Listing, Verifying and Counting Lowest Common Ancestors in DAGs: Algorithms and Fine-Grained Lower Bounds

Surya Mathialagan ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Virginia Vassilevska Williams ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Yinzhan Xu ✉
Massachusetts Institute of Technology, Cambridge, MA, USA

Abstract

The AP-LCA problem asks, given an n-node directed acyclic graph (DAG), to compute for every pair of vertices u and v in the DAG a lowest common ancestor (LCA) of u and v if one exists, i.e. a node that is an ancestor of both u and v but no proper descendent of it is their common ancestor. Recently [Grandoni et al. SODA'21] obtained the first sub-$n^{2.5}$ time algorithm for AP-LCA running in $O(n^{2.447})$ time. Meanwhile, the only known conditional lower bound for AP-LCA is that the problem requires $n^{\omega-o(1)}$ time where ω is the matrix multiplication exponent.

In this paper we study several interesting variants of AP-LCA, providing both algorithms and fine-grained lower bounds for them. The lower bounds we obtain are the first conditional lower bounds for LCA problems higher than $n^{\omega-o(1)}$. Some of our results include:

- In any DAG, we can detect all vertex pairs that have at most two LCAs and list all of their LCAs in $O(n^\omega)$ time. This algorithm extends a result of [Kowaluk and Lingas ESA'07] which showed an $\tilde{O}(n^\omega)$ time algorithm that detects all pairs with a unique LCA in a DAG and outputs their corresponding LCAs.
- Listing 7 LCAs per vertex pair in DAGs requires $n^{3-o(1)}$ time under the popular assumption that 3-uniform 5-hyperclique detection requires $n^{5-o(1)}$ time. This is surprising since essentially cubic time is sufficient to list all LCAs (if $\omega = 2$).
- Counting the number of LCAs for every vertex pair in a DAG requires $n^{3-o(1)}$ time under the Strong Exponential Time Hypothesis, and $n^{\omega(1,2,1)-o(1)}$ time under the 4-Clique hypothesis. This shows that the algorithm of [Echkardt, Mühling and Nowak ESA'07] for listing all LCAs for every pair of vertices is likely optimal.
- Given a DAG and a vertex $w_{u,v}$ for every vertex pair u, v, verifying whether all $w_{u,v}$ are valid LCAs requires $n^{2.5-o(1)}$ time assuming 3-uniform 4-hyperclique requires $n^{4-o(1)}$ time. This defies the common intuition that verification is easier than computation since returning some LCA per vertex pair can be solved in $O(n^{2.447})$ time.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases All-Pairs Lowest Common Ancestor, Fine-Grained Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.94

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2204.10932`

Funding *Surya Mathialagan*: Supported by the MIT Akamai Presidential Fellowship.
Virginia Vassilevska Williams: Supported by an NSF CAREER Award, NSF Grants CCF-1528078, CCF-2129139 and CCF-1909429, a BSF Grant BSF:2020356, a Google Research Fellowship and a Sloan Research Fellowship.
Yinzhan Xu: Partially supported by NSF Grants CCF-1528078 and CCF-2129139.

1 Introduction

A lowest common ancestor (LCA) of two nodes u and v in a directed acyclic graph (DAG) is a common ancestor c of u and v such that no proper descendent of c is a common ancestor of u and v. The AP-LCA problem asks to compute for every pair of nodes in a given DAG, some LCA, provided a common ancestor exists.

Computing LCAs is an important problem with a wide range of applications. For instance, LCA computation is a key ingredient in verification of the correctness of distributed computation (e.g. [10]), object inheritance in object oriented programming languages such as C++ and Java (e.g. [2, 19, 26]), and computational biology for finding the closest ancestor of species in rooted phylogenetic networks (e.g. [22]).

Computing LCAs is very well-understood in trees [7, 8, 17, 24, 27, 37, 38, 40]. Aït-Kaci, Boyer, Lincoln and Nasr [2] were one of the first to consider LCAs in DAGs, focusing on lattices and lower semilattices with object inheritance in mind. Nykänen and Ukkonen [36] obtained efficient algorithms for directed trees and asked if there is a subcubic time algorithm for AP-LCA in DAGs.

Bender, Martin Farach-Colton, Pemmasani, Skiena and Sumazin [6] gave the first subcubic, $O(n^{(3+\omega)/2}) \leq O(n^{2.687})$, time algorithm for AP-LCA in DAGs, where $\omega < 2.37286$ is the matrix multiplication exponent [4]. They also showed that AP-LCA is equivalent to the so-called All-Pairs Shortest LCA Distance problem. Czumaj, Kowaluk and Lingas [30, 18] improved the AP-LCA running time to $O(n^{2.575})$ using a reduction to the Max-Witness Product problem. With the current best bounds for rectangular matrix multiplication [33], their algorithm runs in $O(n^{2.529})$ time.

Notice that all subcubic algorithms above would run in $\tilde{O}(n^{2.5})$ time[1] if $\omega = 2$. For more than a decade, this running time remained unchallenged. It seemed that AP-LCA might actually require $n^{2.5-o(1)}$ time, similar to several other $n^{2.5}$ time problems such as computing the Max-Witness product (see e.g. [34]).

Recently, Grandoni, Italiano, Lukasiewicz, Parotsidis and Uznanski [25] showed that this is not the case, giving an algorithm that runs in $O(n^{2.447})$ time, or in $\tilde{O}(n^{7/3})$ time if $\omega = 2$.

It is not hard to show (see [6, 18]) that any algorithm for AP-LCA can be used to solve Boolean Matrix Multiplication (BMM), and hence beating $O(n^{\omega})$ time for AP-LCA would likely be difficult. No higher conditional lower bounds are known for the problem. It is still open whether $O(n^{\omega})$ time can actually be achieved for AP-LCA.

Partial progresses have been made for DAGs with special structures or for variants of AP-LCA. Czumaj, Kowaluk and Lingas [18] showed that AP-LCA is in $O(n^{\omega})$ time for low-depth DAGs. Kowaluk and Lingas [31] showed that in $O(n^{\omega} \log n)$ time one can return an LCA for every vertex pair that has a unique LCA. Eckhardt, Mühling and Nowak [20] showed that one can solve the AP-All-LCA problem, which asks to output all LCAs for every pair of vertices, in $O(n^{\omega(1,2,1)})$ time. Here $\omega(1, 2, 1) \leq 3.252$ is the exponent of multiplying an $n \times n^2$ by an $n^2 \times n$ matrix. AP-LCA was also studied in the weighted setting [5], the dynamic setting [20] and the space-efficient setting [32].

This paper considers the following questions:

1. Can we return all LCAs for every pair of nodes that has at most 2 LCAs, in $\tilde{O}(n^{\omega})$ time, extending Kowaluk and Lingas's algorithm [31]?
2. The AP-LCA problem asks us to exhibit a single LCA for each vertex pair. What if we want to list 2, 3, $\ldots, k$ LCAs? How fast can we do it?

[1] $\tilde{O}$ hides poly-logarithmic factors.

So far two variants of LCA are studied: list a single LCA per pair and list all LCAs per pair. What about listing numbers in between? This is just as natural. In phylogenetic networks for instance, there can be multiple LCAs per species pair, but typically not too many. Then listing a constant number of LCAs fast can give a better picture than listing a single representative. Other applications of AP-LCA would similarly make more sense for listing multiple LCAs.

3. How fast can we count the number of LCAs each vertex pair has?
4. Suppose that for every pair of nodes u, v in a DAG we are given a node $w_{u,v}$. Can we efficiently determine whether $w_{u,v}$ is an LCA of u and v, for each u, v? One would think that if AP-LCA can be solved faster than $O(n^{2.5})$ time, then this verification version of the problem should also be solvable faster.

We provide algorithms and fine-grained conditional lower bounds to address the above questions. Our lower bounds are the first conditional lower bounds higher than $n^{\omega-o(1)}$ for LCA problems.

1.1 Our results

Detecting and listing $O(1)$ LCAs. Our results for this part are summarized in Table 1.

Table 1 A summary of our results for detecting and listing LCAs. In the second and third columns, we give the best known runtime exponents for AP-AtLeast$k$-LCA and AP-List-k-LCA respectively. An exponent of 3 above corresponds to the trivial brute-force algorithm. In the fourth and fifth columns, we give the best conditional lower bounds for the exponent of AP-AtLeast$k$-LCA, and the corresponding hardness sources for the lower bounds. The exponents and lower bounds in the last row are for AP-All-LCA problem. All values in parentheses are the corresponding values when $\omega = 2$.

k	AP-AtLeast$k$-LCA Exponent		AP-List-k-LCA Exponent		Best Lower Bound		Source of LB
1	ω (2)	Folklore	2.447 (7/3)	[25]	ω (2)	[6]	BMM
2	ω (2)	[31], Thm 26	2.529 (2.5)	Thm 3	ω (2)	[6]	BMM
3	ω (2)	Thm 27	2.529 (2.5)	Thm 3	ω (2)	[6]	BMM
4	3		3		2.5	Thm 30	$(4,3)$-Hyperclique
5	3		3		2.666	Thm 30	$(5,3)$-Hyperclique
6	3		3		2.8	Thm 30	$(6,3)$-Hyperclique
7	3		3		3	Thm 30	$(5,3)$-Hyperclique
All	N/A		$\omega(1,2,1)$ (3)	[20]	$\omega(1,2,1)$ (3)	Thm 4, 5	SETH, 4-Clique

Let us define AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA as the problems of deciding for every pair of vertices in a given DAG, whether they have exactly, greater than or equal to, and less than or equal to k LCAs, respectively.

