Subtree Isomorphism Revisited *

Amir Abboud
Stanford University
abboud@cs.stanford.edu

Arturs Backurs
MIT
backurs@mit.edu

Thomas Dueholm Hansen
Aarhus University
tdh@cs.au.dk

Virginia Vassilevska Williams
Stanford University
virgi@cs.stanford.edu

Or Zamir
Tel Aviv University
orzamir@mail.tau.ac.il

Abstract

The *Subtree Isomorphism* problem asks whether a given tree is contained in another given tree. The problem is of fundamental importance and has been studied since the 1960s. For some variants, e.g., *ordered trees*, near-linear time algorithms are known, but for the general case truly subquadratic algorithms remain elusive.

Our first result is a reduction from the Orthogonal Vectors problem to Subtree Isomorphism, showing that a truly subquadratic algorithm for the latter refutes the Strong Exponential Time Hypothesis (SETH).

In light of this conditional lower bound, we focus on natural special cases for which no truly subquadratic algorithms are known. We classify these cases against the quadratic barrier, showing in particular that:

- Even for binary, rooted trees, a truly subquadratic algorithm refutes SETH.
- Even for rooted trees of depth $O(\log \log n)$, where n is the total number of vertices, a truly subquadratic algorithm refutes SETH.
- For every constant d, there is a constant $\varepsilon_d > 0$ and a randomized, truly subquadratic algorithm for degree-d rooted trees of depth at most $(1 + \varepsilon_d) \log_d n$. In particular, there is an $O(\min\{2.85^h, n^2\})$ algorithm for binary trees of depth h.

Our reductions utilize new "tree gadgets" that are likely useful for future SETH-based lower bounds for problems on trees. Our upper bounds apply a folklore result from randomized decision tree complexity.

1 Introduction

Trees are among the most frequently used and commonly studied objects in computer science. One of the most basic and fundamental computational problems on trees is whether one tree is contained in another, that is, can an isomorphic copy of H be obtained by deleting nodes and edges of G. This problem is known under three names: *Subtree Isomorphism*, *Tree Pattern Matching* and *Subgraph Isomorphism on Trees*. There are a few variants of the problem, mainly determined by (1) whether the trees are rooted or unrooted, (2) whether their degrees are bounded, and (3) whether the trees are ordered, i.e. whether the order of the children of each node must be preserved by the isomorphism. In this paper we focus on the case of rooted, unordered trees with degrees bounded by a constant d.

Because of its fundamental importance, the time complexity of Subtree Isomorphism has been studied since the 1960s, e.g. by Matula [44] and Edmonds (see [45]). The problem is an interesting special case of the Subgraph Isomorphism problem, studied extensively in theoretical computer science. Subgraph Isomorphism is well known to be NP-hard since it generalizes hard problems such as Clique [34]. It is notoriously difficult: unlike most natural NP-complete problems, it *requires* $2^{\omega(n)}$ time (under the exponential time hypothesis (ETH)) [18]. Special cases of subgraph isomorphism, especially ones that are in P, have received extensive attention. A recent 85-page paper by Marx and Pilipczuk [42] covers the case in which H is of fixed constant size. Besides fixing the size of H, there are other non-trivial ways to make the problem polynomial time solvable; Subtree Isomorphism is the earliest and arguably the most natural one. Polynomial time algorithms were also obtained for biconnected outerplanar graphs [39], two-connected series-parallel graphs [41], and more [43, 19], while it is known that further generalizations quickly become NP-hard, e.g., when G is a forest and H is a binary tree [24].

The problem is also of practical relevance, since it can model important applications in a wide variety of areas. Subtree Isomorphism is at the core of many more expressive problems, such as *Largest Common Subtree* [35, 6, 7], which generally ask: how "similar" are two

*A.A. and V.V.W. were supported by NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338. A.B. was supported by the NSF and the Simons Foundation; part of the work was done while the author was at the Thomas J. Watson Research Center. T.D.H. was supported by the Carlsberg Foundation, grant no. CF14-0617. O.Z. was supported by BSF grant no. 2012338 and by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

trees? Application areas include computational biology [58], structured text databases [36], and compiler optimization [52]. Several definitions of tree-similarity have been proposed, and the search for fast algorithms for computing them, both in theory and in practice, has been ongoing for a few decades - see [10, 23, 27, 53] for surveys and textbooks. We focus on Subtree Isomorphism, and then briefly discuss how the techniques introduced in this paper can be adapted to prove new results for the Largest Common Subtree problem as well.

Previous results. According to Matula [45], the first algorithms for Subtree Isomorphism were proposed in 1968 independently by Edmonds and Matula himself [44]. 10 years later, Reyner [48] and Matula [45] showed that these algorithms run in polynomial time and the runtime is $O(n^{2.5})$. The algorithm executes many calls to a subroutine that solves maximum matching in bipartite graphs. These result were for rooted trees, and later Chung [14] showed that the same bounds can be achieved for unrooted trees. In 1983, Lingas [38] shaved a log factor, and the most recent development was in 1999 by Shamir and Tsur [51] who used the more recent randomized algorithms for bipartite matching [13] to reduce the runtime to $O(n^\omega)$ where $\omega < 2.373$ is the matrix multiplication exponent [56, 22].

Interestingly, in the most basic case of rooted and constant degree trees, even the early algorithms run in $O(n^2)$ time, and the fastest known runtime is $O(n^2/\log n)$ [38, 51]. For comparison, when the trees are *ordered*, a long line of STOC/FOCS papers [37, 21, 15, 32, 33, 16] brought down the complexity of the problem from quadratic [28] to $O(n \log n)$ time [17]. It is natural to wonder whether the same improvements can be achieved in the case of unordered trees.

Main results. Our main result is a conditional lower bound for Subtree Isomorphism. We show that a truly subquadratic algorithm is unlikely, even on very restricted cases such as those of binary, rooted trees or rooted trees of depth $O(\log \log n)$. A matching upper bound, up to $n^{o(1)}$ factors, has been known since the 1960s (we briefly discuss this algorithm in Section 3).

Our lower bounds are conditioned on the well-known Strong Exponential Time Hypothesis (SETH) of Impagliazzo, Paturi and Zane [30, 31] which roughly states that as k grows, k-SAT on n variables requires $2^{(1-\varepsilon)n}\,\mathrm{poly}(n)$ time for all $\varepsilon > 0$. Our result for Subtree Isomorphism is the first "SETH-hard" problem on *trees*, which is an exciting addition to the diverse list[1] which already includes problems on vectors [55], (general) graphs [47, 49, 3, 2], sequences [5, 9, 1, 12], and curves [11]. Our ideas and constructions of "tree gadgets" are useful for proving conditional lower bounds for other problems on trees. We demonstrate this with a lower bound for the Largest Common Subtree problem, discussed below.

THEOREM 1.1. *For all $d \geq 2$, Subtree Isomorphism on two rooted, unordered trees of size $O(n)$, degree d, and height $h \leq 2\log_d n + O(\log \log n)$ cannot be solved in truly subquadratic $O(n^{2-\varepsilon})$ time under SETH.*

More generally, if the size of the smaller tree is n and the bigger tree is m, then our lower bound says that $O(nm^{1-\varepsilon})$ time refutes SETH. We remark that since SETH is believed to hold even for randomized algorithms, our lower bound is also a barrier for truly subquadratic randomized algorithms.

To complement our lower bound, we proceed to tackle natural restrictions of the problem algorithmically. The most natural way to restrict tree inputs is to bound the degree or height. Our lower bound leaves little room for improvement: Even on binary trees of height $(2+o(1))\log n$ any algorithm must take quadratic time under SETH (note that the minimum height of a binary tree is $\log n$).

An intriguing case is when the trees are binary and almost complete, i.e., $d = 2$ and $h = (1+o(1))\log n$. We are unable to show a super-linear lower bound in this case, nor are we able to obtain a deterministic algorithm that runs in truly subquadratic time. Nevertheless, we present a *randomized*, Las Vegas, algorithm that solves this case in truly subquadratic $O(n^{1.507})$ time. Our algorithm solves more general cases:

THEOREM 1.2. *There is a randomized algorithm for rooted Subtree Isomorphism with expected running time $O(\min\{2.8431^h, n^2\})$ for trees H and G of size $O(n)$ and height at most h. In particular, the algorithm runs in time $O(n^{1.507})$ for trees of depth $(1+o(1))\cdot\log_2 n$ and is truly subquadratic for trees of depth $h \leq 1.3267 \cdot \log_2 n$.*

Our algorithm is simple, natural, and easy to implement. Perhaps more interesting than the upper bound itself is that the technique we use to obtain it uses a technique from randomized decision tree complexity.

We also consider the case of ternary trees, providing a fast Las Vegas algorithm for it. Our approach is similar to that of the binary tree case. However, here we use a computer program to analyze the expected running time of the algorithm.

THEOREM 1.3. *There is a randomized algorithm that can solve Subtree Isomorphism on two rooted* ternary *trees of size $O(n)$ and height at most h in expected $O\left(\min\left\{6.107^h, n^2\right\}\right)$ time.*

[1] These are problems with $O(n^c)$ upper bounds for some $c > 1$ and an $O(n^{c-\varepsilon})$ algorithm, for some $\varepsilon > 0$, is known to refute SETH.

Finally, we generalize our algorithms to obtain truly subquadratic algorithms for rooted Subtree Isomorphism on trees with small height and constant degree d, for any $d \geq 2$.

THEOREM 1.4. *There is a randomized algorithm that solves Subtree Isomorphism on two rooted trees of size $O(n)$, constant degree d, and height at most h in expected time*

$$O\left(\min\left\{\left(d^2 - \frac{1}{3}d + \frac{2}{3}\right)^h, n^2\right\}\right).$$

In particular, the algorithm is strongly subquadratic for trees of height

$$h \;\leq\; \left(\frac{\log(d^2)}{\log(d^2 - \frac{1}{3}d + \frac{2}{3})} - \epsilon\right) \cdot \log_d n \;,$$

for any constant $\epsilon > 0$.

