and-closure algorithm for graphs. In *Proceedings of the* 8^{th} *Symposium on Mathematical Foundations of Computer Science (MFCS'79)*, volume 74 of *Lecture Notes in Computer Science*, pages 301–307. Springer-Verlag, 1979.

[GR01] Lixin Gao and Jennifer Rexford. Stable Internet routing without global coordination. *IEEE/ACM Transactions on Networking*, 9(6):681–692, 2001.

[Gus97] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Comp. Science and Computational Biology*. Cambridge University Press, 1997.

[GW99] Timothy G. Griffin and Gordon T. Wilfong. An analysis of BGP convergence properties. *ACM SIGCOMM Computer Communication Review*, 29(4):277–288, 1999.

[HB06] Daniel H. Huson and David Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23(2):254–267, 2006.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[HK07] Benjamin Hummel and Sven Kosub. Acyclic type-of-relationship problems on the internet: An experimental analysis. Technical Report TUM-I0709, Institut für Informatik, Technische Universität München, 2007.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 2nd edition*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[HP98] Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of Complexity*, 14(2):257–299, 1998.

[HT84] Dov Harel and Robert E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[KB05] Irit Katriel and Hans L. Bodlaender. Online topological ordering. In *Proceedings of the* 16^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 443–450. SIAM, 2005.

[KKR+99] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. 1627:1–17, 1999.

[KL05] Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In *Proceedings of the* 32^{th} *International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 241–248. Springer-Verlag, 2005.

[KL07] Miroslaw Kowaluk and Andrzej Lingas. Unique lowest common ancestors in dags are almost as easy as matrix multiplication. In *Proceedings of the* 15^{th} *Annual European Symposium on Algorithms (ESA'07)*, volume 4598 of *Lecture Notes in Computer Science*, pages 265–274. Springer-Verlag, 2007.

[KLN08] Miroslaw Kowaluk, Andrzej Lingas, and Johannes Nowak. A path cover technique for LCA problems in dags. submitted for publication, 2008.

[KMT06] Sven Kosub, Moritz G. Maaß, and Hanjo Täubig. Acyclic type-of-relationship problems on the internet. In *Proceedings of the* 3^{rd} *Workshop on Combinatorial and Algorithmic Aspects of Networking (CAAN'06)*, volume 4235 of *Lecture Notes in Computer Science*, pages 98–111. Springer-Verlag, 2006.

[Man89] Udi Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[MNW+04] Bernard M. E. Moret, Luay Nakhleh, Tandy Warnow, C. Randal Linder, Anna Tholse, Anneke Padolina, Jerry Sun, and Ruth E. Timme. Phylogenetic networks: Modeling, reconstructibility, and accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):13–23, 2004.

[MS04] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *Proceedings of the* 45^{th} *Annual Symposium on Foundations of Computer Science (FOCS'04)*, pages 248–255. IEEE, 2004.

[MSNR96] Alberto Marchetti-Spaccamela, Umberto Nanni, and Hans Rohnert. Maintaining a topological order under edge insertions. *Information Processing Letters*, 59(1):53–58, 1996.

[NSW+03] Luay Nakhleh, Jerry Sun, Tandy Warnow, C. Randal Linder, Bernard M. E. Moret, and Anna Tholse. Towards the development of computational tools for evaluating phylogenetic network reconstruction methods. In *Proceedings of the* 8^{th} *Pacific Symposium on Biocomputing (PSB'03)*, pages 315–326. World Scientific Publishing, 2003.

[NU94] Matti Nykänen and Esko Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Information Processing Letters*, 50(1):307–310, 1994.

[NW05] Luay Nakhleh and Li-San 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, 2005.

[PK06] David J. Pearce and Paul H. J. Kelly. A dynamic topological sort algorithm for directed acyclic graphs. *ACM Journal of Experimental Algorithms*, 11, 2006.

[Ryt84] Wojciech Rytter. Fast recognition of pushdown automaton and context-free languages. In *Proceedings of the* 13^{th} *Symposium on Mathematical Foundations of Computer Science (MFCS'84)*, volume 176 of *Lecture Notes in Computer Science*, pages 507–515. Springer-Verlag, 1984.

[San00] Peter Sanders. Presenting data from experiments in algorithmics. In *Experimental Algorithmics*, pages 181–196. Springer-Verlag, 2000.

[San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *Proceedings of the* 45^{th} *Annual Symposium on Foundations of Computer Science (FOCS'04)*, pages 509–517. IEEE, 2004.

[San05] Piotr Sankowski. *Algebraic Graph Algorithms*. PhD thesis, Warsaw University, 2005.

[SCC93] Klaus Simon, Davide Crippa, and Fabian Collenberg. On the distribution of the transitive closure in a random acyclic digraph. In *Proceedings of the* 1^{st} *Annual European Symposium on Algorithms (ESA'93)*, pages 345–356, 1993.

[Sei95] Raimund Seidel. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of Computer and System Sciences*, 51(3):400–403, 1995.

[Sim88] Klaus Simon. An improved algorithm for transitive closure on acyclic digraphs. *Theoretical Computer Science*, 58:325–346, 1988.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 14(3):354–356, 1969.

[SV88] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[SYZ07] Asaf Shapira, Raphael Yuster, and Uri Zwick. All-pairs bottleneck paths in vertex weighted graphs. In *Proceedings of the* 18^{th} *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'07)*, pages 978–985. SIAM, 2007.

[Tho04] Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the* 9^{th} *Scandinavian Workshop on Algorithm Theory (SWAT'04)*, volume 3111 of *Lecture Notes in Computer Science*, pages 384–396. Springer-Verlag, 2004.

[Yus] Raphael Yuster. All-pairs disjoint paths from a common ancestor in $\widetilde{O}(n^{\omega})$ time. submitted for publication.

[Zwi98] Uri Zwick. All pairs shortest paths in weighted directed graphs–exact and almost exact algorithms. In *Proceedings of the* 39^{th} *Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 310–319. IEEE, 1998.

[Zwi02] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49(3):289–317, 2002.