We study how fast AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA can be solved for constant k. More generally, we study the problem of returning k LCAs per vertex pair if it has at least k LCAs, or all LCAs if it has fewer. We call the latter problem AP-List-k-LCA.

For any constant k, one can return up to k LCAs for every vertex pair in a DAG in cubic time using a trivial brute-force algorithm[2]. More generally, if $\omega = 2$, the $O(n^{\omega(1,2,1)})$ time AP-All-LCA algorithm in [20] would also run in essentially cubic time.

[2] We first compute a topological ordering of the graph in $O(n^2)$ time and the transitive closure in $O(n^\omega)$ time using [23]. For each vertex pair (u, v), we scan the vertices in the reverse order of the topological ordering, and declare the current vertex w a new LCA if w can reach both u and v and w cannot reach any LCAs found so far. We stop the scan as soon as we find k LCAs or reach the end of the topological ordering. Given the transitive closure, each reachability check can be finished in $O(1)$ time, so the overall running time of the algorithm is $O(kn^3)$.

It is thus interesting to study *for what values of k,* AP-Exact$k$-LCA*,* AP-AtLeast$k$-LCA*,* AP-AtMost$k$-LCA *and* AP-List-k-LCA *are solvable in truly subcubic, $O(n^{3-\varepsilon})$ for $\varepsilon > 0$, time.*

We show that for every constant k, the listing problem AP-List-k-LCA and the decision problem AP-AtLeast$k$-LCA are subcubically equivalent. This statement appears as Theorem 28 in the main text. Thus, the values k for which one problem is in subcubic time are exactly the same for the other problem.

We also prove a convenient equivalence between AP-Exact$k$-LCA, AP-AtLeast$k$-LCA and AP-AtMost$k$-LCA:

▶ **Theorem 1.** *For any constant $k \geq 0$, the running times of* AP-Exact$k$-LCA*,* AP-AtMost$k$-LCA *and* AP-AtLeast$(k+1)$-LCA *are the same up to constant factors.*

Now we can focus on AP-Exact$k$-LCA, and due to the above equivalence, we also obtain results for the other variants.

Next, we extend the result of Kowaluk and Lingas [31] for pairs with unique LCAs to pairs with two LCAs by showing that AP-Exact$k$-LCA can be solved in $O(n^\omega)$ time for both $k = 1, 2$. Moreover, the corresponding witness LCAs can be listed in the same time.

▶ **Theorem 2.** AP-Exact1-LCA *and* AP-Exact2-LCA *can be solved in $O(n^\omega)$ time with high probability by Las Vegas algorithms. Moreover, finding the LCAs for vertex pairs (u, v) that have exactly 1 or 2 LCAs can also be solved in $O(n^\omega)$ time with high probability.*

This theorem appears as Theorems 26 and 27 in the main text. By our equivalence theorem, the same result applies to AP-AtLeast$(k+1)$-LCA and AP-AtMost$k$-LCA for $k = 1, 2$.

Our algorithm for AP-Exact1-LCA is different from that of [31]. The algorithm of [31] is deterministic while ours is randomized, so it is seemingly weaker. We nevertheless include our approach to AP-Exact1-LCA as it is simple and saves a factor of $\log n$. Additionally, our approach generalizes to AP-Exact2-LCA.

As our techniques no longer seem to work for the case of deciding if there are exactly 3 LCAs, we turn to conditional lower bounds. We prove that under popular fine-grained hypotheses, the following hold in the word-RAM model with $O(\log n)$ bit words (Theorem 30): AP-Exact$k$-LCA requires time $n^{2.5-o(1)}$ for $k = 3$, $n^{8/3-o(1)}$ for $k = 4$, $n^{2.8-o(1)}$ for $k = 5$ and $n^{3-o(1)}$ for $k = 6$.

With our earlier equivalence theorem in mind, our conditional lower bound for AP-Exact3-LCA means that detecting for each pair whether it has at least 4 LCAs, or listing 4 LCAs per vertex pair also requires $n^{2.5-o(1)}$ time. In particular, this shows that listing 4 LCAs is more difficult than listing just one LCA per vertex pair, as the latter has an $O(n^{2.447})$ time algorithm [25].

Furthermore, our conditional lower bound for AP-Exact6-LCA also implies that AP-AtLeast7-LCA requires $n^{3-o(1)}$ time, and hence the clearly even harder problem of listing 7 LCAs per vertex pair requires $n^{3-o(1)}$ time. This is intriguing since as we mentioned earlier, we can list all LCAs per pair in essentially cubic time if $\omega = 2$.

We also show the following algorithmic results for AP-List-2-LCA and AP-List-3-LCA.

▶ **Theorem 3.** *For $k = 2$ and $k = 3$, the* AP-List-k-LCA *problem can be deterministically solved in $\tilde{O}(n^{2+\lambda})$ time, where λ satisfies the equation $\omega(1, \lambda, 1) = 1 + 2\lambda$. Here, $\omega(1, \lambda, 1)$ is the exponent of multiplying an $n \times n^\lambda$ by an $n^\lambda \times n$ matrix.*

The running time for AP-List-k-LCA above matches the best known running time for Max-Witness product [18]. Using the current best bounds for rectangular matrix multiplication [33], the runtime we get for AP-List-k-LCA is $O(n^{2.529})$ for $k = 2$ and 3.

Counting LCAs. We now turn our attention to computing the number of LCAs for every pair of vertices in a DAG. We call this problem AP-#LCA. As shown in [20], we can list all LCAs for every pair of vertices in $O(n^{\omega(1,2,1)})$ time, which is essentially cubic time if $\omega = 2$. Thus in particular, we can also count all the LCAs in the same amount of time.

One might wonder, can the counts be computed faster, in truly subcubic time? We show that under the Strong Exponential Time (SETH) Hypothesis [29, 13, 14], this is impossible, even if we are only required to return the count for a vertex pair if it is smaller than some superconstant function $g(n)$. Notice that we can solve this restrained case in $O(n^3 g(n))$ time using the brute-force algorithm, so the following theorem is tight up to $n^{o(1)}$ factors when $g(n)$ is $\tilde{O}(1)$.

▶ **Theorem 4.** *Assuming SETH,* AP-#LCA *requires $n^{3-o(1)}$ time, even if we only need to return the minimum between the count and $g(n)$ for any $g(n) = \omega(1)$.*

The current best running time $O(n^{\omega(1,2,1)})$ for listing LCAs and also for AP-#LCA is actually supercubic, however. For the current best bounds on $\omega(1,2,1)$, it is $O(n^{3.252})$ [33]. In fact, there are serious limitations of the known matrix multiplication techniques [3, 15, 16] that show that current techniques cannot be used to prove that $\omega(1,2,1) < 3.05$.

In this case, the cubic lower bound for AP-#LCA under SETH would not be entirely satisfactory. We thus present a tight conditional lower bound from the 4-Clique problem.

The 4-Clique problem asks, does a given n-node graph contain a clique on 4 nodes? The fastest known algorithm for 4-Clique runs in $\tilde{O}(n^{\omega(1,2,1)})$ time [21], which has remained unchallenged for almost two decades. We show that an improvement over the $O(n^{\omega(1,2,1)})$ time for AP-#LCA would also solve 4-Clique faster.

▶ **Theorem 5.** *If the* AP-#LCA *problem can be solved in $T(n)$ time, then* 4-Clique *can be computed in $O(T(n) + n^{\omega})$ time.*

Verifying LCAs. Oftentimes in algorithms, one is also concerned with the problem of verifying an answer besides computing an answer. In many cases, verification is an easier problem than computation. For instance, even though computing the product of two $n \times n$ matrices A and B currently is only known to be possible in $O(n^{2.373})$ time, verifying whether the product of A and B is a matrix C can be done in randomized $\tilde{O}(n^2)$ time. This was the basis of the Blum-Luby-Rubinfeld linearity test [9].

We consider the following two verification variants of AP-LCA which we call Ver-LCA and AP-Ver-LCA. In both variants, we are given an n-node DAG, and for every pair of nodes u, v in the DAG, we are also given a node $w_{u,v}$. In Ver-LCA, we want to determine whether all $w_{u,v}$ are LCAs for their respective pair u, v, i.e. that the matrix w of candidate LCAs is all correct (or conversely, that there is *some* pair that has an incorrect entry). In the AP-Ver-LCA variant we want to know for every u, v whether $w_{u,v}$ is an LCA of u and v, so this variant is potentially more difficult. After we compute the transitive closure of the graph, it takes $O(n)$ time to verify whether a vertex $w_{u,v}$ is indeed an LCA of u and v. Thus, both Ver-LCA and AP-Ver-LCA can be solved in $O(n^3)$ time. No faster algorithm is known to the best of our knowledge.

Kowaluk and Lingas [31] solved a variant of AP-Ver-LCA concerning vertex pairs that have at most 2 LCAs. Specifically, given one or two nodes per pair they showed how to verify that those nodes are all the LCAs for the pair, in $O(n^{\omega})$ time. However, their algorithm is not able to compute 2 LCAs for vertex pairs that have exactly 2 LCAs in $O(n^{\omega})$ time.

Surprisingly, we provide strong evidence that Ver-LCA and AP-Ver-LCA are actually *harder* than AP-LCA, as AP-LCA can be solved in $O(n^{2.5-\varepsilon})$ time for $\varepsilon > 0$, while under popular fine-grained hypotheses, Ver-LCA and AP-Ver-LCA require $n^{2.5-o(1)}$ time.

Our first hardness result is that the running time of AP-Ver-LCA is at least as high as that of the Max-Witness problem, whose current best running time is $O(n^{2.529})$ [30, 33]. If $\omega = 2$, then Max-Witness would be solvable in $\tilde{O}(n^{2.5})$ time, and it is hypothesized [34] that no $n^{2.5-o(1)}$ time algorithms exist for it.