The bound in the above theorem is not tight for small d, as our algorithms for $d = 2$ and $d = 3$ show. For example, it is not subquadratic (on small depth trees) unless $d > 3$. To obtain the upper bound, we prove a new randomized query complexity upper bound for bipartite perfect matching, which could be of independent interest (Lemma 3.2).

This work is another example of a fine-grained study of the complexity of fundamental problems in P under natural parameterizations. This approach was formalized in two recent works [4, 25].

Techniques and other results. To prove our SETH hardness results we show reductions from Orthogonal Vectors to Subtree Isomorphism in Section 2. The reductions follow all previous SETH-hardness results in spirit, but require careful constructions of "tree gadgets" that represent vectors, as well as techniques for combining the gadgets into two big trees H and G for which the existence of an orthogonal pair of vectors determines whether H is contained in G. Our reduction is clean and simple, but it gets more tricky when restricted to trees of constant degree.

Our reduction is easily modified to obtain similar lower bounds for related problems such as *Largest Common Subtree* on two trees (LCST). This problem is NP-hard when the number of trees is a parameter or when the two trees are labelled (and unrooted) [59, 57], while some approximation and parameterized algorithms are known [35, 7, 6]. When the two trees are binary and unlabeled, the problem can be solved in quadratic time, and an adaptation of Theorem 1.1 shows that even when the height is $(1 + o(1)) \log n$, a truly subquadratic algorithm refutes SETH.

THEOREM 1.5. *For all $d \geq 2$, The Largest Common Subtree problem on two rooted trees of size $O(n)$, degree d and height $h \leq \log_d n + O(\log \log n)$ cannot be solved in truly subquadratic $O(n^{2-\varepsilon})$ time under SETH.*

Theorem 1.5 is surprising when contrasted with our other results. On the one hand, for arbitrary rooted trees with constant degrees, both Subtree Isomorphism and the harder-looking LCST have tight quadratic upper and (conditional) lower bounds. On the other hand, we show that under the further restriction that the trees have small depth (as in Theorem 1.2), Subtree Isomorphism can be solved in truly subquadratic time, while by Theorem 1.5 the LCST problem *cannot*, under SETH.

We attribute our new algorithmic results to two ingredients. The first important ingredient comes from our *lower bounds*. In particular we noticed that when the trees are binary and the depth is $(1 + \varepsilon) \log n$, it is difficult to implement our reductions. This turned our attention to finding upper bounds. Knowing the hard cases thus allowed us to focus on the solvable cases. This is an important byproduct of the recent research on conditional lower bounds in P.

The second ingredient was making a connection between this problem and a seminal result from randomized decision tree complexity [50]. Our algorithm for binary (and ternary) trees is inspired by the following well-known result from complexity theory: Given a formula represented by a complete AND-OR tree on n leaves that represent the variables, can you evaluate the formula without looking at all the inputs? The surprising fact is that this is possible with randomization: to evaluate a gate, we guess which child to check first at random, and if we see a 1 input to an OR gate, or a 0 input to an AND gate, we do not have to check the other child. Therefore it is possible to evaluate the formula by only looking at $n^{1-\varepsilon}$ inputs. This result has found many applications in various areas of complexity theory, learning theory, and quantum query complexity [8].

Other related work. In the late 1980s, Subtree Isomorphism was considered from the viewpoint of efficient parallel algorithms. Lingas and Karpinski [40] placed the problem in randomized NC^1. Gibbons, Miller, Karp, and Soroker [26] independently obtained the same result and also showed an NC^1 reduction from bipartite matching to Subtree Isomorphism. Their reduction takes a matching instance on n nodes and produces trees on $\Omega(n^3)$ nodes, and therefore does not imply a lower bound on the time complexity of Subtree Isomorphism even assuming that current matching algorithms are optimal. Note that any many-to-one reduction from matching (where the input is of size $\Omega(n^2)$)

will generate trees of size $\Omega(n^2)$. To get our quadratic lower bound we reduce from a different problem, namely Orthogonal Vectors.

Many related cases of the problem can be solved in near-linear time. For example, when both trees have exactly the same size, we get the *Tree Isomorphism* problem which was solved in $O(n)$ time by Hopcroft and Tarjan [29], and later other linear time algorithms were suggested (see [20] and the references therein). Another example is the case of *ordered trees*, meaning that there is an order among the children of a node that cannot be modified in the isomorphism. Also, when a "subtree" is defined to be a node and all its descendants, "subtree" isomorphism can be solved in linear time [54].

2 SETH Lower Bounds

In this section we reduce CNF-SAT, via the Orthogonal Vectors (OV) problem, to different variants of the Subtree Isomorphism problem to prove our SETH-based lower bounds. The inputs to OV are two lists of N vectors in $\{0,1\}^D$ and the output is "yes" if and only if there is a pair of vectors α, β, one from each list, that are orthogonal, i.e. for all $i \in [D]$ either $\alpha[i]$ or $\beta[i]$ is equal to 0. Williams [55] showed that if OV can be solved in $O(N^{2-\varepsilon})$ time when $D = \omega(\log N)$, for some $\varepsilon > 0$, then CNF-SAT on n variables and $n^{O(1)}$ clauses can be solved in $2^{(1-\varepsilon/2)n}$ time, and SETH is false.

2.1 Hardness for Subtree Isomorphism

A simpler reduction. We start with a "warm-up" reduction that presents the high-level idea of our proofs. In Theorem 2.1 below we reduce OV to Subtree isomorphism on trees with $n = O(ND)$ vertices, unbounded degree, and height $h = O(D)$. We later show how to change the construction to get trees with small constant degree and small height.

THEOREM 2.1. *Orthogonal Vectors on two lists of N vectors in $\{0,1\}^D$ can be reduced to Subtree Isomorphism on two trees of size $O(ND)$ and depth $O(d)$.*

Proof. Let us denote the vectors of the first list by $A = \{\alpha_1, \ldots, \alpha_N\}$ and of the second list by $B = \{\beta_1, \ldots, \beta_N\}$ and recall that our goal is to find a pair of vectors $\alpha \in A, \beta \in B$ such that for every coordinate $i \in [D]$ either $\alpha[i] = 0$ or $\beta[i] = 0$.

The first ingredient in the reduction is to construct *vector gadgets*.

For every vector in the first list $\alpha \in A$ we create a vector gadget: a tree H_α of size $O(D)$ as follows. First, add a path $u_0 \to u_1 \to u_2 \to \cdots \to u_{D+2}$ and let u_0 be the root of H_α. Then, for each coordinate $i \in [D]$ we consider $\alpha[i]$ and if it is a 1 we add a node $u_{i,1}$ to the tree H_α as the child of the node u_i, i.e. we add the edge $u_i \to u_{i,1}$. Otherwise, if $\alpha[i] = 0$, the only child of u_i will be u_{i+1}.

We now define the vector gadgets for the vectors in the second list. For every $\beta \in B$ we create a vector gadget: a tree G_β of size $O(D)$ as follows. The first step is similar, we add a path $v_0 \to v_1 \to v_2 \to \cdots \to v_{D+2}$ and let v_0 be the root. The difference is in the second step. For each coordinate $i \in [D]$, we consider $\beta[i]$ and if it is a 0 we add a node $v_{i,0}$ to G_β as the child of the node v_i, i.e. we add the edge $v_i \to v_{i,0}$.

The following simple claim is the key to our reduction and explains our gadget constructions.

CLAIM 2.1. *H_α is isomorphic to G_β iff α, β are orthogonal.*

Proof. For the first direction, assume that α, β are orthogonal and therefore for every $i \in [D]$ we know that either $\alpha[i] = 0$ or $\beta[i] = 0$. We will define a mapping f from H_α to a subgraph of G_β such that if $\{u, v\}$ is an edge in H_α then $\{f(u), f(v)\}$ is an edge in G_β. First, we map the roots and paths to each other, by setting $f(u_i) = v_i$ for all $i \in \{0, \ldots, D+2\}$. Then, we consider every $i \in [D]$ for which $\alpha[i] = 1$ and map $u_{i,1}$ to the node $v_{i,0}$ in G_β. We are guaranteed that $v_{i,0}$ exists because if $\alpha[i] = 1$ then $\beta[i]$ must be 0, by the orthogonality of the vectors. It is easy to check that two neighbours in H_α are mapped to two neighbours in G_β.

For the other direction, assume H_α is isomorphic to a subgraph of G_β, and let f be the mapping. First, note that u_0 must be mapped to v_0 since these are the roots of the two trees. Then we observe that u_{D+2} must be mapped to v_{D+2} and the path $u_0 \to \cdots \to u_{D+2}$ must be mapped to the path $v_0 \to \cdots \to v_{D+2}$ since these are the only paths of length at least $(D+2)$ in the trees. Now, let $i \in [D]$ be such that $\alpha[i] = 1$ and note that u_i must have degree 3 in this case, which implies that $f(u_i) = v_i$ must also have degree at least 3 in G_β, which implies that the node $v_{i,0}$ must exist, and $\beta[i] = 0$. Thus, whenever $\alpha[i] = 1$ it must be the case that $\beta[i] = 0$, and the vectors are orthogonal. □

The final step is to combine the vector gadgets into two trees H, G in a way such that H is isomorphic to a subtree of G if and only if there is a pair of orthogonal vectors within our two lists.

To this end, we define a special vector $\gamma = \vec{0}$ to be the all-zero vector in D dimensions. By Claim 2.1, for any vector $\beta \in \{0,1\}^D$, we have that H_β is isomorphic to a subtree of G_γ.

We are now ready to define the trees H and G of size $O(ND)$.

