Faster Subtree Isomorphism *

Ron Shamir Dekel Tsur

Tel-Aviv University

Dept. of Computer Science

Tel Aviv 69978, Israel

E-Mail: {shamir,dekel}@math.tau.ac.il

March 1998, Revised May 1999

*A preliminary version of this paper has appeared in the Fifth Israel Symposium on Theory of Computing and Systems (ISTCS 97)

Send proofs to:

Dekel Tsur

Dept. of Computer Science

School of Mathematics

Tel Aviv University

Tel Aviv, 69978

Israel

Fax: +972–3–640–9357

Abstract

We study the subtree isomorphism problem: Given trees H and G, find a subtree of G which is isomorphic to H or decide that there is no such subtree. We give an $O(\frac{k^{1.5}}{\log k}n)$-time algorithm for this problem, where k and n are the number of vertices in H and G respectively. This improves over the $O(k^{1.5}n)$ algorithms of Chung and Matula. We also give a randomized (Las Vegas) $O(k^{1.376}n)$-time algorithm for the decision problem.

1 Introduction

A fundamental problem in graph theory is the *subgraph isomorphism problem*: Given two graphs G and H, find a subgraph of G which is isomorphic to H (if there is one). This problem is NP-hard as it includes for example the clique problem.

When restricting the graph G, the subgraph isomorphism problem may become easier. Polynomial-time algorithms for subgraph isomorphism problem were given for trees [20], two-connected outerplanar graphs [16], and two-connected series-parallel graphs [17]. All these graphs have a treewidth at most 2. The subgraph isomorphism problem is NP-hard when the graph G is an arbitrary graph with treewidth at most 2 [19]. However, the problem is polynomial for the family of graphs with treewidth at most p for every $p \geq 2$, if we also add the restriction that G is p-connected, or that H has bounded degree [19]. A faster algorithm for the former case was given in [6].

In this paper we study *subtree isomorphism*, i.e., the subgraph isomorphism problem when G and H are trees. Figure 1 gives an instance of this problem. Throughout this work n and k denote the number of vertices in G and H, respectively. When $k = n$ we get the *tree isomorphism problem*, which has a linear time algorithm due to Hopcroft and Tarjan [13]. Polynomial algorithms for subtree isomorphism were first given by Matula [20] and by Edmonds (cf. [21]). Faster algorithms, with worst case time complexity of $O(k^{1.5}n)$, were given by Matula [21] and Chung [4]. In contrast, the subgraph isomorphism problem is NP-complete when G is a tree and H is a forest (*subforest isomorphism* [10]). The subgraph isomorphism and the subtree isomorphism problems have application in pattern recognition, computational biology and chemistry [1, 24]. Our main result here is an $O(\frac{k^{1.5}}{\log k}n)$-time algorithm. Our algorithm, like most previous studies of the problem, is based on the close relationship between subtree isomorphism and maximum matching in bipartite graphs. (This relationship was also utilized to obtain fast

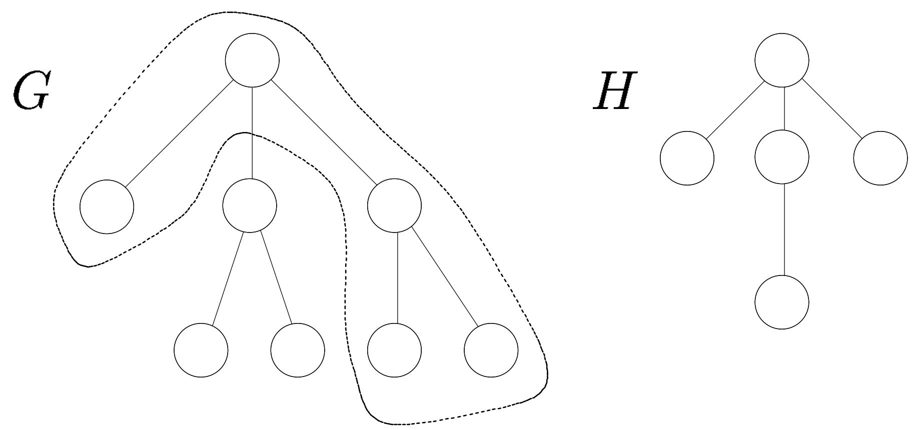

Figure 1: An instance of the subtree isomorphism problem. The tree G has a subtree which is isomorphic to H.

parallel algorithms for subtree isomorphism [11, 15].) The subtree isomorphism problem is recursively translated into a collection of smaller subtree isomorphism problems, that are solved using maximum matching algorithms. The improved complexity is achieved by a combinatorial lemma that bounds the possible number of distinct subtrees involved, and by using the notion of clique partition and its application by Feder and Motwani to finding maximum matching in bipartite graphs [9]. We show that for the matching problem resulting from the subtree isomorphism problem, we can find a clique partition in a simple way. The ideas we use here also give an improved algorithm for the maximum subforest problem (finding a subforest of a given tree with maximum number of edges which does not have a subgraph isomorphic to a tree from a given set of trees) [23].