▶ **Theorem 6.** *If the* AP-Ver-LCA *problem can be solved in $T(n)$ time, then the* Max-Witness *problem can be solved in $\tilde{O}(T(n))$ time.*

Note that Czumaj, Kowaluk and Lingas's algorithm [18] for AP-LCA is essentially a reduction from AP-LCA to Max-Witness. Combined with their algorithm, the above theorem says that we can solve AP-Ver-LCA in $T(n)$ time, then we can solve AP-LCA in $\tilde{O}(T(n))$ time.

Our second result is the hardness of Ver-LCA based on the hardness of the $(4, 3)$-Hyperclique problem: given a 3-uniform hypergraph on n nodes, return whether it contains a 4-hyperclique. This problem is hypothesized to require $n^{4-o(1)}$ time [35], and solving it in $O(n^{4-\varepsilon})$ time for $\varepsilon > 0$ would imply improved algorithms for Max-3-SAT and other problems (see [35] and the discussion therein).

▶ **Theorem 7.** *Assuming the* $(4, 3)$-Hyperclique *hypothesis,* Ver-LCA *requires $n^{2.5-o(1)}$ time.*

Thus, verifying candidate LCAs is most likely harder than finding LCAs, defying the common intuition that verification should be easier than computation.

1.2 Paper Organization

In Section 2, we give necessary definitions. In Section 3, we list basic relationships among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA, including Theorem 1. In Section 4, we show $O(n^{\omega})$ time algorithms for AP-Exact1-LCA and AP-Exact2-LCA, proving Theorem 2. In Section 5, we consider the AP-List-k-LCA problem. In Section 6, we prove several conditional lower bounds for AP-Exact$k$-LCA and AP-#LCA, including Theorem 4 and Theorem 5. In Section 7, we show conditional lower bounds for AP-Ver-LCA, proving Theorem 6 and Theorem 7. Finally, in Section 8, we conclude with several open problems.

2 Preliminaries

2.1 Notation

Let $G = (V, E)$ be a DAG. For every $u, v \in V$, we use $\mathsf{LCA}(u, v)$ to denote the set of vertices that are LCAs for vertex pair u and v. We use $u \rightsquigarrow v$ to denote that u can reach v via zero or more edges and use $u \not\rightsquigarrow v$ to denote that u cannot reach v. In particular, $u \rightsquigarrow u$ for every $u \in V$. We also use $\mathsf{Anc}(u)$ to denote the set of vertices that can reach u. For any $V' \subseteq V$, we use $G[V']$ to denote the subgraph in G induced by the vertex set V'.

We use $\omega < 2.37286$ to denote the matrix multiplication exponent [4]. For any constants $a, b, c \geq 0$, we use $\omega(a, b, c)$ to denote the exponent of multiplying an $n^a \times n^b$ matrix by an $n^b \times n^c$ matrix, in the arithmetic circuit model. Note that the fastest known algorithms for square [4] and rectangular [33] matrix multiplication all work in the arithmetic circuit model.

It is well-known that $\omega(a, b, c) = \omega(b, c, a)$ (see e.g. [12]).

2.2 Variants of AP-LCA

Given a DAG $G = (V, E)$, we study the following variants of AP-LCA.

▶ **Definition 8** (AP-Exact$k$-LCA)**.** *Decide if $|\mathsf{LCA}(u, v)| = k$ for every pair $u, v \in V$.*

▶ **Definition 9** (AP-AtMost$k$-LCA)**.** *Decide if* $|\mathsf{LCA}(u,v)| \le k$ *for every pair* $u, v \in V$*.*

▶ **Definition 10** (AP-AtLeast$k$-LCA)**.** *Decide if* $|\mathsf{LCA}(u,v)| \ge k$ *for every pair* $u, v \in V$*.*

▶ **Definition 11** (AP-#LCA)**.** *Compute* $|\mathsf{LCA}(u,v)|$ *for every pair* $u, v \in V$*.*

▶ **Definition 12** (AP-List-k-LCA)**.** *Compute for every pair* $u, v \in V$ *a list of* k *distinct LCAs. If any pair* $u, v \in V$ *has fewer than* k *LCAs, output all of their LCAs.*

▶ **Definition 13** (AP-All-LCA)**.** *For every pair* $u, v \in V$*, output* $\mathsf{LCA}(u,v)$*.*

▶ **Definition 14** (AP-Ver-LCA)**.** *Given a candidate vertex* $w_{u,v}$ *for each pair* $u, v \in V$*, decide if* $w_{u,v} \in \mathsf{LCA}(u,v)$ *for every pair* $u, v \in V$*.*

▶ **Definition 15** (Ver-LCA)**.** *Given a candidate vertex* $w_{u,v}$ *for each pair* $u, v \in V$*, decide if there exists* $u, v \in V$ *such that* $w_{u,v}$ *is not an LCA for* u *and* v*.*

2.3 Fine-Grained Hypotheses

In this section, we list the hypotheses we use in this paper.

Eisenbrand et al. [21] gave the current best algorithm for 4-Clique that runs in $O(n^{\omega(1,2,1)})$ time. The 4-Clique hypothesis states that we cannot improve this algorithm much.

▶ **Hypothesis 16** (4-Clique Hypothesis [11, 1])**.** *On a Word-RAM with* $O(\log n)$ *bit words, detecting a 4-clique in an* n*-node graph requires* $n^{\omega(1,2,1)-o(1)}$ *time.*

▶ **Hypothesis 17** ((ℓ, k)-Hyperclique Hypothesis, [35])**.** *Let* $\ell > k > 2$ *be constant integers. On a Word-RAM with* $O(\log n)$ *bit words, detecting whether an* n*-node* k*-uniform hypergraph contains an* ℓ*-hyperclique requires* $n^{\ell-o(1)}$ *time.*

Using common techniques (see e.g. [39]), the (ℓ, k)-Hyperclique hypothesis actually implies the hardness of the following unbalanced version of (ℓ, k)-Hyperclique.

▶ **Fact 18.** *Assuming the* (ℓ, k)-Hyperclique *hypothesis, on a Word-RAM with* $O(\log n)$ *bit words, detecting whether a* k*-uniform* ℓ*-partite hypergraph with* $n^{a_1}, \dots, n^{a_\ell}$ *vertices on each part for* $a_1, \dots, a_\ell > 0$ *requires* $n^{a_1+\cdots+a_\ell-o(1)}$ *time.*

▶ **Hypothesis 19** (Max-k-SAT Hypothesis, [35])**.** *On a Word-RAM with* $O(\log n)$ *bit words, for any* $k \ge 3$*, given a* k*-CNF formula on* n *variables and* $\mathrm{poly}(n)$ *clauses, determining the maximum number of clauses that can be satisfied by a Boolean assignment of the variables requires* $2^{n-o(n)}$ *time.*

▶ **Hypothesis 20** (Strong Exponential Time Hypothesis (SETH), [28, 13, 14])**.** *On a Word-RAM with* $O(\log n)$ *bit words, for every* $\epsilon > 0$*, there exists* k *such that* k-SAT *on* n *variables cannot be solved in* $O(2^{(1-\epsilon)n})$ *time.*

▶ **Definition 21.** *The* Max-Witness *product* C *of two* $n \times n$ *Boolean matrices* A *and* B *is defined as*

$$C[i,j] = \max\{k \mid A[i,k] = B[k,j] = 1\}$$

where the maximum is defined to be $-\infty$ *if no such witness exists.*

The best running time to compute the Max-Witness product is $O(n^{2+\lambda})$ where λ satisfies the equation $\omega(1,\lambda,1) = 1 + 2\lambda$ [18]. This running time is $\tilde{O}(n^{2.5})$ if $\omega = 2$. It is used as a hypothesis that this running time cannot be improved much.

▶ **Hypothesis 22** (Max-Witness Hypothesis, [34])**.** *On a Word-RAM with* $O(\log n)$ *bit words, computing the* Max-Witness *product of two* $n \times n$ *matrices requires* $n^{2.5-o(1)}$ *time.*

3 Relationships among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA

In this section, we consider the relationships between AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA. Our results are depicted in Figure 1 and are proven in the full version of the paper.

$$\begin{tikzcd}
\text{AP-AtLeast1-LCA} \arrow[r] \arrow[d, leftrightarrow] & \text{AP-AtLeast2-LCA} \arrow[r] \arrow[d, leftrightarrow] & \text{AP-AtLeast3-LCA} \arrow[r] \arrow[d, leftrightarrow] & \cdots \\
\text{AP-Exact0-LCA} \arrow[r] \arrow[d, leftrightarrow] & \text{AP-Exact1-LCA} \arrow[r] \arrow[d, leftrightarrow] & \text{AP-Exact2-LCA} \arrow[r] \arrow[d, leftrightarrow] & \cdots \\
\text{AP-AtMost0-LCA} \arrow[r] & \text{AP-AtMost1-LCA} \arrow[r] & \text{AP-AtMost2-LCA} \arrow[r] & \cdots
\end{tikzcd}$$

Figure 1 Reductions between AP-AtMost$k$-LCA, AP-Exact$k$-LCA and AP-AtLeast$k$-LCA. All arrows in this figure represent $O(n^2)$ time reductions from an instance to another instance with the same input sizes up to constant factors.

We first show the following lemma (whose proof is deferred to the full version) which then allows us to show that AP-Exact$(k+1)$-LCA (resp. AP-AtMost$(k+1)$-LCA, AP-AtLeast$(k+1)$-LCA) is harder than AP-Exact$k$-LCA (resp. AP-AtMost$k$-LCA, AP-AtLeast$k$-LCA) for $k \geq 0$.