G will be composed of a root node g of degree $(2N-1)$ that has G_{β_j} as a child for every $\beta_j \in B$,

in addition to $(N-1)$ distinct G_γ gadgets. That is, first, for each $j \in [N]$ add the vector gadget G_{β_j} to G and add the edge $g \to v_0$ where v_0 is the root of G_{β_j}. And then, we add $(N-1)$ trees $G_\gamma^{(1)}, \ldots, G_\gamma^{(n-1)}$ to G and for each $j \in [N-1]$ we add the edge $g \to v_0$ where v_0 is the root of $G_\gamma^{(j)}$.

H will be constructed in a similar way, except we do not add the γ vector gadgets. Create a root node h of degree N that has H_{α_j} as a child for every $\alpha_j \in A$. As in the definition of G, we add edges $h \to u_0$ where u_0 is the root of H_{α_j}, for every $j \in [N]$.

Before proving the correctness of the reduction, note that the size of each tree is indeed $O(ND)$ since each gadget has size $O(D)$ and we are combining $O(N)$ gadgets into our trees H, G. To conclude the proof, we claim that H is isomorphic to a subgraph of G iff there is a pair of orthogonal vectors.

CLAIM 2.2. *In the above reduction, H is isomorphic to a subtree of G iff there is a pair $\alpha \in A, \beta \in B$ of orthogonal vectors.*

Proof. For the first direction, assume that there is a pair of orthogonal vectors $\alpha \in A, \beta \in B$ and we will show that H is isomorphic to a subtree of G. Consider the mapping which maps H_α to G_β as in Claim 2.1, and then for each of the $(N-1)$ $H_{\alpha'}$ subtrees, for $\alpha' \neq \alpha$, we map it to a different G_γ subtree of G. Finally, the root h is mapped to g. It is easy to check that neighbours in H are mapped to neighbours in G.

For the other direction, assume that H is isomorphic to a subgraph of G and let f be the corresponding mapping. We know that $f(h) = g$ and for each vector gadget H_{α_j} in H, its image using our mapping f must be entirely contained in exactly one vector gadget G_x in G, where $x \in B \cup \{\gamma\}$. Moreover, two gadgets $H_\alpha, H_{\alpha'}$ cannot be mapped to the same gadget G_x. There are N H_α gadgets but only $(N-1)$ G_γ gadgets, thus, by the pigeonhole principle, there must be at least one $\alpha \in A$ for which H_α is mapped to a gadget G_x for $x \neq \gamma$, i.e., $x = \beta$ for some $\beta \in B$. We conclude that there is a mapping from H_α to G_β in which every two neighbours are mapped to neighbours, that is, that H_α is isomorphic to a subgraph of G_β, which, by Claim 2.1, implies that $\alpha \in A, \beta \in B$ are orthogonal. □

□

Shorter Vector Gadgets. Next, we show how our reductions can be implemented with trees of smaller depth, by introducing a new construction of vector gadgets. We will use these gadgets in our final reductions that prove Theorems 1.1 and 1.5.

LEMMA 2.1. *Given two vectors $\alpha, \beta \in \{0,1\}^D$ we can construct two binary rooted trees H_α, G_β of depth $3\log_2(D)+O(1)$ in linear time, such that H_α is isomorphic to a subtree of G_β if and only if α, β are orthogonal.*

Proof. Our constructions will involve careful combinations of "index gadgets", which are defined as follows. For a sequence of ℓ binary values $b_1, b_2, \ldots, b_l$, we define a tree "index gadget" $Q_{b_1,b_2,\ldots,b_l}$ (think of ℓ as being $\lceil \log_2(D+1) \rceil$ and think of $b_1, b_2, \ldots, b_l$ as bits representing an index in $[D]$) to be composed of a path $z_1 \to z_2 \to \ldots \to z_l$ of length l, in which z_1 is the root, and for all $i \in [l]$ we attach a child $z_{i,1}$ to z_i if and only if $b_i = 1$. That is, our index gadget $Q_{b_1,b_2,\ldots,b_l}$ is representing the index in the natural way: the edge $z_i \to z_{i,1}$ will exist if and only if $b_i = 1$.

Our first "vector gadget" H_α is constructed as follows. First, we build a complete binary tree with D leaves $u_1, u_2, \ldots, u_D$ where the subtree at each leaf u_i will encode the entry $\alpha[i]$ using our "index gadgets". We assume that every index $i \in [D]$ can be represented by $l = \lceil \log_2(D+1) \rceil$ bits and we let $\bar{i}$ denote this representation and let $\bar{i}^S$ denote the binary sequence obtained by flipping each bit of $\bar{i}$. For each node u_i we will attach three gadgets, one after the other: first we will attach the $Q_{\bar{i}}$ index gadget, then we follow it by the $Q_{\bar{i}^S}$ index gadget, and finally we append a path of length either 2 or 3 – depending on $\alpha[i]$. The necessity of this complicated encoding will become clear in the proof of correctness below. More formally, we first attach $u_i \to Q_{\bar{i}}$, then we let z'_l denote the node of $Q_{\bar{i}}$ corresponding to z_l in the above construction (i.e. the last node on the path), and attach $z'_l \to Q_{\bar{i}^S}$. Then, similarly, we let z''_l be the node of $Q_{\bar{i}^S}$ which corresponds to z_l in the above construction (i.e. the last node on the path), and we either attach three nodes $z''_l \to a_i \to b_i \to c_i$ if $\alpha[i] = 1$, or we attach only two nodes $z''_l \to a_i \to b_i$.

The second "vector gadget" G_β is constructed in the same way except that we attach a path of length 3 if $\beta[i] = 0$ (as opposed to 1) and attach a path of length 2 if $\beta[i] = 1$. By construction, the depth of both trees is $3\log_2(D) + O(1)$ as claimed.

To complete the proof we show that H_α is isomorphic to a subtree of G_β iff $\alpha \cdot \beta = 0$. The first direction is easy: if the vectors are orthogonal then the natural mapping from H_α to G_β that follows from our construction shows the isomorphism: map the binary trees on top to each other so that the u_i's are mapped to each other, then map the attached $Q_{\bar{i}} \to Q_{\bar{i}^S}$ subtrees to each other, and finally, we can map the paths $a_i \to b_i \to c_i$ (if $\alpha[i] = 1$) or $a_i \to b_i$ (if it is 0) to each other since in the first case $\beta[i]$ must be zero and c_i will also exist in G_β.

It remains to show that if H_α is isomorphic to a subtree of G_β, then $\alpha \cdot \beta = 0$. Our index gadgets $Q_{\bar{i}}$ and $Q_{\bar{i}^S}$ will play a crucial role in this part, as they will show that in any mapping between the leaves of the complete tree we must map u_i in H_α to u_i in G_β or else the index gadgets will not map into each other properly. We claim that for any two indices $i, j \in [D]$ we have that $i = j$ if and only if both $Q_{\bar{i}}$ is contained in $Q_{\bar{j}}$ *and* $Q_{\bar{i}^S}$ is contained in $Q_{\bar{j}^S}$. This is true because of the following observation: $Q_{\bar{x}}$ is isomorphic to a subtree of $Q_{\bar{y}}$ iff the set of positions in $\bar{x}$ with 1 is a subset of the set of positions of $\bar{y}$ with 1. Therefore, any mapping from H_α to a subtree of G_β must map the path representing $\alpha[i]$ to the path representing $\beta[i]$, for all $i \in [D]$. By construction, this can only happen if $\alpha \cdot \beta = 0$. □

Constant Degree Trees. Perhaps the most challenging element towards the proof of Theorem 1.1 is the combination of all the vector gadgets into two big trees, *without using large degrees*.

To see the difficulty, recall the reduction in the proof of Theorem 2.1: in both trees, we added all X vector gadgets as children of a root of degree X. By doing so we have essentially allowed the isomorphism to pick *any* matching between the gadgets. Combined with the auxiliary gadgets that we added, this allowed us to show that the final two trees are a "yes" instance of Subtree Isomorphism if and only if the original vectors contained an orthogonal pair. However, when the trees have constant degree (say, binary) it is much harder to combine the vector gadgets into two trees such that any matching between the gadgets can be chosen by the isomorphism. A natural approach would be to add the gadgets at the leaves of a complete binary tree. One reason this does not work is that any isomorphism must map the first and second gadgets to adjacent gadgets in the second tree – that is, only special kinds of matchings can be "implemented".

We overcome this difficulty with a two-level construction that allows the isomorphism to pick exactly one gadget from each of the two trees and "match" them, while all the other gadgets do not affect the outcome.

THEOREM 2.2. *Given sets of vectors A, B, we can construct two rooted trees $H = H(A)$ and $G = G(B)$ such that the following properties hold.*

1. *The number of nodes in both trees and the construction time is upper bounded by $O(ND)$.*
2. *The degree of both trees is upper bounded by d.*
3. *The depth of both trees is upper bounded by $2\log_d(N) + O(\log D)$.*
4. *H is isomorphic to a subtree of G iff there are $\alpha \in A$ and $\beta \in B$ with $\alpha \cdot \beta = 0$.*

Proof. Let $\{H_\alpha\}_{\alpha \in A} = \{H_{\alpha_i}\}_{i \in [N]}$ and $\{G_\beta\}_{\beta \in B} = \{G_{\beta_i}\}_{i \in [N]}$ be the two sets of vector gadgets corresponding to the vectors of A and B that are obtained by the construction in Lemma 2.1. We will now combine these vector gadgets into two big trees H and G, which will be constructed quite differently from each other.