We also give a randomized (Las Vegas) algorithm algorithm for the decision problem whose expected running time is $O(k^{\omega-1}n)$ where ω is the exponent of matrix multiplication. Using the best known bound for ω [5], the above bound is $O(k^{1.376}n)$. This algorithm follows directly from a randomized algorithm for the cardinality of a maximum matching given by Cheriyan [3].

The rest of the paper is organized as follows: Section 2 describes an $O(k^{1.5}n)$ algorithm for subtree isomorphism. In Section 3 we prove some simple lemmas on matching that are needed in later sections. In Section 4 we prove the combinatorial lemma and use it together with the clique partition in order to improve the running time of the algorithm from Section 2. Finally, Section 5 describes the randomized algorithm.

We finish this section with some basic definitions. For more definitions and basic graph terminology see e.g. [2]. A *rooted tree* is a triplet $G = (V, E, r)$ where (V, E) is an unrooted tree, and r is some vertex in V which is called the *root*. We will sometime write G^r to denote that r is the root of the rooted tree G. Also, for an unrooted tree G, we denote by G^r the rooted tree formed by choosing the vertex r to be its root. We denote by G^r_v the rooted subtree of G^r whose vertices are all descendants of v, and its root is v.

We say that two rooted trees G^r and $H^{r'}$ are *isomorphic* if there is an isomorphism between G and H which maps r to r'. Likewise, we write $H^{r'} \subseteq_R G^r$ if there is a rooted subtree J^r of G^r which is isomorphic to $H^{r'}$ (note that the subtree J^r must have the same root as G^r).

We define the *open neighborhood* of a vertex v in graph G by $N(v) = \{u : uv \in E\}$, and the *closed neighborhood* by $N[v] = N(v) \cup \{v\}$. We use $\mathrm{d}_G(v)$ to denote the degree of a vertex v in a graph G (or $\mathrm{d}(v)$ if G is clear from the context).

2 An $O(k^{1.5}n)$ algorithm

In this section we describe an $O(k^{1.5}n)$ algorithm for the subtree isomorphism problem that will be the basis for the improved algorithms in sections 4 and 5. It is based on Chung's algorithm [4] and has the same asymptotic time complexity, but is simpler as the tree G is traversed only once, compared to twice in Chung's algorithm. We briefly describe the algorithm for completeness. We note that the improvements of sections 4 and 5 can also be applied to Chung's original algorithm. For simplicity, we describe an

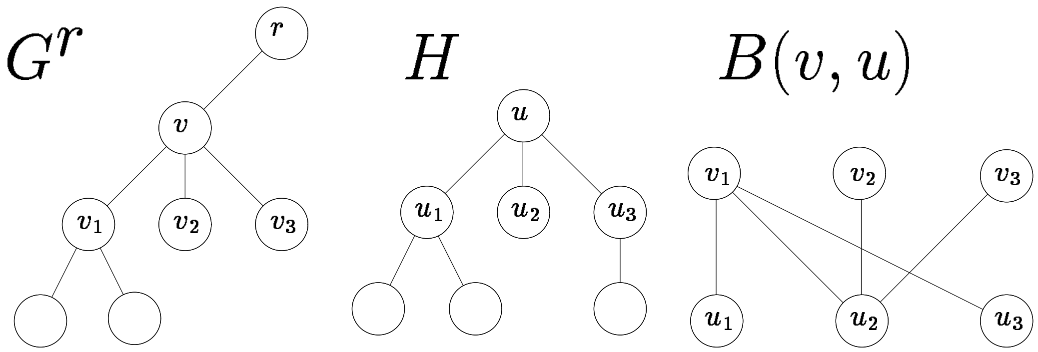

Figure 2: An instance of subtree isomorphism. Here, we have $H^u, H_u^{u_2} \not\subseteq_R G_v^r$ and $H_u^{u_1}, H_u^{u_3} \subseteq_R G_v^r$, so $S(v,u) = \{u_1, u_3\}$. The graph $B(v,u)$ is the bipartite graph which we construct in order to compute $S(v,u)$. There is an edge $u_i v_j$ in this graph iff $u \in S(v_j, u_i)$. $H^u \not\subseteq_R G_v^r$ as $B(v,u)$ does not contain a matching of size 3. $H_u^{u_1} \subseteq_R G_v^r$ as $B_{u_1}(v,u) = B(v,u) - u_1$ contains a matching of size 2.

algorithm for the decision problem. It can be extended easily to an algorithm for the search problem.