▶ **Lemma 23.** *Given a DAG G with n vertices, we can create another DAG G' with $2n+1$ vertices and a map $\rho : V(G) \to V(G')$ in $O(n^2)$ time such that for every $u, v \in V(G)$, the number of LCAs of u and v in G is exactly one fewer than the number of LCAs of $\rho(u)$ and $\rho(v)$ in G'.*

▶ **Corollary 24.** *For any $k \geq 0$, an instance of* AP-Exact$k$-LCA *(resp.* AP-AtMost$k$-LCA, AP-AtLeast$k$-LCA*) with n vertices reduces to an instance of* AP-Exact$(k+1)$-LCA *(resp.* AP-AtMost$(k+1)$-LCA, AP-AtLeast$(k+1)$-LCA*) with $O(n)$ vertices in $O(n^2)$ time.*

Finally, we recall the relationship among AP-Exact$k$-LCA, AP-AtMost$k$-LCA and AP-AtLeast$k$-LCA.

▶ **Theorem 1.** *For any constant $k \geq 0$, the running times of* AP-Exact$k$-LCA, AP-AtMost$k$-LCA *and* AP-AtLeast$(k+1)$-LCA *are the same up to constant factors.*

4 Algorithms for AP-Exact$k$-LCA

As noted in the introduction, AP-Exact$k$-LCA can be solved in $O(n^3)$ time for any constant k. Interestingly, an algorithm by Kowaluk and Lingas [31] that finds and verifies the LCAs for vertex pairs with a unique LCA implies that AP-Exact1-LCA can be solved in $\tilde{O}(n^\omega)$ time deterministically. In this section, we present an alternative randomized algorithm for AP-Exact1-LCA, and also extend the algorithm for AP-Exact2-LCA.

The following claim is essential to our AP-Exact1-LCA algorithm. We defer its proof to the full version fo the paper.

▷ Claim 25. Given a DAG $G = (V, E)$, for every pair of vertices $u, v \in V$, we have that

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \bigcup_{w \in \mathsf{LCA}(u,v)} \mathsf{Anc}(w). \tag{1}$$

Moreover, if $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \bigcup_{w \in S} \mathsf{Anc}(w)$ for some $S \subseteq V$, it must be the case that $\mathsf{LCA}(u, v) \subseteq S$.

▶ **Theorem 26.** *There exists an $O(n^\omega)$ time Las Vegas algorithm for* AP-Exact1-LCA *that succeeds with high probability. Additionally, this algorithm can find the unique LCA for all pairs of vertices that have exactly 1 LCA.*

Proof. For every pair of vertices u and v with a unique LCA w, we rewrite Equation (1) as $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(w)$. In fact, Claim 25 gives us that this holds if and only if w is a unique LCA of the pair u and v.

Let $f : V \to \mathbb{Z}_p$ be a random function for some $p = \Theta(n^{10})$. For every $S \subseteq V$, we will use $f(S)$ to denote $\sum_{x \in S} f(x)$. Then with high probability, for any $u, v, x \in V$,

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(x) \quad \text{if and only if} \quad f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v)) = f(\mathsf{Anc}(x)).$$

To see this, note that for $S, S' \subseteq V$, if $S \neq S'$, then $f(S) - f(S')$ is a sum of a nonzero number of independent uniform random variables from $\mathbb{Z}_p$. Thus if $S \neq S'$, then $\Pr[f(S) = f(S')] = \frac{1}{p}$. Since we are comparing $O(n^2)$ such sets of the form $f(\mathsf{Anc}(x))$ and $f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$, by a union bound, the probability that two distinct sets collide is $O(n^4/p)$.

Therefore, it suffices to compute $f(\mathsf{Anc}(x))$ and $f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ for all $u, v, x \in V$. For each $x \in V(G)$, it is easy to compute $f(\mathsf{Anc}(x)) = \sum_{v \in \mathsf{Anc}(x)} f(v)$ in $O(n)$ time. To compute $F(u, v) = f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ for all $u, v \in V$, we construct the following matrices. Let A be the transitive closure of G and let $B[x, v] = f(x) \cdot A[x, v]$. Now, note that the (u, v)-th entry of $C = A^T B$ gives us $C[u, v] = \sum_{x \in \mathsf{Anc}(u) \cap \mathsf{Anc}(v)} f(x) = F(u, v)$, as desired. Therefore, we can compute all $F(u, v)$ in $O(n^\omega)$ time.

Now, we sort the list $L = \{f(v) \mid v \in V(G)\}$ in $\tilde{O}(n)$ time. For each $u, v \in V$, we can find an arbitrary $w_{u,v}$ such that $F(u, v) = f(w_{u,v})$ in $\tilde{O}(1)$ time. Assuming none of the $O(n^2)$ sets we are interested in collide, which happens with probability at least $1 - O(n^4/p) = 1 - O(1/n^6)$, we find such a $w_{u,v}$ if and only if it is the unique LCA of $u, v \in V$.

To make this algorithm Las Vegas, we first notice that if our algorithm does not report a $w_{u,v}$, then u and v does not have a unique LCA. For the vertex pairs that our algorithm does find a $w_{u,v}$, we run [31]'s verification algorithm (Theorem 2 in [31]) to verify if each $w_{u,v}$ is in fact the unique LCA of u, v in $O(n^\omega)$ time. If we find any errors, we can simply repeat the algorithm. ◀

Now we show how to extend our AP-Exact1-LCA algorithm to AP-Exact2-LCA.

▶ **Theorem 27.** *There exists an $O(n^\omega)$ time Las Vegas algorithm for* AP-Exact2-LCA *that succeeds with high probability. Additionally, this algorithm can find the two LCAs for all pairs of vertices with exactly 2 LCAs.*

Proof. For all pair of vertices u and v with exactly two LCAs, say a and b, we rewrite (1) as $\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(a) \cup \mathsf{Anc}(b)$. Moreover, for any u, v, a, b such that the above equation holds, it must be the case that either both a and b are the only LCAs of u and v, or exactly one of them is the unique LCA (and the other is a common ancestor). We can detect the latter case with high probability by performing the algorithm as described in Theorem 26.

Let $f : V(G) \to \mathbb{Z}_p$ be a random function for some $p = \Theta(n^{10})$. By the same argument as Theorem 26, with high probability, for any $u, v, a, b \in V$,

$$\mathsf{Anc}(u) \cap \mathsf{Anc}(v) = \mathsf{Anc}(a) \cup \mathsf{Anc}(b) \quad \text{if and only if} \quad f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v)) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b)).$$

Let $F(u, v) = f(\mathsf{Anc}(u) \cap \mathsf{Anc}(v))$ and $H(a, b) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b))$. As we saw in Theorem 26, we can compute $F(u, v)$ in $O(n^\omega)$ time.

To compute $H(a, b)$, note that $\mathsf{Anc}(a) \cup \mathsf{Anc}(b) = \overline{\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}}$. First, we compute the transitive closure A of G in $O(n^\omega)$ time. Then, we construct an $n \times n$ matrix M by setting $M[x, a] = 1 - A[x, a]$. Now, construct another matrix N by setting $N[x, b] = f(x) \cdot M[x, b]$. Then, it is easy to see that

$$(M^T N)[a, b] = \sum_{x \in \overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}} f(x) = f(\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}).$$

Therefore, one can compute

$$H(a, b) = f(\mathsf{Anc}(a) \cup \mathsf{Anc}(b)) = f(V) - f(\overline{\mathsf{Anc}(a)} \cap \overline{\mathsf{Anc}(b)}) = f(V) - (M^T N)[a, b]$$

for all $a, b \in V$ in $O(n^\omega)$ time.

Now, sort $L = \{H(a, b) \mid a, b \in V\}$. For each u, v which does not have a unique LCA, search for an arbitrary pair $a_{u,v}, b_{u,v}$ (if one exists) such that $F(u, v) = H(a_{u,v}, b_{u,v})$ in $\tilde{O}(1)$ time. With probability $1 - O(1/n^6)$, we find such a pair for each u, v if and only if $a_{u,v}$ and $b_{u,v}$ are the only two LCAs of u and v.

To make this algorithm Las Vegas, we first notice that if our algorithm does not report a pair $a_{u,v}, b_{u,v}$, then u and v does not have exactly two LCAs. For vertex pairs for which our algorithm does find two LCA candidates, we run [31]'s verification algorithm (it is described in a remark in [31]) to verify that $a_{u,v}$ and $b_{u,v}$ are the only two LCAs of u and v in $O(n^\omega)$ time. If we find any errors, we can simply repeat the algorithm from the beginning. ◀

Note that our technique for AP-Exact1-LCA and AP-Exact2-LCA does not extend to AP-Exact3-LCA because it would require us to list $f(\mathsf{Anc}(x) \cup \mathsf{Anc}(y) \cup \mathsf{Anc}(z))$ for all $x, y, z \in V$, which easily exceeds n^ω time. In fact, in Section 6, we show it is unlikely to obtain an $\tilde{O}(n^\omega)$ time algorithm for AP-Exact3-LCA by proving that any $O(n^{2.5-\epsilon})$ time algorithm for $\epsilon > 0$ for AP-Exact3-LCA would refute the $(4, 3)$-Hyperclique hypothesis. Thus, AP-Exact3-LCA is indeed (conditionally) harder than AP-Exact1-LCA and AP-Exact2-LCA.

5 AP-LCA Listing Algorithms

In this section, we consider the AP-List-k-LCA problem. First, we show that AP-AtLeast$k$-LCA and AP-List-k-LCA are *subcubically equivalent*, i.e. either both or neither have a truly subcubic time algorithm.