Assume that $\log_d(N)$ is an integer, otherwise add dummy vectors to increase N. The first tree H will be composed of a complete d-ary tree with N leaves $u_1, u_2, \ldots u_N$, followed by a path of length $\log_d(N) + 1$, followed by the vector gadgets H_{α_i}. More formally, for every $i \in [N]$ we add:

$$u_i \to h_{i,1} \to h_{i,2} \to \ldots \to h_{i,\log_d(N)+1} \to H_{\alpha_i}.$$

To construct the second tree G we need to construct vector gadgets G_γ corresponding to the all-zero vector $\gamma = \vec{0}$ of length D. As before, we start with a complete d-ary tree with N leaves $v_1, v_2, \ldots v_N$ and attach a path of length $\log_d(N) + 1$ to each leaf, except for v_N which will be treated differently. Then, we attach a copy of G_γ at the end of each one of these paths, that is $N - 1$ copies in total. Formally, for every $i = 1, \ldots, N-1$ we add:

$$v_i \to h_{i,1} \to h_{i,2} \to \ldots \to h_{i,\log_d(N)+1} \to G_\gamma.$$

Note that none of the vectors in the second list are encoded in this part of G and they will appear now in the subtree rooted at v_N which we construct next. Rooted at v_N, we add another complete d-ary tree with N leaves $v'_1, v'_2, \ldots v'_N$, and then attach the vector gadgets right after these leaves. That is, for every $i \in [N]$ we add: $v'_i \to G_{\beta_i}$.

This finishes the construction of H and G and the first two properties are immediate. The third property follows from Lemma 2.1, and we now turn to proving the fourth property which is the correctness of our construction.

CLAIM 2.3. *There is a pair of vectors $\alpha \in A$ and $\beta \in B$ with $\alpha \cdot \beta = 0$ if and only if H is isomorphic to a subtree of G.*

Proof. For the first direction, let α_i and β_j be a pair of orthogonal vectors and we will show that H is contained in G. First, consider the rearrangement of H so that the rightmost leaf of the complete d-ary tree (where u_N used to be) is u_i, the node to which the vector gadget H_{α_i} is attached. We claim that all vector gadgets in H can now be properly mapped to subtrees of G, without rearranging the v_i nodes in G. To see this, first note

that all vector gadgets H_{α_x} for $x \neq i$ will be paired up with the G_γ vector gadgets, and by Lemma 2.1 and the fact that γ is orthogonal to any vector, we know that there is a proper mapping. Then, it remains to show that the subtree of H rooted at u_i is contained in the subtree of G rooted at v_N, which follows because we can map the vector gadget H_{α_i} to the vector gadget G_{β_j} since $\alpha_i \cdot \beta_j = 0$.

For the second direction, assume that there is a mapping from H to a subtree of G and we will show that there must exist a pair of orthogonal vectors. First, note that under this mapping, there is some $i \in [N]$ such that u_i is mapped to v_N. By construction of the subtree rooted at v_N, this means that the vector gadget H_{α_i} must be mapped into one of the vector gadgets G_{β_j} for some $j \in [N]$, and not into G_γ. By Lemma 2.1, this can only happen if $\alpha_i \cdot \beta_j = 0$. □

□

Theorem 2.2 and the connection between SETH and OV of Williams [55] imply Theorem 1.1 from the introduction.

2.2 Hardness for Largest Common Subtree

Next, we prove a lower bound for the Largest Common Subtree (LCST) problem, which is a generalization of Subtree Isomorphism. Although the reductions above already imply a quadratic lower bound for LCST, we will now optimize these reductions and prove a stronger hardness result: we will show that even on binary trees of depth $(1 + o(1)) \log n$ the LCST cannot be computed in truly subquadratic time. This will show an interesting gap between LCST and Subtree Isomorphism, since the latter can be solved in truly subquadratic time on such trees - we present such upper bounds in Section 3. Our strengthened hardness result gives an explanation for why we are not able to extend our upper bounds to LCST: such extensions would refute SETH. The next theorem implies Theorem 1.5 from the introduction.

THEOREM 2.3. *If for some $\varepsilon > 0$, the Largest Common Subtree problem on two trees size n can be solved in $O(n^{2-\varepsilon})$ time, then Orthogonal Vectors on N vectors in $\{0,1\}^D$ can be solved in $O(N^{2-\varepsilon} \cdot D^{O(1)})$ time. The trees produced in the reduction from the Orthogonal Vectors problem have degree d and height at most $\log_d(N) + O(\log D)$ for arbitrary $d \geq 2$.*

Proof. We note that the construction provided in Theorem 2.2 is not sufficient for our purposes because the height of the produced trees is $2\log_d(N) + O(\log D)$, which is larger than what we want. We will use the more expressive nature of LCST to implement our reduction with smaller height.

To achieve smaller height, we will try to implement vector gadgets such that the largest common subtree of two gadgets would be of a certain fixed size E if the vectors are not orthogonal, while it will be of a larger size $E' > E$ if the vectors are orthogonal. This trick was introduced by Backurs and Indyk in their reduction to Edit-Distance [9] and later used in the reductions to LCS [1]. Here, we carefully implement such gadgets with degree d trees of small height instead of sequences. WLOG, we can assume that all vectors in A start with 1 and all vectors in B start with 0. If it is not so, we can add an extra coordinate at the beginning of every vector and set the entry accordingly. This does not change the answer to the problem (whether there are two orthogonal vectors). Also, we assume that all vectors in A have the same number of entries equal to 1. If it is not so, we can subdivide the set A into smaller sets so that every set contain vectors with the same number of entries equal to 1. Then we run the reduction on every subset of A and B. This increase the runtime to solve the Orthogonal Vectors problem by a factor of $D + 1$ but we are fine with that.

For each vector in the first list, $\alpha \in A$, we construct a vector gadget H_α as follows. Let H'_α be the vector gadget constructed in Lemma 2.1 corresponding to vector $\alpha \in A$. Then H_α is equal to $r \to root(H'_\alpha)$ for some vertex r, which is the root of H_α.

For each vector in the second list, $\beta \in B$, we construct a vector gadget G_β as follows. Let δ be a vector with D coordinates. The first entry is equal to 1 and the rest of entries are equal to 0. Let G'_β be the vector gadget constructed in Lemma 2.1 corresponding to vector $\beta \in B$. Then we obtain G_β by choosing a vertex r to be its root and adding $r \to G'_\delta$ and $r \to G'_\beta$.

The main idea behind this construction is that, when matching H_α and G_β, one has a choice: either match H'_α to G'_δ (giving a fixed score, independent of α), or match it to G'_β (and the score then depends on the orthogonality of α and β.) We make this argument formal in the next lemma. Let E' denote the size of H_α for $\alpha \in A$, which is independent of α since all vectors in A contain the same number of 1's. Let $E = E' - 1$.

LEMMA 2.2. *The largest common subtree of H_α and G_β is of size $E' = |H_\alpha|$ if α, β are orthogonal and it is of size $E = E' - 1$ otherwise. We have that the size of H_α and $H_{\alpha'}$ are equal $|H_\alpha| = |H_{\alpha'}|$ for all $\alpha, \alpha' \in A$.*

Proof. First, if α, β are orthogonal, then by Lemma 2.1 we have that H_α is isomorphic to a subgraph of G_β and the LCST has size E'.

For the second case, assume that α, β are not

orthogonal. We first remark that there is a common subtree of size $E'-1$: Let α' denote α where we set the first coordinate of α (which is equal to 1) to 0, then $H'_{\alpha'}$ is a subtree of H'_α of size $|H'_{\alpha'}| = E'-1$, and by Lemma 2.1, it is also a subtree of G'_δ because $\alpha' \cdot \delta = 0$. It remains to show that we cannot map the entire tree H_α to a subtree of G_β, which follows because H'_α is neither isomorphic to a subtree of G'_δ (since $\alpha \cdot \delta = 1$) nor to a subtree of G'_β (since $\alpha \cdot \beta \neq 0$). □

We are now ready to present the final trees H, G. We construct H as follows. First, we build a complete d-ary tree with N leaves $h_1, \ldots, h_N$ at the lowest level. For every $j \in [N]$, we add $h_j \to H_{\alpha_j}$, where $A = \{\alpha_1, \ldots, \alpha_N\}$. Similarly we construct G. Take a complete d-ary tree with leaves $g_1, \ldots, g_N$ at the lower level. For every $j \in [N]$, we add $g_j \to G_{\beta_j}$, where $B = \{\beta_1, \ldots, \beta_N\}$.

THEOREM 2.4. *The Largest Common Subtree of H and G is of size at most $(2N-1)+(N \cdot E)$ if there is no pair of orthogonal vectors, and is at least $(2N-1)+(N \cdot E+1)$ otherwise.*

Proof. We must map the nodes h_i for every $i \in [N]$ to nodes $g_{\pi(i)}$, for some permutation $\pi : [N] \to [N]$. Notice, however, that π cannot be an arbitrary permutation since, e.g. $\pi(1) = \pi(2) \pm 1$ (the permutation must be implemented by swapping children in a complete binary tree.)

On the one hand, the total size of the common subtree can be upper bounded by the size of a complete binary tree with N leaves, plus $\sum_{i=1}^{N} LCST(H_{\alpha_i}, G_{\beta_{\pi(i)}})$, for an arbitrary permutation π. If there is no pair of orthogonal vectors, then by Lemma 2.2, the latter sum is exactly $N \cdot E$, and the total size is bounded by $(2N-1) + N \cdot E$.

On the other hand, if there is an orthogonal pair α_i, β_j, we can take any mapping in which h_i is mapped to g_j while the other h_x's are mapped arbitrarily to different g_y's. This induces some permutation $\pi : [N] \to [N]$ so that h_x is mapped to $g_{\pi(x)}$. Since $\alpha_i \cdot \beta_j = 0$, Lemma 2.2 implies that this mapping can be completed to a mapping of score

$$\begin{aligned}
&(2N-1) + \sum_{v=1}^{N} LCST(H_{\alpha_v}, G_{\beta_{\pi(v)}}) \;\geq \\
&(2N-1) + (N-1) \cdot E + (E+1) \;= \\
&(2N-1) + (N \cdot E + 1)\ .
\end{aligned}$$

□

□

3 Algorithms

In this section we present new algorithms for Subtree Isomorphism on rooted trees with vertices of bounded degree. Edmonds and Matula independently described a procedure for reducing the rooted Subtree Isomorphism problem to a polynomially bounded collection of recursively smaller Subtree Isomorphism problems, and how to combine the answers by solving a maximum bipartite matching problem (see [45]). We follow the same approach but focus on the case where the degrees are bounded by a constant.