Let $G = (V,E), H = (V_H, E_H)$ be the input trees (w.l.o.g. $|V_H| > 1$). Select a vertex r of G to be the root. We wish to know for each $v \in V$,$u \in V_H$ whether $H^u \subseteq_R G_v^r$. In order to compute this efficiently we will also need to know for each neighbor w of u if $H_u^w \subseteq_R G_v^r$ (notice that H_u^w is the graph formed by removing H_w^u from H^u). This information is relevant because $H^u \subseteq_R G_v^r$ iff for every child u' of u there is a distinct child v' of v such that $H_{u'}^u \subseteq_R G_{v'}^r$. More generally, we have the following lemma:

Lemma 2.1 *For any vertex v in G^r, vertex u in H, and a vertex $w \in N[u]$ we have that $H_u^w \subseteq_R G_v^r$ iff for every child u' of u in H_u^w there is a distinct child v' of v such that $H_{u'}^u \subseteq_R G_{v'}^r$.*

We store this information in sets $S(v,u)$ defined as follows: for every $v \in V$, and for every $u \in V_H$

$$S(v,u) = \{w \in N[u] : H_u^w \subseteq_R G_v^r\}.$$

See figure 2 for an example for this definition. Notice that (1) $u \in S(v,u)$ if and only if $H^u = H^u_u \subseteq_R G^r_v$, (2) $u \in S(v,u)$ implies $S(v,u) = N[u]$, and (3) $\mathrm{d}(v) < \mathrm{d}(u) - 1$ implies $S(v,u) = \phi$.

The algorithm computes the sets $S(v,u)$ for all v and u. We show how to compute $S(v,u)$ inductively by going over the vertices v of G^r from the leaves to the root and computing $S(v,u)$ for all $u \in V_H$ (the algorithm is also described using pseudo-code in figure 3). The base of the induction is when v is a leaf of G^r. Then $S(v,u)$ can be computed for all u as follows: If u is a leaf of H, then $S(v,u)$ consists of one vertex which is the single neighbor of u in H. Otherwise, if u is an internal vertex, $S(v,u)$ is empty.

For the inductive step, consider an internal vertex v (from (3) we can assume that $\mathrm{d}(v) \geq \mathrm{d}(u) - 1$). We first need to compute $S(v',u)$ for all the children v' of v, for all $u \in V_H$. Then, in order to decide for $w \in N[u]$ if $w \in S(v,u)$, we construct a bipartite graph $B_w(v,u)$ with the two parts $X^{v,u}_w$ and $Y^{v,u}$, where $X^{v,u}_w$ is the set of children of u in H^w_u, $Y^{v,u}$ is the set of children of v, and $u'v'$ is an edge of $B_w(v,u)$ iff $H^u_{u'} \subseteq_R G^r_{v'}$ (i.e., iff $u \in S(v',u')$). By Lemma 2.1, w is in $S(v,u)$ iff $B_w(v,u)$ has a matching of size $|X^{v,u}_w|$. Therefore, in order to compute $S(v,u)$ we need to find maximum matchings in $\mathrm{d}(u)+1$ bipartite graphs. However, all these graphs are similar one to another: Each graph $B_w(v,u)$ (for $w \neq u$) is obtained by deleting the vertex w from the graph $B_u(v,u)$. In Section 3 (Corollary 3.2) we shall show that it suffices to find a maximum matching only in $B_u(v,u)$, and then we can efficiently compute the size of the maximum matching in $B_w(v,u)$ for all $w \neq u$. In the following, we will use $B(v,u)$ instead of $B_u(v,u)$. See figure 2 for an example of the relation between $S(v,u)$ and the graph $B(v,u)$.

Algorithm Subtree-Isomorphism is described by the pseudo-code shown in figure 3.

Theorem 2.2 *Algorithm Subtree-Isomorphism solves the subtree isomorphism problem in $O(k^{1.5}n)$ time and $O(kn)$ space.*

1 Select a vertex r of G to be the root of G.
2 **for** all $u \in H, v \in G$ **do** $S(v,u) \leftarrow \phi$.
3 **for** all leaves v of G^r **do**
4   **for** all leaves u of H **do** $S(v,u) \leftarrow N(u)$.
5 **for** all internal vertices v of G^r in a postorder **do**
6   Let $v_1, v_2, \ldots, v_t$ be the children of v.
7   **for** all vertices $u = u_0$ of H with degree at most $t+1$ **do**
8   Let $u_1, u_2, \ldots, u_s$ be the neighbors of u.
9   Construct a bipartite graph $B(v,u) = (X, Y, E_{vu})$,
            where $X = \{u_1, \ldots, u_s\}$, $Y = \{v_1, \ldots, v_t\}$,
            and $E_{vu} = \{u_i v_j : u \in S(v_j, u_i)\}$.
            Denote $X_0 = X$ and $X_i = X - \{u_i\}$.
10   **for** all $0 \leq i \leq s$ **do** compute the size m_i of a maximum
            matching between X_i and Y.
11        $S(v,u) \leftarrow \{u_i : m_i = |X_i|, 0 \leq i \leq s\}$.
12   **if** $u \in S(v,u)$ **then** answer YES and **stop**
13   **end for**
14 **end for**
15 Answer NO.