▶ **Theorem 28.** *Suppose* AP-AtLeast$k$-LCA *can be computed in $T(n)$ time for a constant k. Then,* AP-List-k-LCA *can be computed in $O(\sqrt{n^3 \cdot T(n)})$ time. In particular,* AP-AtLeast$k$-LCA *and* AP-List-k-LCA *are subcubically equivalent.*

Proof. Suppose we are given a DAG $G = (V, E)$. First compute a topological ordering π of the vertices in $O(n^2)$ time, and the transitive closure D in $O(n^\omega)$ time. Now, for every pair of vertices u and v, we inductively find their k topologically latest (with respect to π) LCAs.

Suppose we have found the set $S(u, v)$ of the topologically latest $\ell - 1$ LCAs for every pair of vertices u, v, for some $1 \le \ell \le k$ with respect to π. Now, partition the vertices into sets $V = V_1 \sqcup V_2 \sqcup \cdots \sqcup V_{n/L}$, where V_1 contains the first L vertices in the topological ordering, V_2 contains the next L and so on for a parameter L that we will set later.

Let $\mathsf{LCA}_{G[W]}(u, v)$ denote the set of LCAs of u and v in the subgraph induced by W (note the distinction between $\mathsf{LCA}_{G[W]}(u, v)$ and $\mathsf{LCA}(u, v) \cap W$). Consider the vertex set $U_i = V_i \sqcup V_{i+1} \sqcup \cdots \sqcup V_{n/L}$ and the induced subgraph $G_i = G[U_i]$. We will prove the following claim in the full version of the paper.

▷ Claim 29. For $u, v \in U_i$, it must be the case that $\mathsf{LCA}_{G_i}(u, v) = \mathsf{LCA}(u, v) \cap U_i$.

Now we describe our algorithm. For $i = n/L, n/L - 1, \dots, 1$, run AP-AtLeast$\ell$-LCA on each G_i. For each $(u, v) \in V \times V$, keep track of the largest index $i_{u,v}$ where AP-AtLeastℓ-LCA outputs 1, i.e. largest index such that $|\mathsf{LCA}_{G_i}(u, v)| \geq \ell$. By Claim 29, this must mean that $|\mathsf{LCA}(u, v) \cap U_{i_{u,v}}| \geq \ell$ whereas $|\mathsf{LCA}(u, v) \cap U_{i_{u,v}-1}| < \ell$. In other words, the ℓth LCA lies in $V_{i_{u,v}}$. By Corollary 24, we can compute AP-AtLeastℓ-LCA in time $O(T(n))$. Therefore, this step takes $O\left(\frac{n}{L} \cdot T(n)\right)$ time in total.

Next, for each $u, v \in V$, note that the topologically ℓth LCA must lie in the vertex partition $V_{i_{u,v}}$, if $i_{u,v}$ exists. Therefore, it suffices to find the latest vertex $x \in V_{i_{u,v}}$ such that $x \in \mathsf{Anc}(u) \cap \mathsf{Anc}(v)$ and no $y \in S(u, v)$ is a descendent of x. Such an x must in fact be the ℓth LCA. Note that these checks can be done in $O(1)$ time for each $x \in V_{i_{u,v}}$ using the transitive closure D. If there is no $i_{u,v}$ such that AP-AtLeastℓ-LCA outputs 1, then u and v have fewer than ℓ LCAs. This step takes $O(\ell \cdot L) = O(L)$ time for each pair $u, v \in V$.

Since we have to iteratively find up to k LCAs per vertex pair, the overall runtime of the algorithm is $O(n^\omega + k(\frac{n}{L} \cdot T(n) + n^2 \cdot L))$. Choosing $L = \sqrt{T(n)/n}$, we have a runtime of $O(\sqrt{n^3 \cdot T(n)})$.

Moreover, it is clear that if there is a subcubic algorithm for AP-List-k-LCA, we can use the same algorithm to solve AP-AtLeast$k$-LCA with an $\tilde{O}(n^2)$ additional cost. Therefore the two problems are in fact subcubically equivalent. ◀

In Theorem 26 and Theorem 27, we showed $O(n^\omega)$ time algorithms for AP-Exact1-LCA and AP-Exact2-LCA. By their equivalences with AP-AtLeast2-LCA and AP-AtLeast3-LCA respectively, we can also solve AP-AtLeast2-LCA and AP-AtLeast3-LCA in $O(n^\omega)$ time. By Theorem 28, these imply $O(n^{(\omega+3)/2})$ time algorithms for AP-List-2-LCA and AP-List-3-LCA.

In the following theorem, we show that we can further improve the $O(n^{(3+\omega)/2})$ running time for AP-List-2-LCA and AP-List-3-LCA to $\tilde{O}(n^{2+\lambda})$ time where $\omega(1, \lambda, 1) = 1 + 2\lambda$. Interestingly this running time matches the current best running time of the Max-Witness problem [18]. For these algorithms, we use an idea from [31] about comparing the sizes of two sets for verifying whether a set of one or two vertices are all the LCAs.

▶ **Theorem 3.** *For $k = 2$ and $k = 3$, the* AP-List-k-LCA *problem can be deterministically solved in $\tilde{O}(n^{2+\lambda})$ time, where λ satisfies the equation $\omega(1, \lambda, 1) = 1 + 2\lambda$. Here, $\omega(1, \lambda, 1)$ is the exponent of multiplying an $n \times n^\lambda$ by an $n^\lambda \times n$ matrix.*

We defer the proof of Theorem 3 to the full version of the paper.

6 Lower Bounds

In this section, we show our conditional lower bounds for AP-Exact$k$-LCA and AP-#LCA. These lower bounds are the first conditional lower bounds for LCA problems that are higher than $n^{\omega - o(1)}$.

6.1 Lower Bounds for AP-Exact$k$-LCA

First, we show lower bounds for the AP-Exact$k$-LCA problem by reducing from 3-uniform hypercliques. Combined with Corollary 24, the following theorem also shows that, for all constant $k \geq 6$, AP-Exact$k$-LCA requires $n^{3-o(1)}$ time.

▶ **Theorem 30.** *Assuming the* $(4,3)$-Hyperclique *hypothesis,* AP-Exact3-LCA *requires* $n^{2.5-o(1)}$ *time. Assuming the* $(5,3)$-Hyperclique *hypothesis,* AP-Exact4-LCA *and* AP-Exact6-LCA *require* $n^{8/3-o(1)}$ *and* $n^{3-o(1)}$ *time respectively. Also, assuming the* $(6,3)$-Hyperclique *hypothesis,* AP-Exact5-LCA *requires* $n^{14/5-o(1)}$ *time.*

Proof. All four reductions share the same underlying ideas. Thus, we only give full details for the first reduction. The remaining reductions are deferred to the full version of the paper.

$(4,3)$-Hyperclique $\longrightarrow$ AP-Exact**3**-LCA. Suppose we are given a 3-uniform 4-partite hypergraph G on vertex sets A, B, C, U, where $|A| = |B| = |C| = \sqrt{n}$, and $|U| = n$. By Fact 18, the $(4,3)$-Hyperclique hypothesis implies that it requires $(|A||B||C||U|)^{1-o(1)} = n^{2.5-o(1)}$ time to determine whether G contains a 4-hyperclique.

We construct the following instance of AP-Exact3-LCA as depicted in Figure 2. The graph G' contains 3 layers of vertices V_1, V_2, V_3. Vertex set V_1 is a copy of U, vertex set V_2 equals $(A \times B) \sqcup (B \times C) \sqcup (C \times A)$ and vertex set V_3 equals $(A \times B) \sqcup C$. To distinguish vertices from V_2 and V_3, we use subscript 2 and 3, e.g. $(a,b)_2$ and $(a,b)_3$, to denote vertices from V_2 and V_3 respectively.

We also add the following edges to the graph G':

- Add a directed edge from every vertex in V_1 to every vertex in V_3.
- Add a directed edge from any vertex in V_2 to any vertex in V_3 as long as they do not have *inconsistent* labels. For instance, for every $a \in A, b \in B, c \in C$, we add an edge from $(a,b)_2$ to $(a,b)_3$ and to c_3, but we do not add an edge from $(a,b)_2$ to $(a,b')_3$ if $b \neq b'$.
- For every $u \in V_1$ and every $(x,y)_2 \in V_2$, add a directed edge from u to (x,y) if and only if there is *not* a 3-hyperedge among u, x and y.

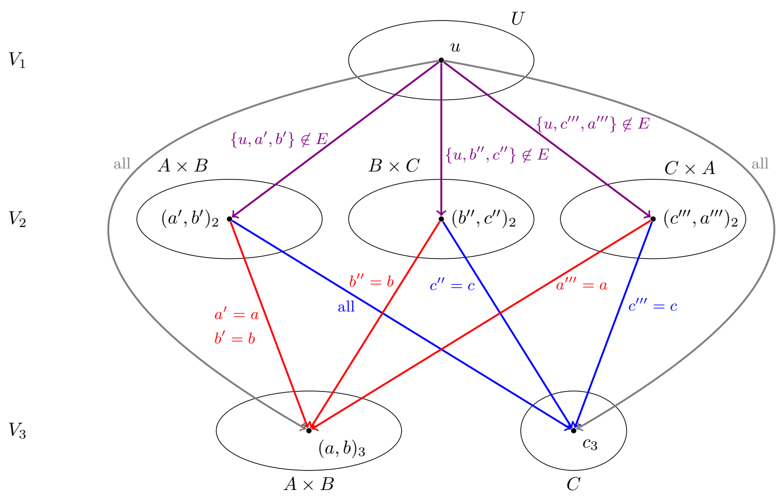


Figure 2 Construction of G' in Theorem 30 from the 3-uniform 4-hyperclique instance. Between the parts where we mark "all", we add all possible edges. Between the parts where we mark a condition, we only add an edge when the corresponding condition holds.