Given two rooted trees H and G, we want to decide whether H is isomorphic to a subtree of G where the root of H maps to the root of G. Let $H_1, H_2, \ldots, H_k$ and $G_1, G_2, \ldots, G_\ell$ be the subtrees of H and G, respectively, with roots that are children of the root of H and the root of G. Let $\mathcal{G}$ be a bipartite graph with vertex set $\mathcal{V} = \{u_1, \ldots, u_k\} \cup \{v_1, \ldots, v_\ell\}$, and let (u_i, v_j) be an edge of $\mathcal{G}$ if and only if H_i is isomorphic to a subtree of G_j. Then H is isomorphic to a subtree of G if and only if $\mathcal{G}$ contains a matching of size k. The Edmonds-Matula procedure constructs the graph $\mathcal{G}$ by recursion and then solves the maximum bipartite matching problem on $\mathcal{G}$.

Designing similar algorithms for rooted Subtree Isomorphism thus involves two challenges: constructing $\mathcal{G}$ and solving the maximum bipartite matching problem on $\mathcal{G}$. The currently fastest randomized algorithm for the maximum bipartite matching problem is due to Mucha and Sankowski [46] and runs in expected time $O((k+\ell)^\omega)$, where $\omega < 2.373$ is the matrix multiplication exponent. Improving this algorithm is itself a challenging open problem.

For constructing the graph $\mathcal{G}$, it is not hard to see that any deterministic algorithm needs to know all edges of $\mathcal{G}$. For randomized algorithms, however, it is not always necessary to know for every pair u_i, v_j whether the edge (u_i, v_j) is in the graph. The expected number of node pair queries ("is the pair an edge in the graph?") that a randomized algorithm needs to make in order to be able to determine whether a perfect matching exists, is known as the *randomized query complexity* (or decision tree complexity) of bipartite perfect matching. It is an easy exercise to check that the randomized query complexity of the problem is $\Omega(k\ell)$. Estimating the exact number of queries is, however, not straightforward. It is not even clear whether $k\ell$ queries are necessary in expectation, or whether $(1-\varepsilon)k\ell$ queries might be sufficient for some $\varepsilon > 0$. Factoring this into the analysis of the maximum bipartite matching algorithm complicates things further.

To simplify things, we restrict our attention to the case where the degrees of the trees are bounded by a constant. In this case we can check in constant time

whether $\mathcal{G}$ contains the desired perfect matching, once a sufficient number of edge queries have been made. We can thus focus solely on the randomized query complexity of the bipartite matching problem and its use in recursive algorithms for the Subtree Isomorphism problem.

It is easy to show that in this case the algorithm of Edmonds and Matula runs in time $O(mn)$, where $|H| = m$ and $|G| = n$. The same algorithm is also able to handle labelled vertices, i.e., each vertex has a label and the labels of H are required to match the labels of the subtree of G. Moreover, the algorithm can solve the largest common subtree problem in $O(mn)$ time as well. This is done by recursively assigning a weight to every edge (u_i, v_j) of $\mathcal{G}$ equal to the size of the largest common subtree of H_i and G_i, and then asking for the matching of largest weight. (We refer to the appendix for a short complexity analysis and further description of these algorithms.) Our lower bounds from theorems 1.1 and 1.5 are thus tight for trees of constant degree.

For the remainder of the section we restrict our attention to trees of constant degree d and height h. We first introduce a randomized algorithm that solves the binary problem in expected time $O(\min\{2.8431^h, mn\})$. For comparison, the corresponding upper bound by Edmonds and Matula [45] is $O(\min\{4^h, mn\})$, i.e., their algorithm makes four recursive calls at each level of the tree. In particular our algorithm is truly subquadratic when $h < 1.3267 \log_2 n$. For $d = 3$ we give a similar, but more complicated case analysis showing that the problem can be solved in expected time $O(\min\{6.107^h, mn\})$, improving the straightforward $O(\min\{9^h, mn\})$ bound by Edmonds and Matula. For $d > 3$ we introduce a randomized algorithm with expected running time upper bounded by $O(\min\{(d^2 - \frac{1}{3}d + \frac{2}{3})^h, mn\})$.

3.1 A faster algorithm for binary trees

For trees with degree at most two, the Edmonds-Matula procedure can be interpreted as follows. Let H_L and H_R be the left and right subtrees of H, and let G_L and G_R be the left and right subtrees of G. H is isomorphic to a subtree of G if and only if one of the following two conditions are true:

1. H_L is isomorphic to a subtree of G_L, and H_R is isomorphic to a sutree of G_R.
2. H_L is isomorphic to a subtree of G_R, and H_R is isomorphic to a sutree of G_L.

Each case can be checked with two recursive calls, and checking whether H is isomorphic to a subtree of G can thus be done with at most four recursive calls, giving an $O(4^h)$ upper bound.

Algorithm RandBinarySubIso(H, G)

1. If $|H| = 0$, return **true**;
2. If $|G| = 0$, return **false**;
3. With probability 1/2 swap H_L and H_R in H;
4. With probability 1/2 swap G_L and G_R in G;
5. If $RandBinarySubIso(H_L, G_L) =$ **false**, then go to step 7;
6. If $RandBinarySubIso(H_R, G_R) =$ **true**, then return **true**;
7. If $RandBinarySubIso(H_L, G_R) =$ **false**, then return **false**;
8. If $RandBinarySubIso(H_R, G_L) =$ **true**, then return **true**. Otherwise return **false**;

Figure 1: A randomized, recursive algorithm for rooted Subtree Isomorphism on binary trees.

Observe that if H_L is not isomorphic to a subtree of G_L, then there is no reason to check whether H_R is isomorphic to a sutree of G_R. Similarly, if the algorithm concludes that the first condition is met, then there is no reason to check the second condition since we already know that H is isomorphic to a subtree of G. Based on these observations, we introduce a simple randomized variant of the algorithm that achieves a significantly better running time by saving recursive calls: Swap H_L and H_R with probability 1/2, and swap G_L and G_R with probability 1/2. Then run the Edmonds-Matula algorithm, but do not perform unnecessary recursive calls. We give a formal description of the algorithm in Figure 1. We refer to the algorithm as *RandBinarySubIso*.

THEOREM 3.1. *The RandBinarySubIso algorithm runs in expected time $O(\min\{2.8431^h, n^2\})$ for trees H and G of size $O(n)$ and height at most h. In particular, it runs in time $O(n^{1.507})$ for trees of height $(1 + o(1)) \cdot \log_2 n$, and is strongly subquadratic for trees of height $h < 1.3267 \log_2 n$.*

Before proving Theorem 3.1 we first prove a useful lemma. Let $T(h)$ be the maximum expected number of times $RandBinarySubIso(H, G)$ makes a recursive call with an empty tree when H and G are arbitrary rooted trees with height at most h. Let $T_{yes}(h)$ and

$T_{no}(h)$ be defined similarly, but under the assumption that the algorithm returns **true** and **false**, respectively. Note that $T(0) = T_{yes}(0) = T_{no}(0) = 1$. Also note that $T(h) = \max\{T_{yes}(h),\ T_{no}(h)\}$.

LEMMA 3.1. *For all $h \geq 0$,*

$$\begin{aligned} T_{yes}(h) &\leq 2.25 \cdot T_{yes}(h-1) + 0.5 \cdot T_{no}(h-1) \ , \\ T_{no}(h) &\leq T_{yes}(h-1) + 2 \cdot T_{no}(h-1) \ . \end{aligned}$$

Proof. To simplify notation we write $H \subseteq G$ when H is isomorphic to a subtree of G, and $H \not\subseteq G$ otherwise.

We first show that $T_{yes}(h) \leq 2.25 \cdot T_{yes}(h-1) + 0.5 \cdot T_{no}(h-1)$. Assume therefore that $H \subseteq G$. With probability $1/2$ we then have $H_L \subseteq G_L$ and $H_R \subseteq G_R$, such that the algorithm returns **true** in line 6 after spending $2 \cdot T_{yes}(h-1)$ time in expectation. On the other hand, with probability $1/2$ the outcomes of lines 5 and 6 depend on the trees in question, and the recursive calls in lines 7 and 8 both return **true** if reached. More precisely, we get three cases that depend on the trees:

(*i*) $H_L \subseteq G_L$ and $H_R \subseteq G_R$: The recursive calls in lines 5 and 6 both return **true**, and the algorithm spends $2 \cdot T_{yes}(h-1)$ time in expectation.

(*ii*) $H_L \not\subseteq G_L$ and $H_R \not\subseteq G_R$: The recursive call in line 5 returns **false**, and the recursive calls in lines 7 and 8 both return **true**. The algorithm spends $T_{no}(h-1) + 2 \cdot T_{yes}(h-1)$ time in expectation.

(*iii*) $H_L \subseteq G_L$ and $H_R \not\subseteq G_R$, or $H_L \not\subseteq G_L$ and $H_R \subseteq G_R$: The recursive call in line 5 returns **false** with probability $1/2$ and **true** with probability $1/2$. In the second case the recursive call in line 6 returns **false**. The recursive calls in lines 7 and 8 both return **true**. The algorithm spends $T_{no}(h-1) + 2.5 \cdot T_{yes}(h-1)$ time in expectation.

The third case thus dominates the two others, and we conclude that $T_{yes}(h) \leq 2.25 \cdot T_{yes}(h-1) + 0.5 \cdot T_{no}(h-1)$.