Figure 3: Algorithm Subtree-Isomorphism(G, H)

Proof: The correctness of the algorithm follows from Lemma 2.1. Let us consider the time complexity. Let $V = \{x_1, x_2, \ldots, x_n\}$, and $V_H = \{y_1, y_2, \ldots, y_k\}$. As each set $S(v,u)$ is a subset of $N[u]$, we can maintain $S(v,u)$ using a binary vector $A(v,u)$ of size $|N[u]|$. Also, we maintain a $k \times k$ matrix I, where $I(u,w)$ is the index of $w \in N[u]$ in the vector $A(v,u)$. In other words, $w \in S(v,u)$ iff bit number $I(u,w)$ in $A(v,u)$ is set. Thus the space complexity is $O(k^2 + \sum_{j=1}^{k} \mathrm{d}(y_j) n) = O(kn)$ and each access to $S(v,u)$ takes constant time.

As for the algorithm's time complexity, the crux is step 10. It computes the size of several maximum matchings in some graphs. Since these graphs are very related to each other, we are able to show in Section 3 that computing the sizes of their maximum matchings can be done in the same time bound taken by computing a single maximum matching. We therefore perform step 10 of the algorithm using Corollary 3.2, and therefore the dominant part of the algorithm's time complexity is finding maximum matchings in the graphs $B(x_i, y_j)$ for all i, j in step 10. Finding a maximum matching in $B(x_i, y_j)$ takes $O(\mathrm{d}(y_j)^{1.5}\mathrm{d}(x_i))$ time using the algorithm of Hopcroft and Karp [12] (or the equivalent algorithm of Dinic [7]) and therefore the time needed to handle a vertex x_i is at most $O(\sum_{j=1}^{k} \mathrm{d}(y_j)^{1.5}\mathrm{d}(x_i)) = O((2k)^{1.5}\mathrm{d}(x_i))$. Thus, the total time complexity is $O(k^{1.5}n)$. ■

We note that the above algorithm can be changed to solve the *subtree homeomorphism* problem without changing the asymptotic complexity. The same modification applies to the algorithms in the sections below.

3 Matching

In this section, we give several lemmas about matchings which are needed for obtaining the more efficient algorithms for the subtree isomorphism problem. For the following lemmas, let $B = (X, Y, E)$ be a bipartite graph, where $X = \{x_1, x_2, \ldots, x_s\}, Y = \{y_1, y_2, \ldots, y_t\}$ and $s \leq t + 1$. Denote $X_0 = X$ and $X_i = X - \{x_i\}$ for $1 \leq i \leq s$. For $0 \leq i \leq s$, denote by m_i the size of a maximum matching between X_i and Y. Clearly for every $i \geq 1$, either $m_i = m_0$ or $m_i = m_0 - 1$.

An important notion in matching theory is critical vertices (see e.g. [18]): A vertex x in a graph G is *critical* if the size of a maximum matching in $G - x$ is strictly less than the size of a maximum matching in G (i.e., x_i is critical iff $m_i = m_0 - 1$). Let M be a maximum matching of B. We define the directed graph B_M by $B_M = (X \cup Y, E_M)$ where

$$E_M = \{(x, y) : xy \in E - M, x \in X, y \in Y\} \cup \{(y, x) : xy \in M, x \in X, y \in Y\}.$$

We denote by X_M all the vertices from X which are unmatched in M.

Lemma 3.1 *For any maximum matching M of B and any vertex $x_i \in X$, x_i is critical (i.e. $m_i = m_0 - 1$) if and only if x_i is matched in M, and there is no directed path in B_M from a vertex in X_M to x_i.*

Proof: ($\rightarrow$) The proof is by contradiction. If x_i is unmatched in M, then M is a matching between X_i and Y and therefore $m_i = m_0$, a contradiction. Also, if there is a path P in B_M from a vertex in X_M to x_i, then $M \triangle P$ $(= M \cup P - M \cap P)$ is a maximum matching in which x_i is unmatched, and again we have a contradiction as $m_i = m_0$.

($\leftarrow$) Conversely, assume by contradiction that $m_i = m_0$. Let y be the vertex matched to x_i in M. As $|M - \{x_i y\}| = m_0 - 1 < m_i$, $M - \{x_i y\}$ is not a maximum matching between X_i and Y, and by Berge's theorem (see [18]) there is an augmenting path P whose one end is a vertex $x_j \in X_M$, and the other end is y (because if the other end is a vertex in Y that is unmatched in M then P is an augmenting path for M). But this implies a directed path in B_M from x_j to x_i, a contradiction. ■

Corollary 3.2 *Given a maximum matching M of B, we can compute the value of m_i for $0 \leq i \leq s$ in $O(st)$ time.*

Proof: Building the graph B_M and finding all the vertices reachable from X_M can be done in $O(st)$ time using depth first search [25]. We then apply Lemma 3.1. For each vertex x_i, if x_i is matched in M and is not reachable from X_M we set $m_i = |M| - 1$ and otherwise $m_i = |M|$. ■