We consider the set of LCAs for every pair of $(a,b)_3, c_3 \in V_3$.

First, since G' is a three layered graph, all common ancestors of $(a,b)_3$ and c_3 in V_2 are their LCAs. Since we only add edges from V_2 to V_3 when the labels are consistent, it is easy to verify that the set of LCAs in V_2 is $\{(a,b)_2, (b,c)_2, (c,a)_2\}$.

Now we claim that a, b, c are in a 4-hyperclique in G if and only if $(a,b)_3$ and c_3 have an LCA in V_1 and there is a 3-hyperedge among a, b, c in G.

Suppose a, b, c are in a 4-hyperclique with a vertex $u \in V(G)$. Since G is 4-partite, we must have $u \in U$. The copy of u in G' is clearly a common ancestor of $(a,b)_3$ and c_3, since we add all possible edges from V_1 to V_3. Because a, b, c, u is in a 4-hyperclique, $\{a,b,u\}, \{b,c,u\}, \{c,a,u\} \in E(G)$. Therefore, in G' we do not add edges from u to any of $(a,b)_2, (b,c)_2$ and $(c,a)_2$. Since these are the only common ancestors of $(a,b)_3$ and c_3 in V_2, u in fact cannot reach any other vertex that can reach both $(a,b)_3$ and c_3, which makes u an LCA. Clearly, there is a 3-hyperedge among a, b, c in G.

To prove the converse, suppose $u \in V_1$ is an LCA of $(a,b)_3$ and c_3 and there is a 3-hyperedge among a, b, c in G. In that case, u cannot reach any vertex that can reach both $(a,b)_3$ and c_3. In particular, u cannot reach any of $(a,b)_2, (b,c)_2, (c,a)_2$. When we add edges from V_1 to V_2, we have that $\{a,b,u\}, \{b,c,u\}, \{c,a,u\}$ are all 3-hyperedges in G. Also, since $\{a,b,c\}$ is a 3-hyperedge, there is indeed a 4-hyperclique with vertices a, b, c, u.

Thus, a, b, c are in a 4-hyperclique in G if and only if the number of LCAs of $(a,b)_3$ and c_3 is not 3 and there is a 3-hyperedge among a, b, c in G. Thus, given the result of an AP-Exact3-LCA computation of G', we can easily determine if G has a 4-hyperclique. Therefore, assuming the $(4,3)$-Hyperclique hypothesis, AP-Exact3-LCA requires $n^{2.5-o(1)}$ time. ◀

▶ Remark 31. Note that in all our reductions to AP-Exact$k$-LCA for $3 \le k \le 5$, we only need to output the results for $o(n^2)$ pairs of u and v. For instance, in the reduction from $(4,3)$-Hyperclique to AP-Exact3-LCA, we only need to output whether (u,v) has exactly 3 LCAs for $u \in A \times B$ and $v \in C$. The total number of such pairs is only $O(n^{1.5})$. This is the main reason why we do not get $n^{3-o(1)}$ conditional lower bounds for AP-Exact$k$-LCA for $3 \le k \le 5$. On the other hand, in the reduction to AP-Exact6-LCA, we do have $\Theta(n^2)$ queries.

Williams [41] showed that Max-3-SAT reduces to 3-uniform hypercliques. Lincoln, Vassilevska Williams and Williams [35] further generalized this reduction to a reduction from Constraint Satisfaction Problem (CSP) on degree-3 formulas to 3-uniform hypercliques. Therefore, Theorem 30 also works assuming the Max-3-SAT hypothesis or the hardness of maximizing the number of satisfying clauses in degree-3 CSP formulas.

▶ **Corollary 32.** *Assuming* Max-3-SAT *(or even max degree 3 CSP formulas) on N variables and $\mathrm{poly}(n)$ clauses requires $2^{N-o(N)}$ time,* AP-Exact3-LCA*,* AP-Exact4-LCA*,* AP-Exact5-LCA *and* AP-Exact6-LCA *requires $n^{2.5-o(1)}$, $n^{8/3-o(1)}$, $n^{14/5-o(1)}$ and $n^{3-o(1)}$ time respectively.*

6.2 Lower Bounds for Counting LCAs

In this section, we show two conditional lower bounds for AP-#LCA, one based on SETH and one based on the 4-Clique hypothesis.

The next lemma is a crucial tool for the SETH lower bound. It is a generalization of our previous reduction from $(5,3)$-Hyperclique to AP-Exact6-LCA.

▶ **Lemma 33.** *If there exists a $T(N)$ time algorithm for* AP-Exact$\binom{2(k-1)}{k-1}$-LCA *for graphs with N vertices, then there exists an $O(f(k)\,\mathrm{poly}(n)^{f(k)}T(2^{n/3}))$ time algorithm for* Max-k-SAT *with n variables and $\mathrm{poly}(n)$ clauses for some function f.*

To prove the lemma, we first reduce Max-k-SAT to k-uniform $(2k-1)$-hyperclique, which is a straightforward generalization of Williams' Max-2-SAT algorithm [41]. Then we reduce k-uniform $(2k-1)$-hyperclique to AP-Exact$\binom{2(k-1)}{k-1}$-LCA, building on ideas similar to the proof of Theorem 30. The full proof can be found in the full version of the paper.

▶ Remark 34. Lemma 33 implies that if we assume the Max-k-SAT hypothesis, then AP-Exact$\binom{2(k-1)}{k-1}$-LCA requires $n^{3-o(1)}$ time. Since our reduction uses $(2k-1,k)$-Hyperclique as an intermediate problem, the same lower bound also holds assuming the $(2k-1,k)$-Hyperclique hypothesis.

Now we show our SETH lower bound using Lemma 33.

▶ **Theorem 4.** *Assuming SETH,* AP-#LCA *requires $n^{3-o(1)}$ time, even if we only need to return the minimum between the count and $g(n)$ for any $g(n)=\omega(1)$.*

Proof. For the sake of contradiction, assume AP-#LCA has an $O(n^{3-\epsilon})$ time algorithm for $\epsilon>0$ when the algorithm only needs to return the minimum between the count and $g(n)$. For any fixed k, when n is large enough, we have $\binom{2(k-1)}{k-1}<g(n)$, so we can solve AP-Exact$\binom{2(k-1)}{k-1}$-LCA in $O(n^{3-\epsilon})$ time. Thus, by Lemma 33, we can solve Max-k-SAT (and thus k-SAT) with n variables and poly(n) clauses in time

$$O(f(k)\,\mathrm{poly}(n)^{f(k)}(2^{n/3})^{3-\epsilon})=O(f(k)\,\mathrm{poly}(n)^{f(k)}2^{(1-\epsilon/3)n})=O(\mathrm{poly}(n)\cdot 2^{(1-\epsilon/3)n}),$$

which would refute SETH. ◀

Finally, we present our reduction from 4-Clique to AP-#LCA, showing an $n^{\omega(1,2,1)-o(1)}$ lower bound for AP-#LCA assuming the current algorithm for 4-Clique is optimal.

▶ **Theorem 5.** *If the* AP-#LCA *problem can be solved in $T(n)$ time, then* 4-Clique *can be computed in $O(T(n)+n^{\omega})$ time.*

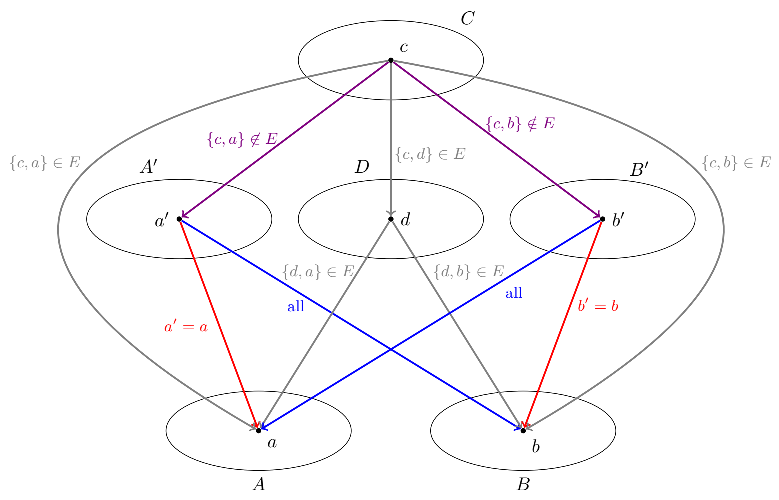
Proof. Suppose we are given a 4-Clique instance $G=(V,E)$. Without loss of generality, we assume G is a 4-partite graph with four vertex parts $V=A\sqcup B\sqcup C\sqcup D$ of size n each.

First, make a copy $G'=(V',E')$ of G, and modify the edge set of G' as follows:
- Remove all edges between A and B.
- Direct all edges from D to A and B.
- Direct all edges from C to A,B and D.

Then we add two additional vertex sets A' and B' to G', where A' is a copy of A and B' is a copy of B. We use a' to denote the copy of $a\in A$ in A' and use b' to denote the copy of $b\in B$ in B'. We also add the following edges:
- For every $a\in A$, add an edge (a',a).
- For every $b\in B$, add an edge (b',b).
- For every $a\in A, b\in B$, add two edges (a',b) and (b',a).
- For every $a\in A, c\in C$, add an edge (c,a') if $\{c,a\}\notin E$.
- For every $b\in B, c\in C$, add an edge (c,b') if $\{c,b\}\notin E$.