We next show that $T_{no}(h) \leq T_{yes}(h-1) + 2 \cdot T_{no}(h-1)$. Assume therefore that $H \not\subseteq G$. We get the contribution $2 \cdot T_{no}(h-1)$ as follows. In either line 5 or 6 we get the answer **false** from a recursive call, and in either line 7 or 8 we also get the answer **false** from a recursive call. This amounts to two "no" answers which cost $2 \cdot T_{no}(h-1)$ in expectation. We get the contribution $T_{yes}(h-1)$ as follows. With probability at most $1/2$ we get the answer **true** in line 5 (which means that we get **false** in line 6). Similarly, with probability at most $1/2$ we get the answer **true** in line 7 (which means that we get **false** in line 8). In total, we get that $T_{no}(h) \leq 2 \cdot T_{no}(h-1) + \frac{1}{2} T_{yes}(h-1) + \frac{1}{2} T_{yes}(h-1)$. □

Proof. [Proof of Theorem 3.1] Lemma 3.1 gives us that

$$\begin{aligned} \begin{pmatrix} T_{yes}(h) \\ T_{no}(h) \end{pmatrix} &\leq \begin{pmatrix} 2.25 & 0.5 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} T_{yes}(h-1) \\ T_{no}(h-1) \end{pmatrix} \\ &\leq \begin{pmatrix} 2.25 & 0.5 \\ 1 & 2 \end{pmatrix}^h \begin{pmatrix} 1 \\ 1 \end{pmatrix} . \end{aligned}$$

A diagonalization of the matrix yields

$$\begin{pmatrix} 2.25 & 0.5 \\ 1 & 2 \end{pmatrix} = Q^{-1} J Q \ ,$$

where

$$\begin{aligned} Q^{-1} &= \begin{pmatrix} \frac{1-\sqrt{33}}{8} & \frac{1-\sqrt{33}}{8} \\ 1 & 1 \end{pmatrix} \\ J &= \begin{pmatrix} \frac{17-\sqrt{33}}{8} & 0 \\ 0 & \frac{17+\sqrt{33}}{8} \end{pmatrix} \\ Q &= \begin{pmatrix} -\frac{4}{\sqrt{33}} & \frac{1}{2} + \frac{1}{2\sqrt{33}} \\ \frac{4}{\sqrt{33}} & \frac{1}{2} - \frac{1}{2\sqrt{33}} \end{pmatrix} , \end{aligned}$$

and therefore

$$\begin{pmatrix} T_{yes}(h) \\ T_{no}(h) \end{pmatrix} \leq \begin{pmatrix} 0.065 \cdot 1.407^h + 0.94 \cdot 2.8431^h \\ -0.109 \cdot 1.407^h + 1.109 \cdot 2.8431^h \end{pmatrix} .$$

Thus, $T(h) = O(2.8431^h)$, which proves the theorem. □

3.2 A Faster Algorithm for Ternary Trees

Here we discuss the subtree isomorphism problem for rooted ternary trees. We prove Theorem 1.3 by showing that Subtree isomorphism for rooted ternary trees of height h can be solved in expected time $O(6.107^h)$.

We note that this running time is lower than the runtime given by our generic algorithm for constant degree trees in Section 3.3.

Let's prove the theorem. As before, we proceed by a recursive approach. In each recursive call, we consider a randomized decision tree for 3×3 bipartite perfect matching, where each query corresponds to a recursive call on height one less. We then analyze the runtime similar to the binary tree case: we distinguish between the "yes" and "no" case of the query answer, and write the running time as two recurrences, one for T_{yes}, when the algorithm said the trees are isomorphic, and one for T_{no} when they were not. We analyze the randomized decision tree in terms of the expected number of "yes" and "no" query answers in the worst case.

The randomized query protocol is as follows. Let U and V be the two partitions of the bipartite matching instance (respectively, U are the subtrees of the root of one tree and V are the subtrees of the root of the other). First we pick U or V at random w.p. $1/2$. If we pick

V, then the names of U and V are swapped. Now, with probability 1/6 we pick a permutation of the vertices in U, and with probability 1/6 we pick a permutation of V. From this point on, the protocol is deterministic. Let a, b, c be the nodes of U and x, y, z be the nodes of V. The deterministic decision tree we use is depicted in Figures 2 and 3.

For each of the 2^9 choices for the answers to the 9 edge queries in the 3×3 matching instance, we consider each of the 72 randomized choices as described above (swap U and V, permute U and V) and consider the decision tree, computing the expected number of "yes" and "no" calls. Using a computer program, we establish that when the instance has no perfect matching, the expected number of "yes" calls is always at most 26/9, and the expected number of "no" calls is always at most 37/9; this happens when the complement of the graph consists of a 4-cycle, disjoint from a single edge. On the other hand, if the instance has a perfect matching there are two cases that dominate all others: when the expected number of "yes" calls is 131/36, and the expected number of "no" calls is 61/36, or when the expected number of "yes" calls is 133/36, and the expected number of "no" calls is 5/3. There are thus two options for the recurrence relation, and one of them dominates the other. We present the recurrence that achieves the maximum, and hence gives the worst-case expected runtime for the ternary case.

$$\begin{pmatrix} T_{yes}(h) \\ T_{no}(h) \end{pmatrix} \leq \begin{pmatrix} 133/36 & 5/3 \\ 26/9 & 37/9 \end{pmatrix} \begin{pmatrix} T_{yes}(h-1) \\ T_{no}(h-1) \end{pmatrix} \leq \begin{pmatrix} 133/36 & 5/3 \\ 26/9 & 37/9 \end{pmatrix}^h \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The diagonalization yields

$$\begin{pmatrix} 133/36 & 5/3 \\ 26/9 & 37/9 \end{pmatrix} = Q^{-1} J Q,$$

where

$$J = \begin{pmatrix} \frac{281-\sqrt{25185}}{72} & 0 \\ 0 & \frac{281+\sqrt{25185}}{72} \end{pmatrix}$$

$$Q = \begin{pmatrix} \frac{1}{2} + \frac{\sqrt{25185}}{3358} & \frac{1}{2} - \frac{\sqrt{25185}}{3358} \\ -\frac{104}{\sqrt{25185}} & \frac{104}{\sqrt{25185}} \end{pmatrix},$$

which gives that

$$\begin{pmatrix} T_{yes}(h) \\ T_{no}(h) \end{pmatrix} \leq \begin{pmatrix} 0.17 \cdot 1.7^h + 0.831 \cdot 6.107^h \\ -0.2 \cdot 1.69^h + 1.21 \cdot 6.107^h \end{pmatrix}.$$

Thus, the running time overall is $O(6.107^h)$.

3.3 An algorithm for any constant degree

In this section we describe a way to use randomization to save subtree comparisons in the Edmonds-Matula algorithm [45] for all degrees $d > 2$. Recall that the algorithm works as follows. Given two trees H and G of constant degree d, the goal is to decides whether H is isomorphic to a subtree of G by using recursion. If the roots of either H or G have less than d children, we simply view the missing subtrees as being a special empty subtree.

1. Let $H_1, \ldots, H_d$ be the d subtrees of H, and let $G_1, \ldots, G_d$ be the d subtrees of G;
2. Build a bipartite graph $\mathcal{G}$ with d vertices $\mathcal{U} = \{u_1, \ldots, u_d\}$ on the left and d vertices $\mathcal{W} = \{v_1, \ldots, v_d\}$ on the right. For all $i, j \in [d]$, connect u_i and v_j if and only if H_i is isomorphic to a subtree of G_j. We decide which edges appear in the graph recursively.
3. Output that H is isomorphic to a subtree of G if and only if there is a perfect matching in the bipartite graph $\mathcal{G}$.

The runtime of the algorithm is $O(\min\{d^{2h}, n^2\})$, where h is the height. Intuitively, we can improve the runtime of the algorithm as follows. Perform recursive calls corresponding to edges (u_i, v_j) in a random order, and stop as soon as we either detect a perfect matching or rule out the existence of a perfect matching. It is not difficult to show that this randomized version of the algorithm performs $d^2 - \Omega(1)$ recursive calls in expectation out of the d^2 possible calls. That is, in expectation, we save at least a constant number of recursive calls. This implies that the algorithm runs in $O((d^2 - \Omega(1))^h)$ expected time, which is faster than the deterministic algorithm. However, we prove below that we can save $\Omega(d)$ recursive calls in expectation using a slightly different variant of the randomized algorithm.

LEMMA 3.2. *Let $\mathcal{G}$ be a bipartite graph with d vertices $\mathcal{U} = \{u_1, \ldots, u_d\}$ on the left and d vertices $\mathcal{W} = \{v_1, \ldots, v_d\}$ on the right, and suppose we are given query access to the adjacency matrix of $\mathcal{G}$. There is a randomized query algorithm that decides whether $\mathcal{G}$ contains a perfect matching by making $d^2 - \frac{1}{3}d + \frac{2}{3}$ queries in expectation, with probability 0 of making an error.*

We use the following two claims to prove the lemma.