Lemma 3.3 *The problem of finding a maximum matching in B can be reduced in $O(st)$ time to the problem of finding a maximum matching in a subgraph of B with at most s^2 vertices and edges and with maximum degree at most s.*

Proof: Let X' be the set of all vertices of X with degree less than s. Let B' the subgraph induced from B by the vertices of X' and their neighbors. Building X' and B' takes $O(|X| + |Y| + |E|) = O(st)$ time. We claim that given a maximum matching M' of B' we can build a maximum matching M of B: First, add all the edges of M' to M. For each vertex $x \in X - X'$ find an unmatched neighbor $y \in Y$ and add xy to M (such a vertex y must exist since each such x has at least s neighbors and at most $s-1$ of them are matched). Finding an unmatched neighbor of x takes $O(\mathrm{d}_{B'}(x)) = O(s)$ time, and therefore building M takes $O(s^2)$ time. ■

4 An $O(\frac{k^{1.5}}{\log k}n)$ algorithm

We now improve the algorithm from Section 2 by a $\log k$ factor. We use the previous algorithm but we solve the maximum matching problems more efficiently using the idea of clique partition of a bipartite graph and its usage in finding maximum matching [9]. The algorithm of Feder and Motwani, originally stated for bipartite graphs with equal size parts, can be extended to general bipartite graphs. This allows to give an algorithm for subtree isomorphism whose time complexity is $O(\frac{k^{1.5}}{\log k}n)$. Instead of describing the modifications to the algorithm of Feder and Motwani, we will give here a simpler way for obtaining an $O(\frac{k^{1.5}}{\log k}n)$-time algorithm for subtree isomorphism. In contrast with [9] where the denseness of the graph is exploited, we achieve the reduction in time complexity by utilizing the special structure of the matching problems that must be solved in the subtree isomorphism algorithm.

The modified algorithm, called Improved-Subtree-Isomorphism, is the same as the algorithm Subtree-Isomorphism with the exception that, in step 10, we solve the maximum matching problems differently. Let v be some vertex in G^r whose children are $v_1, v_2, \ldots, v_t$, and let u be a vertex in H whose neighbors are $u_1, u_2, \ldots, u_s$. We now consider finding a maximum matching in $B = B(v, u)$. Recall that $B = (X, Y, E)$ with $X = \{u_1, \ldots, u_s\}$ and $Y = \{v_1, \ldots, v_t\}$. We assume that $s \leq t + 1$ (because otherwise $S(v, u) = \phi$).

We first apply Lemma 3.3 and build a subgraph $B' = (X', Y', E')$ of B having maximum degree at most s. Then, like in [9], we will partition the edges of B' into complete bipartite graphs $C_1, C_2, \ldots, C_p$. We do the partition in the following way: First, we sort the vertices of X' in lexicographic order where the key of a vertex u is $N(u)$. Afterwards, we split X' into sets of equal keys $X^1, X^2, \ldots, X^p$ (i.e., all the vertices in a set X^i have the same neighbors in Y'). Now, for $1 \leq i \leq p$ we set C_i to be the subgraph induced by the vertices of X^i and all their neighbors in Y'. We now follow the method of [9] and build a network B^* whose vertices are $V^* = X' \cup Y' \cup \{c_1, \ldots, c_p, a, b\}$. The edges are $E^* = E_1 \cup E_2$ where

$$E_1 = \{au_i : u_i \in X'\} \cup \{v_i b : v_i \in Y'\}$$

$$E_2 = \{u_i c_j : j \leq p, u_i \in C_j\} \cup \{c_j v_i : j \leq p, v_i \in C_j\}$$

All edges have capacity 1. The source is a and the sink is b. We find a maximum (integral) flow f in B^* using Dinic's algorithm [7] (see also [8]), and construct from this flow a maximum matching in B'. (Since the capacity of all edges is one, the flow f can be decomposed into edge-disjoint paths from a to b where the flow along each path is one. Since each such path is of the form $a - u_i - c_k - v_j - b$, we can define a matching in B' by taking the edge $u_i v_j$ for each such path. The maximality of this matching follows from the maximality of the flow).

We will now analyze the time complexity of the algorithm described above. We denote by $D(u)$ the number of distinct trees in the forest $H^u_{u_1}, \ldots, H^u_{u_s}$.

Lemma 4.1 *Algorithm Improved-Subtree-Isomorphism finds a maximum matching in $B(v,u)$ in $O(st + ts^{0.5}D(u))$ time.*

Proof: Note that if for some i, j the rooted trees $H^u_{u_i}$ and $H^u_{u_j}$ are isomorphic, then in $B = B(v,u)$ the vertices u_i, u_j have exactly the same neighbors, and this remains true in B' (assuming u_i and u_j were not deleted in B'). Therefore $p \leq D(u)$.