This construction of the graph is also depicted in Figure 3. From there, it is clear that G' is a 3-layered graph.

▷ Claim 35. For every $a\in A, b\in B, c\in C$, c is an LCA of a and b in G' if and only if $\{c,a\},\{c,b\}\in E$ and there doesn't exist any $d\in D$ such that $\{c,d\},\{d,a\},\{d,b\}\in E$.



■ **Figure 3** Construction of G' in Theorem 5 given a 4-partite 4-Clique instance. Between parts where we mark "all", we add all possible edges. Between parts where we mark a condition, we only add an edge when the corresponding condition holds.

Proof. First, suppose c is an LCA of a and b. For the sake of contradiction, suppose $\{c, a\} \notin E$. Then by the construction of G', $(c, a') \in E'$. Also, $(a', a), (a', b) \in E'$, so c cannot be an LCA. This leads to a contradiction, so we must have $\{c, a\} \in E$. Similarly, we must have $\{c, b\} \in E$. Finally, suppose for the sake of contradiction that there exists a $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$, then by construction, $(c, d), (d, a), (d, b) \in E'$, so c cannot be an LCA. Thus, there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$.

Now we prove the converse direction. Suppose $\{c, a\}, \{c, b\} \in E$ and there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$. By our construction, $(c, a), (c, b) \in E'$, so c is at least a common ancestor of a and b. Since G' is a 3-layered graph, it suffices to show that there isn't any vertex u in the middle layer such that $(c, u), (u, a), (u, b) \in E'$. First, for any $u \in A'$, if $u \neq a'$, then $(u, a) \notin E'$; if $u = a'$, then $(c, u) \notin E'$ because $\{c, a\} \in E$. Therefore, there isn't any $u \in A'$ such that $(c, u), (u, a), (u, b) \in E'$. Similarly, there isn't any $u \in B'$ such that $(c, u), (u, a), (u, b) \in E'$. For any $d \in D$, we already have the condition that at least one of $\{c, d\}, \{d, a\}, \{d, b\}$ is not in E, so at least one of $(c, d), (d, a), (d, b)$ is not in E'. Therefore, c is an LCA. ◁

Using this claim, we describe our algorithm below.

First, run AP-#LCA to compute $|\mathsf{LCA}(a, b)|$ for all $(a, b) \in A \times B$. Since G' is a three-layered graph, the set of LCAs of a and b in the middle layer is exactly the set of their common neighbors in the middle layer. Therefore, we can easily compute $|\mathsf{LCA}(a, b) \cap (A' \cup B' \cup D)|$ in $O(n^\omega)$ time by using matrix multiplication to count the number of their common neighbors in the middle layer. Also, clearly, there isn't any LCA of a and b in A or B. Thus, we can compute the number of $c \in C$ that is an LCA of a and b by

$$|\mathsf{LCA}(a, b) \cap C| = |\mathsf{LCA}(a, b)| - |\mathsf{LCA}(a, b) \cap (A' \cup B' \cup D)|.$$

By Claim 35, $|\mathsf{LCA}(a, b) \cap C|$ is exactly the number of $c \in C$ such that $\{c, a\}, \{c, b\} \in E$ and there doesn't exist any $d \in D$ such that $\{c, d\}, \{d, a\}, \{d, b\} \in E$.

Next, in $O(n^\omega)$ we can use matrix multiplication again to compute $Q(a, b)$ for every (a, b) where $Q(a, b)$ is defined as the number of $c \in C$ such that $\{c, a\}, \{c, b\} \in E$.

Note that $Q(a,b) - |\mathsf{LCA}(a,b) \cap C|$ is exactly the number of $c \in C$ such that $\{c,a\}, \{c,b\} \in E$ and there *exists* $d \in D$ such that $\{c,d\}, \{d,a\}, \{d,b\} \in E$. Thus, a and b are in a 4-clique if and only if $\{a,b\} \in E$ and $Q(a,b) - |\mathsf{LCA}(a,b) \cap C| > 0$.

Overall, if we can compute AP-#LCA in $T(n)$ time, then we can solve the 4-partite 4-Clique instance in $O(T(n) + n^{\omega})$ time. ◀

Since AP-#LCA is easier than AP-All-LCA, this lower bound shows that the AP-All-LCA algorithm in [20] is in fact conditionally optimal.

7 AP-Ver-LCA

In this section, we show two conditional lower bounds for AP-Ver-LCA, based on the Max-Witness hypothesis and the $(4,3)$-Hyperclique hypothesis. First, recall the following theorem:

▶ **Theorem 6.** *If the* AP-Ver-LCA *problem can be solved in $T(n)$ time, then the* Max-Witness *problem can be solved in $\tilde{O}(T(n))$ time.*

At the high level, we reduce the MaxWitness problem to $O(\log n)$ calls of the AP-Ver-LCA problem using a parallel binary search technique.

Proof. Without loss of generality, suppose $n = 2^{\ell}$ for some integer ℓ. Suppose we have two $n \times n$ Boolean matrices A and B, each already padded with a column and row of ones to ensure that there always exists a Boolean witness. Now, we will describe an algorithm to compute $C = \mathsf{Max\text{-}Witness}(A,B)$ using an AP-Ver-LCA algorithm $\ell = \log n$ times. At the high level, we will be using a parallel binary search to find the maximum witness corresponding to each entry of C.

Construct a tripartite graph G on vertices $V = I \sqcup J \sqcup K$, where $|I| = |J| = |K|$ and identify each of the sets with $[n]$. Add a directed edge from $k \in K$ to $i \in I$ if $A[i,k] = 1$ and an edge from $k \in K$ to $j \in J$ if $B[k,j] = 1$. Then, computing $C[i,j]$ is the same as determining the largest $k \in K$ that is a common ancestor of both $i \in I$ and $j \in J$. Now, we will iteratively find the tth bit in the binary representation of each $C[i,j]$ for $t = 1, \ldots, \ell$ (the first bit is the highest order bit, and the last bit is the lowest order bit). In the first iteration, we do the following.

Phase 1: Construct a graph G_1 by first making a copy of G and adding a vertex w. Then, we add a directed edge from w to every vertex in $I \cup J$. Now, add a directed edge from w to all vertices $k \in K$ whose binary representation starts with 1. Finally, run AP-Ver-LCA where we guess w is an LCA for all pairs $(i,j) \in I \times J$. If w is in fact an LCA, set $c_{i,j}^{(1)} = 0$, and otherwise, set $c_{i,j}^{(1)} = 1$.

More generally, at the tth iteration of the algorithm, we do the following.

Phase t: At the tth iteration of the algorithm for $1 \le t \le \ell$, construct the graph G_t as follows. First, make a copy of G. Then, for each string $b = b_1 b_2 \ldots b_{t-1} \in \{0,1\}^{t-1}$, create a vertex w_b. Now, add an edge from w_b to all vertices in K whose binary representation starts with $b_1 b_2 \ldots b_{t-1} || 1$. Then, add an edge from every w_b to every vertex in $I \cup J$. If $c_{i,j}^{(t-1)} = b$, guess that w_b is an LCA for $(i,j) \in I \times J$. Run AP-Ver-LCA with all of these guesses. If the algorithm outputs YES for (i,j), set $c_{i,j}^{(t)} = b||0$. Otherwise, set $c_{i,j}^{(t)} = b||1$.

We show by induction that at Phase t, $c_{i,j}^{(t)}$ is the first t bits of $C[i,j]$. In Phase 1, note that w is an LCA for $(i,j) \in I \times J$ exactly when none of its children are common ancestors of (i,j). In other words, (i,j) has no common ancestor (and hence no witness) $k \in K$ whose first bit is 1.

Suppose at iteration $t-1$, this claim is true. In other words, for each i, j, $d_{i,j} = c_{i,j}^{(t-1)}$ corresponds to the first $t-1$ bits of $C[i,j]$. Then, at iteration t, we guessed that $w_{d_{i,j}}$ is an ancestor. Since $w_{d_{i,j}}$ only has children whose first t bits are $d_{i,j}||1$, it is an LCA of (i,j) exactly when none of these children are common ancestors, i.e. the largest common ancestor of (i,j) has binary representation starting with $d_{i,j}||0$. Otherwise, it starts with $d_{i,j}||1$, as desired.

Therefore, after ℓ iterations, we have that $C[i,j] = c_{i,j}^{(\ell)}$ (where we interpret $c_{i,j}$ as an ℓ-bit binary integer). The algorithm does $O(n^2)$ work at each phase to construct G_t, and then invokes an AP-Ver-LCA algorithm. Hence the overall runtime is $\tilde{O}(n^2 + T(n)) = \tilde{O}(T(n))$, as desired. ◀

▶ **Theorem 7.** *Assuming the* $(4,3)$-Hyperclique *hypothesis,* Ver-LCA *requires* $n^{2.5-o(1)}$ *time.*

Proof. Suppose we are given a 3-uniform 4-partite hypergraph G on vertex sets A, B, C, U, where $|A| = |B| = |C| = \sqrt{n}$, and $|U| = n$. The $(4,3)$-Hyperclique hypothesis implies that it requires $n^{2.5-o(1)}$ time to determine whether G contains a 4-hyperclique by Fact 18.

We construct the following Ver-LCA instance G' on $O(n)$ vertices.

The graph G' contains 3 layers of vertices V_1, V_2, V_3 with an additional vertex s. We set V_1 to be $A \times B$, set V_2 to be a copy of U and set V_3 to be $(B \times C) \sqcup (C \times A)$.

We also add the following edges to the graph G'.