CLAIM 3.1. *Assume that $\mathcal{G}$ has a perfect matching. Then the following algorithm finds a perfect matching after making $d^2 - d + 2$ expected queries: Query edges*

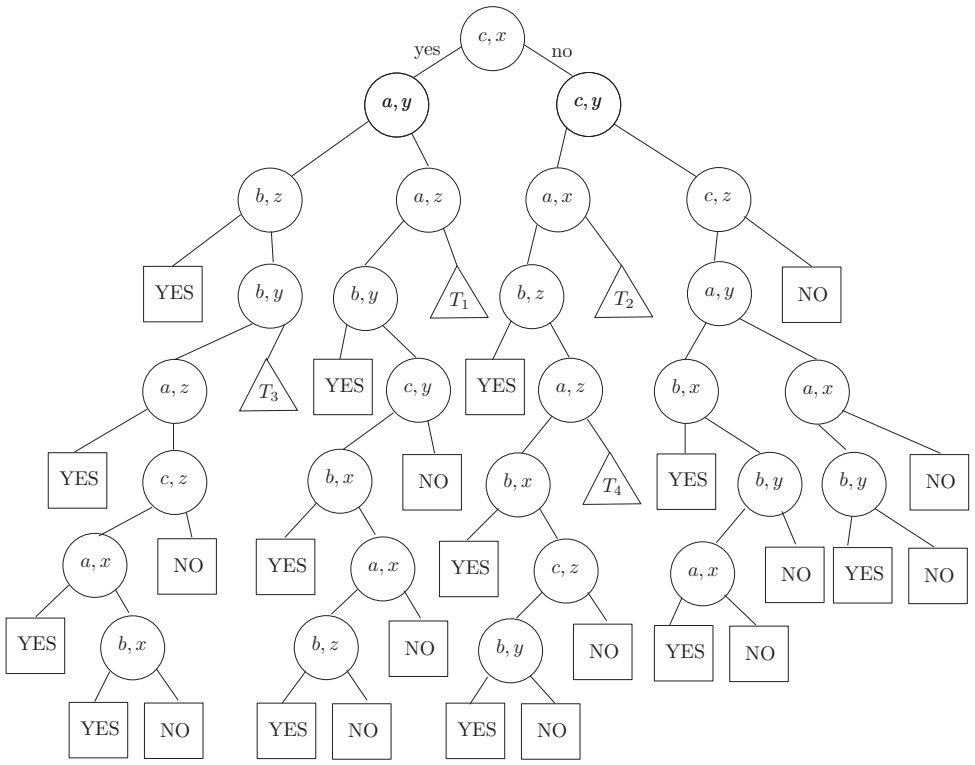

Figure 2: The decision tree used for bipartite matching in the degree 3 case.

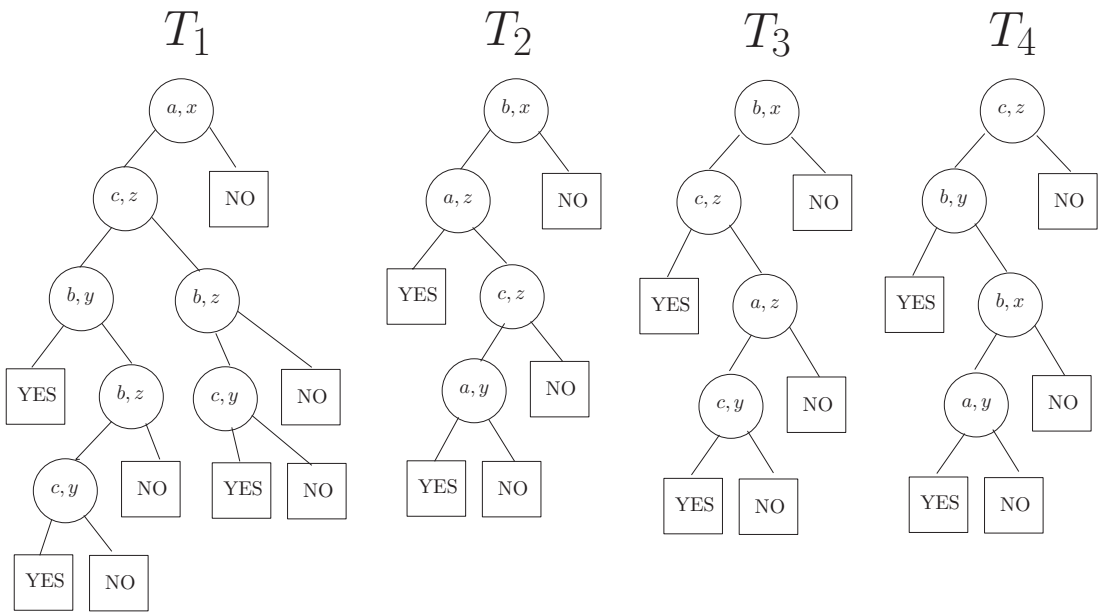

Figure 3: The missing subtrees of the decision tree used for bipartite matching in the degree 3 case.

(u_i, v_j) *in a random order, and stop when finding a perfect matching.*

Proof. Fix a perfect matching present in $\mathcal{G}$ and call its d edges "marked". We stop when all marked edges have been queried. There are $d^2 - d$ unmarked edges. The probability that a given unmarked edge is not queried is $\frac{1}{d+1}$. Therefore, the expected number of unqueried, unmarked edges is $\frac{d^2-d}{d+1} \geq d - 2$. □

CLAIM 3.2. *Assume that $\mathcal{G}$ does not have a perfect matching. Then the following algorithm makes at most $d^2 - \frac{1}{2}d + 1$ queries in expectation before determining that $\mathcal{G}$ does not contain a perfect matching.*

1. *With probability 1/2 swap $\mathcal{U}$ and $\mathcal{W}$;*
2. *Randomly permute the vertices of $\mathcal{U} = \{u_1, u_2, \ldots, u_d\}$;*
3. *Query all edges adjacent to u_i for i going from 1 to d, but stop when ruling out the existence of a perfect matching, i.e., stop when the set of processed vertices $S = \{u_1, \ldots, u_i\}$ contains a subset S' with a neighbourhood $N(S')$ that is smaller than the size of S'.*

Proof. Consider the sets $\mathcal{U}$ and $\mathcal{W}$ prior to running the algorithm. By Hall's theorem, the set $\mathcal{U}$ contains a set S' such that $|N(S')| < |S'|$. We can assume that $|S'| = |N(S')| + 1$, since otherwise we can iteratively remove a vertex from S' until this condition is satisfied. Consider two cases.

- d is even: If $|S'| \geq \frac{d}{2}+1$, we define $T' = \mathcal{W} \setminus N(S')$. Because $N(S') \geq d/2$, we get that $|T'| \leq \frac{d}{2}$. By our construction of T', we have that $N(T') \subseteq \mathcal{U} \setminus S'$ and, as a result, $|N(T')| < |T'|$. Given the first step of the algorithm, with probability at least 1/2 the set $\mathcal{U}$ therefore contains a set S' such that $|N(S')| < |S'| \leq \frac{d}{2}$.
- d is odd: It follows from as similar argument that, with probability at least 1/2, the set $\mathcal{U}$ contains a set S' such that $|N(S')| < |S'| \leq \frac{d+1}{2}$.

We now condition on the set $\mathcal{U}$ containing S' with $|S'| \leq \frac{d+1}{2}$ and $|N(S')| < |S'|$.

The algorithm stops once it queries all vertices from S', since a perfect matching is then ruled out by Hall's theorem. The probability that we do not process a given vertex before processing all vertices in S' is $1/(|S'| + 1)$. Therefore the expected number of unprocessed vertices when the algorithm stops is at least

$$(d - |S'|) \cdot \frac{1}{|S'|+1} \;\geq\; \frac{d-1}{2} \cdot \frac{1}{\frac{d+1}{2}+1} \;=\; \frac{d-1}{d+3} \;.$$

Hence, with probability 1/2, we query $d\left(d - \frac{d-1}{d+3}\right)$ edges, and overall the number of queried edges is

$$\begin{aligned} \frac{1}{2}\left[d\left(d - \frac{d-1}{d+3}\right)\right] + \frac{1}{2}d^2 \;&=\; d^2\left(1 - \frac{1 - \frac{1}{d}}{2(d+3)}\right) \\ &\leq\; d^2 - \frac{1}{2}d + 1 \;. \end{aligned}$$

In the last inequality we use that $d \geq 3$. □

Proof. [Proof of Lemma 3.2]

We prove the lemma by using claims 3.1 and 3.2.

With probability 1/3 we run the algorithm from Claim 3.1 and with probability 2/3 we run the algorithm from Claim 3.2. Consider the case when $\mathcal{G}$ has a perfect matching. Then the expected number of edges queried is upper bounded by

$$\frac{1}{3}(d^2 - d + 2) + \frac{2}{3}d^2 \;=\; d^2 - \frac{1}{3}d + \frac{2}{3} \;.$$

On the other hand, for the case when $\mathcal{G}$ does not contain a perfect matching, the expected number of edges queried is upper bounded by

$$\frac{1}{3}d^2 + \frac{2}{3}\left(d^2 - \frac{1}{2}d + 1\right) \;=\; d^2 - \frac{1}{3}d + \frac{2}{3} \;.$$

Overall, regardless of $\mathcal{G}$, we therefore query at most $d^2 - \frac{1}{3}d + \frac{2}{3}$ edges in expectation. □

THEOREM 3.2. *There is a randomized algorithm that solves Subtree Isomorphism on two rooted trees of size $O(n)$, constant degree d, and height at most h in expected time $O\left(\left(d^2 - \frac{1}{3}d + \frac{2}{3}\right)^h\right)$. In particular, the algorithm is strongly subquadratic for trees of height*

$$h \;\leq\; \left(\frac{2\log d}{\log(d^2 - \frac{1}{3}d + \frac{2}{3})} - \epsilon\right) \cdot \log_d n \;,$$

for any constant $\epsilon > 0$.

Proof. We run the following randomized, recursive algorithm that decides whether H is isomorphic to a subtree of G.

1. Let $H_1, \ldots, H_d$ be the d subtrees of H, and let $G_1, \ldots, G_d$ be the d subtrees of G;
2. Let $\mathcal{G}$ be a bipartite graph with d vertices $\mathcal{U} = \{u_1, \ldots, u_d\}$ on the left and d vertices $\mathcal{W} = \{v_1, \ldots, v_d\}$ on the right. For all $i, j \in [d]$, let u_i and v_j be connected if and only if H_i is isomorphic to a subtree of G_j.
3. Decide whether the graph $\mathcal{G}$ has a perfect matching by running the algorithm from Lemma 3.2. Whenever we need to decide whether an edge (u_i, v_j) is present in $\mathcal{G}$, do it recursively.