The time for constructing B, B' and B^* is $O(st)$ (we sort the vertices of X' using radix-sort). We now bound the time for finding a maximum flow in B^*: The size of E_1 is $|X'| + |Y'|$ where $|X'| \leq s$ and $|Y'| \leq sp$ (since each vertex in Y' has at least one edge arriving from some vertex c_j, and no more than s edges depart from each vertex c_j to the vertices in Y'). The size of E_2 is at most $2sp$ as the number of edges in E_2 incident on some vertex c_j is at most $|X'| + \mathrm{d}_{B'}(u_i) \leq 2s$ where $u_i \in C_j$. Hence, the number of edges in B^* is $O(sp)$. The number of vertices in B^* is at most $s+t+p+2 = O(t)$ (as $s \leq t+1$). Now, Dinic's algorithm performs $O(\sqrt{|V^*|})$ stages (see [9]), and each stage takes $O(|E^*|)$ time. Hence, the total time is $O(t^{0.5}sp) = O(ts^{0.5}p) = O(ts^{0.5}D(u))$. ■

We denote again $V = \{x_1, x_2, \ldots, x_n\}$ and $V_H = \{y_1, y_2, \ldots, y_k\}$. By Lemma 4.1, the time complexity of algorithm Improved-Subtree-Isomorphism is

$$O(\sum_i \sum_j (\mathrm{d}(y_j)\mathrm{d}(x_i) + \mathrm{d}(x_i)\mathrm{d}(y_j)^{0.5}D(y_j))) = O(kn + n\sum_j \mathrm{d}(y_j)^{0.5}D(y_j)).$$

We will bound the summation $\sum_j \mathrm{d}(y_j)^{0.5}D(y_j)$ by $O(k^{1.5}/\log k)$.

We first need a simple combinatorial Lemma. Let $g(n)$ denote the number of distinct (i.e., non-isomorphic) rooted trees with n vertices. We shall use the following result:

Lemma 4.2 *(see e.g. [22, p. 1197])* $g(n) = 2^{\Theta(n)}$.

Let $f(n)$ denote the maximum number of distinct rooted trees in a forest of n vertices.

Lemma 4.3 $f(n) = \Theta(\frac{n}{\log n})$.

Proof: To show the lower bound, we use the fact that $g(n) \geq 2^n$ for large n. Hence, the number of distinct rooted trees with $l = \lfloor \log \frac{n}{\log n} \rfloor$ vertices is $g(l) \geq 2^{\log \frac{n}{\log n} - 1} = \frac{n}{2\log n}$. Therefore we can build a forest by taking $\lfloor \frac{n}{2\log n} \rfloor$ distinct trees with l vertices each, and the total number of vertices in this forest is $\lfloor \frac{n}{2\log n} \rfloor l < n$. Thus $f(n) = \Omega(\frac{n}{\log n})$. We will now show the upper bound.

If we have a forest of rooted trees and r_i is the number of trees with i vertices, then the number of distinct trees in this forest is at most $\sum_i \min(r_i, g(i))$. Hence,

$$f(n) \leq \max\{\sum_i \min(r_i, g(i)) : r_1, \ldots, r_n \in N, \sum_{i=1}^{n} i r_i \leq n\}$$

By Lemma 4.2,

$$f(n) \le \max\{\sum_i \min(r_i, c^i) : r_1, \ldots, r_n \in N, \sum_{i=1}^{n} i r_i \le n\}$$

for some integer constant c. Let x be the minimum integer for which $\sum_{i=1}^{x} i c^i \ge n$. Let $r_1, \ldots, r_n$ be the integers that maximize $\sum_i \min(r_i, c^i)$ under the constraint $\sum_{i=1}^{n} i r_i \le n$. We can assume that $r_i \le c^i$ for all i, because if $r_j > c^j$ for some j, we can set $r_j = c^j$ and the value of $\sum_i \min(r_i, c^i)$ does not change. Now, suppose $r_j > 0$ for some $j > x$. This implies that there is a $k \le x$ for which $r_k < c^k$ (because otherwise $\sum_{i=1}^{n} i r_i \ge \sum_{i=1}^{x} i r_i + j r_j = \sum_{i=1}^{x} i c^i + j r_j > n$, a contradiction). If we decrease r_j by one, and increase r_k by one, then the value of $\sum_i \min(r_i, c^i)$ does not change, and the constraint $\sum_{i=1}^{n} i r_i \le n$ still holds (as $k < j$). We can repeat this process until $r_j = 0$ for all $j > x$ and therefore

$$f(n) \le \sum_{i=1}^{n} \min(r_i, c^i) \le \sum_{i=1}^{x} c^i = O(c^x).$$

The lemma follows from the fact that $x = \log_c n - \log_c \log_c n - O(1)$. ■

We now continue with the analysis of algorithm Improved-Subtree-Isomorphism. Let ϵ be some constant $0 < \epsilon < 1/3$. We call a vertex of H *heavy* if its degree is at least $2k^{1-\epsilon}$, and otherwise it is called *light*. Clearly, the number of heavy vertices is at most k^{ϵ}. If u is a heavy vertex and v is a neighbor of u such that H_v^u does not contain a heavy vertex, then we call every vertex in H_v^u a *private vertex* of u. We denote by l_j the number of the private vertices of a heavy vertex y_j.