- Add a directed edge from every $v_1 \in V_1$ to every $v_3 \in V_3$.
- Add a directed edge from $(a,b) \in V_1$ to $u \in V_2$ if and only if $\{u,a,b\} \in E(G)$.
- Add a directed edge from $u \in V_2$ to $(b,c) \in V_3$ if and only if $\{u,b,c\} \in E(G)$. Similarly, add a directed edge from $u \in V_2$ to $(c,a) \in V_3$ if and only if $\{u,c,a\} \in E(G)$.
- Add a directed edge from s to every other vertex in G'. This ensures that every pair of vertices has some common ancestors, and thus has at least one LCA.

We claim that for every $a \in A, b \in B, c \in C$, a, b, c are in a 4-hyperclique in G if and only if $\{a,b,c\} \in E(G)$ and (a,b) is *not* an LCA of (b,c) and (c,a) in G.

First, if a, b, c are in a 4-hyperclique with u, then clearly $\{a,b,c\} \in E(G)$. Also, by the construction of G', $((a,b),u), (u,(b,c)), (u,(c,a))$ are all edges in G'. Thus, (a,b) can reach a vertex u which can reach both (b,c) and (c,a), so (a,b) is not an LCA of (b,c) and (c,a).

Conversely, if $\{a,b,c\} \in E(G)$ and (a,b) is *not* an LCA of (b,c) and (c,a), then since (a,b) can reach both (b,c) and (c,a) via edges added from V_1 to V_3, (a,b) must be able to reach some vertex that can reach both (b,c) and (c,a). Such a vertex must belong to V_2. Say the vertex is u, then by the construction of G', we must have $\{a,b,u\}, \{b,c,u\}, \{c,a,u\} \in E(G)$. Together with the hyperedge $\{a,b,c\}$, a, b, c is in a 4-hyperclique.

Therefore, we can run Ver-LCA on G' with the following set of LCA candidates:

- For every $a \in A, b \in B, c \in C$ such that $\{a,b,c\} \in E(G)$, let $w_{(b,c),(c,a)} = (a,b)$.
- For every other pair of vertices $u, v \in V(G')$, we use Grandoni et al.'s algorithm [25] to find an actual LCA $\ell_{u,v}$ for them in $O(n^{2.447})$ time and set $w_{u,v} = \ell_{u,v}$.

If some LCA candidate is incorrect, it must be that (a,b) is not an LCA for (b,c) and (c,a) for some $a \in A, b \in B, c \in C$ such that $\{a,b,c\} \in E(G)$ and thus by previous discussion, the hypergraph G has a 4-hyperclique. On the other hand, if all LCA candidates are correct, then the hypergraph G does not have a 4-hyperclique.

Therefore, assuming the $(4,3)$-Hyperclique hypothesis, Ver-LCA requires $n^{2.5-o(1)}$ time. ◀

Our conditional lower bounds for AP-Ver-LCA and Ver-LCA are surprising because they suggest that AP-Ver-LCA and Ver-LCA require $n^{2.5-o(1)}$ time, while AP-LCA can be computed in $O(n^{2.447})$ time [25]. This defies the common intuition that verification should be easier than computation.

8 Open problems

We conclude this work by pointing out some potential future directions.

1. Does there exist a subcubic time algorithm for AP-Exact$k$-LCA for any $3 \leq k \leq 5$? Or, can we show an $n^{3-o(1)}$ conditional lower bound for AP-Exact$k$-LCA for any such k? How about AP-Ver-LCA?
2. Is it possible to show conditional lower bounds for AP-List-k-LCA without using AP-AtLeast$k$-LCA as an intermediate problem? For instance, since AP-AtLeast$k$-LCA has $O(n^\omega)$ time algorithms for $k \leq 3$, we cannot hope to get a higher than n^ω lower bound for AP-List-k-LCA for $k \leq 3$ using AP-AtLeast$k$-LCA as an intermediate problem. However, the current best algorithm for AP-List-1-LCA runs in $O(n^{2.447})$ and the best algorithm for AP-List-2-LCA and AP-List-3-LCA runs in $O(n^{2.529})$ time.
3. All our reductions reduce to instances of LCA variants in graphs with $O(1)$ layers. In such graphs, some variants could have faster algorithms. In particular, AP-LCA has an $\tilde{O}(n^\omega)$ time algorithm [18] for graphs with $O(1)$ layers, and thus we cannot hope to show a higher conditional lower bound using our techniques. In order to overcome this, we need to find reductions that show hardness for LCA variants in graphs with many layers.
4. Are there any other related problems whose verification version is easier than the computation version? Can we reduce these problems to or from AP-LCA?

References

1 Amir Abboud, Loukas Georgiadis, Giuseppe F Italiano, Robert Krauthgamer, Nikos Parotsidis, Ohad Trabelsi, Przemysław Uznański, and Daniel Wolleb-Graf. Faster algorithms for all-pairs bounded min-cuts. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 7:1–7:15. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2019.

2 Hassan Aït-Kaci, Robert S. Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Trans. Program. Lang. Syst.*, 11(1):115–146, 1989.

3 Josh Alman. Limits on the universal method for matrix multiplication. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, volume 137, pages 12:1–12:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

4 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

5 Matthias Baumgart, Stefan Eckhardt, Jan Griebsch, Sven Kosub, and Johannes Nowak. All-pairs ancestor problems in weighted dags. In *Proceedings of the First International Conference on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, ESCAPE'07, pages 282–293. Springer-Verlag, 2007.

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

7 Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993.

8 Omer Berkman and Uzi Vishkin. Finding level-ancestors in trees. *J. Comput. Syst. Sci.*, 48(2):214–230, 1994.

9 Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *J. Comput. Syst. Sci.*, 47(3):549–595, 1993.

10 Vincent Bouchitté and Jean-Xavier Rampon. On-line algorithms for orders. *Theor. Comput. Sci.*, 175(2):225–238, 1997.

11 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Proceedings of the 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE, 2017.

12 Peter Bürgisser, Michael Clausen, and Mohammad Amin Shokrollahi. *Algebraic Complexity Theory*. Springer Verlag, 1997.

13 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the Exact Complexity of Evaluating Quantified k-CNF. In *Proceedings of the 5th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 6478, pages 50–59. Springer, 2010.

14 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. On the Exact Complexity of Evaluating Quantified k-CNF. *Algorithmica*, 65(4):817–827, 2013.

15 Matthias Christandl, François Le Gall, Vladimir Lysikov, and Jeroen Zuiddam. Barriers for rectangular matrix multiplication. *CoRR*, abs/2003.03019, 2020. arXiv:2003.03019.

16 Matthias Christandl, Péter Vrana, and Jeroen Zuiddam. Barriers for fast matrix multiplication from irreversibility. In *Proceedings of the 34th Computational Complexity Conference (CCC)*, volume 137, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

17 Richard Cole and Ramesh Hariharan. Dynamic LCA queries on trees. *SIAM J. Comput.*, 34(4):894–923, 2005.

18 Artur Czumaj, Miroslaw Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theor. Comput. Sci.*, 380(1-2):37–46, 2007.

19 Roland Ducournau and Michel Habib. On some algorithms for multiple inheritance in object-oriented programming. In *Proceedings of ECOOP' 87 European Conference on Object-Oriented Programming*, volume 276, pages 243–252. Springer, 1987.

20 Stefan Eckhardt, Andreas Michael Mühling, and Johannes Nowak. Fast lowest common ancestor computations in dags. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, volume 4698, pages 705–716, 2007.

21 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comput. Sci.*, 326(1-3):57–67, 2004.

22 Johannes Fischer and Daniel H. Huson. New common ancestor problems in trees and directed acyclic graphs. *Inf. Process. Lett.*, 110(8-9):331–335, 2010.

23 Michael J. Fischer and Albert R. Meyer. Boolean matrix multiplication and transitive closure. In *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (SWAT)*, pages 129–131. IEEE, 1971.

24 Harold N. Gabow, Jon Louis Bentley, and Robert Endre Tarjan. Scaling and related techniques for geometry problems. In *Proceedings of the 16th Annual ACM Symposium on Theory of Computing (STOC)*, pages 135–143. ACM, 1984.

25 Fabrizio Grandoni, Giuseppe F. Italiano, Aleksander Lukasiewicz, Nikos Parotsidis, and Przemyslaw Uznanski. All-pairs LCA in dags: Breaking through the $O(n^{2.5})$ barrier. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 273–289. SIAM, 2021.

26 Michel Habib, Marianne Huchard, and Jeremy P. Spinrad. A linear algorithm to decompose inheritance graphs into modules. *Algorithmica*, 13(6):573–591, 1995.

27 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984.

28 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

29 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

30 Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580, pages 241–248, 2005.

31 Miroslaw Kowaluk and Andrzej Lingas. Unique lowest common ancestors in dags are almost as easy as matrix multiplication. In *Proceedings of the 15th Annual European Symposium (ESA)*, volume 4698, pages 265–274. Springer, 2007.

32 Mirosław Kowaluk, Andrzej Lingas, and Johannes Nowak. A path cover technique for lcas in dags. In *Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 222–233. Springer, 2008.

33 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046. SIAM, 2018.

34 Andrea Lincoln, Adam Polak, and Virginia Vassilevska Williams. Monochromatic Triangles, Intermediate Matrix Products, and Convolutions. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 151, pages 53:1–53:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

35 Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1236–1252. SIAM, 2018.

36 Matti Nykänen and Esko Ukkonen. Finding lowest common ancestors in arbitrarily directed trees. *Inf. Process. Lett.*, 50(6):307–310, 1994.

37 Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.

38 Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, 1979.

39 Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.

40 Zhaofang Wen. New algorithms for the LCA problem and the binary tree reconstruction problem. *Inf. Process. Lett.*, 51(1):11–16, 1994.

41 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.