By the proof of Lemma 3.2, it suffices to query $d^2 - \frac{1}{3}d + \frac{2}{3}$ edges for every level. Given that the height of the trees is upper bounded by h, we get the desired running time. □

Acknowledgements. We would like to thank Shiri Chechik, Piotr Indyk, Haim Kaplan, Michael Kapralov, Huacheng Yu, and Uri Zwick for many helpful discussions.

References

[1] A. Abboud, A. Backurs, and V. Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. In *Proc. of the 56th FOCS*, 2015.

[2] A. Abboud, F. Grandoni, and V. V. Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proc. of the 26th SODA*, pages 1681–1697, 2015.

[3] A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. *Proc. of the 55th FOCS*, pages 434–443, 2014.

[4] A. Abboud, V. V. Williams, and J. R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. *CoRR*, abs/1506.01799, 2015.

[5] A. Abboud, V. V. Williams, and O. Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

[6] T. Akutsu and M. M. Halldórsson. On the approximation of largest common subtrees and largest common point sets. *Theoretical Computer Science*, 233(1):33–50, 2000.

[7] T. Akutsu, T. Tamura, A. A. Melkman, and A. Takasu. On the complexity of finding a largest common subtree of bounded degree. *Theoretical Computer Science*, 590:2–16, 2014.

[8] A. Ambainis, A. M. Childs, B. Reichardt, R. Spalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM J. Comput.*, 39(6):2513–2530, 2010.

[9] A. Backurs and P. Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proc. of the 47th STOC*, pages 51–58, 2015.

[10] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1–3):217–239, 2005.

[11] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless seth fails. In *Proc. of the 55th FOCS*, pages 661–670, 2014.

[12] K. Bringmann and M. Kunnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. of the 56th FOCS*, 2015.

[13] J. Cheriyan. Randomized O(M(|V|)) algorithms for problems in matching theory. *SIAM Journal on Computing*, 26(6):1635–1655, 1997.

[14] M. J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987.

[15] R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. In *Proc. of the 29th STOC*, pages 66–75, 1997.

[16] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proc. of the 34th STOC*, pages 592–601, 2002.

[17] R. Cole and R. Hariharan. Tree pattern matching to subset matching in linear time. *SIAM Journal on Computing*, 32(4):1056–1066, 2003.

[18] M. Cygan, J. Pachocki, and A. Socala. The hardness of subgraph isomorphism. *CoRR*, abs/1504.02876, 2015.

[19] A. Dessmark, A. Lingas, and A. Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. *Algorithmica*, 27(3-4):337–347, 2000.

[20] Y. Dinitz, A. Itai, and M. Rodeh. On an algorithm of zemlyachenko for subtree isomorphism. *Information Processing Letters*, 70(3):141–146, 1999.

[21] M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. *Journal of the ACM (JACM)*, 41(2):205–213, 1994.

[22] F. L. Gall. Powers of tensors and fast matrix multiplication. In *Proc. of the 39th ISSAC*, pages 296–303, 2014.

[23] B. Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[24] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. W. H. Freeman, 2002.

[25] A. C. Giannopoulou, G. B. Mertzios, and R. Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *CoRR*, abs/1506.01652, 2015.

[26] P. B. Gibbons, R. M. Karp, G. L. Miller, and D. Soroker. Subtree isomorphism is in random NC. *Discrete Applied Mathematics*, 29(1):35–62, 1990.

[27] D. Gusfield. *Algorithms on strings, trees and sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[28] C. M. Hoffmann and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM (JACM)*, 29(1):68–95, 1982.

[29] J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In *Proc. of Complexity of Computer Computations*, pages 131–152. 1972.

[30] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[31] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[32] P. Indyk. Deterministic superimposed coding with applications to pattern matching. In *Proc. of the 38th FOCS*, pages 127–136, 1997.

[33] P. Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proc. of the 39th FOCS*, pages 166–173, 1998.
[34] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.
[35] S. Khanna, R. Motwani, and F. F. Yao. Approximation algorithms for the largest common subtree problem. Technical report, Stanford University, 1995.
[36] P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.
[37] S. R. Kosaraju. Efficient tree pattern matching (preliminary version). In *Proc. of the 30th FOCS*, pages 178–183, 1989.
[38] A. Lingas. An application of maximum bipartite c-matching to subtree isomorphism. In *Proc. of the 8th CAAP*, pages 284–299, 1983.
[39] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63(3):295–302, 1989.
[40] A. Lingas and M. Karpinski. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.
[41] L. Lovász and M. D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.
[42] D. Marx and M. Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism (but were afraid to ask). In *Proc. of the 31st STACS*, pages 542–553, 2014.
[43] J. Matoušek and R. Thomas. On the complexity of finding iso-and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1):343–364, 1992.
[44] D. W. Matula. An algorithm for subtree identification. *SIAM Review*, 10:273–274, 1968.
[45] D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. In *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91–106. Elsevier, 1978.
[46] M. Mucha and P. Sankowski. Maximum matchings via gaussian elimination. In *Proc. of the 45th FOCS*, pages 248–255, 2004.
[47] M. Patrascu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. of the 21st SODA*, volume 10, pages 1065–1075, 2010.
[48] S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal on Computing*, 6(4):730–732, 1977.
[49] L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. of the 45th STOC*, pages 515–524, 2013.
[50] M. Saks and A. Wigderson. Probabilistic boolean decision trees and the complexity of evaluating game trees. In *Proc. of the 27th FOCS*, pages 29–38, 1986.
[51] R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.
[52] K.-C. Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):422–433, 1979.
[53] G. Valiente. *Algorithms on trees and graphs*. Springer Science & Business Media, 2013.
[54] R. M. Verma. Strings, trees, and patterns. *Information Processing Letters*, 41(3):157–161, 1992.
[55] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.
[56] V. V. Williams. Multiplying matrices faster than coppersmith-winograd. In *Proc. of the 44th STOC*, pages 887–898, 2012.
[57] K. Zhang and T. Jiang. Some max snp-hard results concerning unordered labeled trees. *Information Processing Letters*, 49(5):249–254, 1994.
[58] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.
[59] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information processing letters*, 42(3):133–139, 1992.

A Analysis of the Edmonds-Matula algorithm and its variants

LEMMA A.1. *On binary trees, the Edmonds-Matula algorithm takes $O(mn)$ time, where $m = |H|$, $n = |G|$.*

Proof. Denote by m_L, m_R, n_L, n_R the sizes of H_L, H_R, G_L, G_R, the left and right subtrees of H and G, notice that $m_L + m_R = m - 1$, $n_L + n_R = n - 1$. The runtime of the algorithm is described by the recurrence

$$\begin{aligned} T(0,n) &= T(m,0) = 1\ , \\ T(m,n) &= 1 + T(m_L,n_L) + T(m_R,n_R) + \\ &\qquad T(m_L,n_R) + T(m_R,n_L)\ . \end{aligned}$$

Then, by induction, we prove $T(m,n) \le mn$,

$$\begin{aligned} T(m,n) &= 1 + T(m_L,n_L) + T(m_R,n_R) + \\ &\qquad T(m_L,n_R) + T(m_R,n_L) \\ &\le 1 + m_L \cdot n_L + m_R \cdot n_R + m_L \cdot n_R + m_R \cdot n_L \\ &= 1 + (m_L + m_R) \cdot (n_L + n_R) \\ &= 1 + (m-1)(n-1) \\ &\le mn\ . \end{aligned}$$

□

As mentioned in section 3, this algorithm is easily extended to solve the labelled version of the problem or the *Largest Common Subtree* problem for any constant bounded degree $d = O(1)$. For completeness, we include pseudo-code of a variant that solves the *Labelled Largest Common Subtree* problem, generalizing both.

Algorithm 1: $LLCS(H, G, d)$

if $Size(F) = 0$ **or** $Size(G) = 0$ **then**
  return 0
end if
for $i = 1$ to d **do**
  for $j = 1$ to d **do**
    if $Label(H.Children[i]) = Label(G.Children[j])$ **then**
      $Sub[i, j] \leftarrow LLCS(Subtree(H.Children[i]),$
        $Subtree(G.Children[j]),\ d)$
    else
      $Sub[i, j] \leftarrow 0$
    end if
  end for
end for
$w \leftarrow$ the weight of a maximum weight bipartite matching in the bipartite graph with edges defined by $Sub[i, j]$.
return $w + 1$

LEMMA A.2. *Algorithm 1 solves the* Labelled Largest Common Subtree *problem in time $O(mn)$ for rooted trees H,G of bounded degree $d = O(1)$, where m, n are the sizes of H, G respectively.*

Proof. Correctness is straightforward, it is also clear that as $d = O(1)$, Algorithm 1 makes a constant number of operations excluding the recursive calls. Denote by $m_1, m_2, ..., m_r$ the sizes of the (maximal) subtrees rooted at the $r \leq d$ children of the root of H, and by $n_1, n_2, ..., n_s$ the sizes of those rooted at the $s \leq d$ children of the root of G. It holds that $\sum_{i=1}^{r} m_i = m-1$ and $\sum_{j=1}^{s} n_j = n - 1$. The runtime of the algorithm is described by the recurrence

$$\begin{aligned} T(0, n) &= T(m, 0) = 1 , \\ T(m, n) &= 1 + \sum_{i=1,j=1}^{r,s} T(m_i, n_j) . \end{aligned}$$

Then, by induction, we prove $T(m, n) \leq mn$,

$$\begin{aligned} T(m, n) &= 1 + \sum_{i=1,j=1}^{r,s} T(m_i, n_j) \\ &\leq 1 + \sum_{i=1,j=1}^{r,s} m_i \cdot n_j \\ &= 1 + (\sum_{i=1}^{r} m_i) \cdot (\sum_{j=1}^{s} n_j) \\ &= 1 + (m-1)(n-1) \\ &\leq mn . \end{aligned}$$

□