Lemma 4.4 *For any heavy vertex y_j, $\mathrm{d}(y_j) \le k^{\epsilon} + l_j$ and $D(y_j) \le k^{\epsilon} + f(l_j)$.*

Proof: Each neighbor of y_j is either a private vertex of y_j or not. The number of private neighbors of y_j is at most the total number of private vertices of y_j which is l_j. For each non-private neighbor v of u, we can choose a heavy vertex v' in H_u^v. As all vertices we chose are distinct, we have that the number of non-private neighbors of u is less than the number of heavy vertices. Hence, $\mathrm{d}(y_j) \leq k^\epsilon + l_j$.

Let $u_1, \ldots, u_s$ be the neighbors of u. Among the trees $H^u_{u_1}, \ldots, H^u_{u_s}$ at most k^ϵ contain a heavy vertex. The rest of the trees constitute all the private vertices of u, and therefore they have a total of l_j vertices and there are at most $f(l_j)$ distinct trees among them. Hence, $D(y_j) \leq k^\epsilon + f(l_j)$. ■

Lemma 4.5 $\sum_j \mathrm{d}(y_j)^{0.5} D(y_j) = O(\frac{k^{1.5}}{\log k})$.

Proof:

We split $\sum_{j=1}^k \mathrm{d}(y_j)^{0.5} D(y_j)$ into two sums. Summing over the light vertices of H we have

$$\begin{aligned}\sum_{j:y_j \text{ is light}} \mathrm{d}(y_j)^{0.5} D(y_j) &\leq \sum_{j:y_j \text{ is light}} \mathrm{d}(y_j)^{1.5} \leq (2k^{1-\epsilon})^{0.5} \sum_{j:y_j \text{ is light}} \mathrm{d}(y_j) \leq \\ &\leq (2k^{1-\epsilon})^{0.5} 2k = 2^{1.5} k^{1.5-\epsilon/2}\end{aligned}$$

where the last inequality follows from the fact that $\sum_{j=1}^k d(y_j) = 2k-2$.

Summing over the heavy vertices and using Lemma 4.4 we have

$$\sum_{j:y_j \text{ is heavy}} \mathrm{d}(y_j)^{0.5} D(y_j) \leq \sum_{j:y_j \text{ is heavy}} (k^{\epsilon/2} + l_j^{0.5})(k^\epsilon + f(l_j))$$

Since $f(l_j) \leq l_j \leq k$ and since the number of heavy vertices is at most k^ϵ, we have

$$\begin{aligned}&\leq \sum_{j:y_j \text{ is heavy}} (k^{3\epsilon/2} + k^{0.5+\epsilon} + k^{1+\epsilon/2} + l_j^{0.5} f(l_j)) \\ &\leq 3k^{1+3\epsilon/2} + \sum_{j:y_j \text{ is heavy}} l_j^{0.5} f(l_j)\end{aligned}$$

and by Lemma 4.3, the fact that $\sum_{j:y_j \text{ is heavy}} l_j \leq k$ (as each vertex can be a private vertex of at most one heavy vertex), and the fact that the function $h(x) = x^{1.5}/\log x$ is convex,

$$\leq \ 3k^{1+3\epsilon/2} + \sum_{j:y_j \text{ is heavy}} c\frac{l_j^{1.5}}{\log l_j} \leq 3k^{1+3\epsilon/2} + c\frac{k^{1.5}}{\log k}$$

■

We therefore proved the following theorem:

Theorem 4.6 *Algorithm Improved-Subtree-Isomorphism solves the subtree isomorphism problem in* $O(\frac{k^{1.5}}{\log k}n)$ *time.*

5 An $O(k^{1.376}n)$ algorithm

In this section we give a randomized algorithm for the decision problem of subtree isomorphism. The algorithm is more efficient asymptotically than the deterministic algorithm of the previous section.

Again, the algorithm Randomized-Subtree-Isomorphism is based on the algorithm Subtree-Isomorphism but solving the maximum matching problems is done differently. Consider some vertex v in G^r whose children are $v_1, v_2, \ldots, v_t$, and some vertex u in H whose neighbors are $u_1, u_2, \ldots, u_s$. We use the algorithm of Cheriyan [3] to find the size of a maximum matching in $B(v, u)$ and to find all the critical vertices in $B(v, u)$.

Cheriyan's algorithm for finding critical vertices (in a general graph) is as follows: Given an input graph $G = (V, E)$ where $V = \{1, 2, \ldots, n\}$, choose a large prime number q and build an $n \times n$ matrix C in the following way: For each edge $ij \in E$, choose a random number w_{ij} uniformly from $\{1, 2, \ldots, q-1\}$, and set $c_{ij} = w_{ij}$ and $c_{ji} = -w_{ij}$. Set all the other elements of C to zero. Finally, find a basis $a_1, a_2, \ldots a_r$ for the null space of C over the field Z_q. Then, with high probability, the size of a maximum matching in

G is equal to $(n-r)/2$ and for all $i \in V$, vertex i is critical iff all the i-th coordinates of $a_1, a_2, \ldots a_r$ are zeros.

As the graph $B(v,u)$ is bipartite, when applying Cheriyan's algorithm on $B(v,u)$, the matrix C is of the form

$$C = \begin{pmatrix} 0 & C' \\ C'' & 0 \end{pmatrix}$$

where C' is an $t \times s$ block and C'' is an $s \times t$ block. Therefore, we can find a basis for the null space of C by finding bases for the null spaces of C' and C''. This can be done in $O(s^{\omega-1}t)$ time [14] where ω denotes the exponent of matrix multiplication. Thus, the running time of algorithm Randomized-Subtree-Isomorphism is $O(\sum_{i=1}^{n} \sum_{j=1}^{k} \mathrm{d}(y_j)^{\omega-1}\mathrm{d}(x_i)) = O(k^{\omega-1}n)$.

Theorem 5.1 *Algorithm Randomized-Subtree-Isomorphism solves the decision problem in $O(k^{\omega-1}n)$ expected time.*

Since $\omega < 2.376$ [5], we have

Corollary 5.2 *Algorithm Randomized-Subtree-Isomorphism solves the decision problem in $O(k^{1.376}n)$ expected time.*

6 Acknowledgments

We are grateful to the anonymous referee for careful reading of the manuscript and many helpful comments.

References

[1] S. Anderson. Graphical representation of molecules and substructure-search queries in MACCS. *J. of Molecular Graphics*, 2:8–90, 1984.

[2] M. Behzad, G. Chartrand, and L. Lesniak-Foster. *Graphs & Digraphs*. Wadsworth International Group, 1979.

[3] J. Cheriyan. Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory. *SIAM J. Computing*, 26:1635–1655, 1997.

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

[5] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comp.*, 9:23–52, 1990.

[6] A. Dessmark, A. Lingas, and A. Proskurowski. Faster algorithms for subgraph isomorphism of k-connected partial k-trees. In *Proc. Fourth European Symposium on Algorithms, (ESA 96)*, pages 501–513. Springer-Verlag Lecture Notes in Computer Science, 1996.

[7] E. A. Dinic. An algorithm for solution of a problem of maximum flow in a network with power estimation. *Soviet Math. Doklady*, 11:1277–1280, 1970.

[8] S. Even. *Graph Algorithms*. Computer Science Press, Rockville, Maryland, 1979.

[9] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. In *Proc. 23rd Symposium on the Theory of Computing (STOC 91)*, pages 123–133. ACM press, 1991.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Francisco, 1979.

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

[12] J. E. Hopcroft and R. M. Karp. A $n^{5/2}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2:225–231, 1973.

[13] J. E. Hopcroft and R. E. Tarjan. Isomorphism of planar graphs. In *Raymond E. Miller and James W. Thatcher, Complexity of Computer Computations, Plenum Press*, pages 131–152. 1972.

[14] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *J. Algorithms*, 3(45–56), 1982.

[15] M. Karpinski and A. Lingas. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.

[16] A. Lingas. Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science*, 63:295–302, 1989.

[17] A. Lingas and M. M. Sysło. A polynomial-time algorithm for subgraph isomorphism of two-connected series-parallel graphs. In *Proc. 15th Int. Colloq. Automata, Languages and Programming*, volume 317, pages 394–409. Springer-Verlag Lecture Notes in Computer Science, 1988.

[18] L. Lovasz and M. D. Plummer. *Matching Theory.* North-Holland, Amsterdam, 1986.

[19] J. Matoušek and R. Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Math.*, 108:343–364, 1992.

[20] D. W. Matula. An algorithm for subtree identification. *SIAM Rev.*, 10:273–274, 1968.

[21] D. W. Matula. Subtree isomorphism in $O(n^{5/2})$. *Ann. Discrete Math.*, 2:91–106, 1978.

[22] A. M. Odlyzko. Asymptotic enumeration methods. In *Handbook of Combinatorics, edited by R. L. Graham, M. Grotschel, and L. Lovasz*, volume 2, pages 1063–1229. Elsevier and the MIT press, 1995.

[23] R. Shamir and D. Tsur. The maximum subforest problem: Approximation and exact algorithms. In *Proc. Ninth Symposium on Discrete Algorithms (SODA 98)*, pages 394–399. ACM press, 1998.

[24] R.E. Stobaugh. Chemical substructure searching. *J. of Chemical Information and Computer Sciences*, 25:271–275, 1985.

[25] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1:146–160, 1972.