Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 507–518

DOI: 10.15439/2015F329
ACSIS, Vol. 5

Maximum Exploratory Equivalence in Trees

Luka Fürst, Uroš Čibej, and Jurij Mihelič
University of Ljubljana, Faculty of Computer and Information Science
Večna pot 113, SI-1000 Ljubljana, Slovenia
Email: {luka.fuerst,uros.cibej,jurij.mihelic}@fri.uni-lj.si

***Abstract*—Many practical problems are modeled with networks and graphs. Their exploration is of significant importance, and several graph-exploration algorithms already exist. In this paper, we focus on a type of vertex equivalence, called *exploratory equivalence*, which has a great potential to speed up such algorithms. It is an equivalence based on graph automorphisms and can, for example, help us in solving the subgraph isomorphism problem, which is a well-known NP-hard problem. In particular, if a given pattern graph has nontrivial automorphisms, then each of its nontrivial exploratory equivalent classes gives rise to a set of constraints to prune the search space of solutions. In the paper, we define the maximum exploratory equivalence problem. We show that the defined problem is at least as hard the graph isomorphism problem. Additionally, we present a polynomial-time algorithm for solving the problem when the input is restricted to tree graphs. Furthermore, we show that for trees, a maximum exploratory equivalent partition leads to a globally optimal set of subgraph isomorphism constraints, whereas this is not necessarily the case for general graphs.**

I. Introduction

Searching for patterns in structured data is one of the most ubiquitous applications of computer algorithms in various scientific areas. Such data is often modeled with graphs, which efficiently represent diverse types of entities (modeled by graph vertices) and relations between them (modeled by graph edges) and also enable a more general and global view on the data. In the era of ever-growing, even planetary-wide (social, citation, traffic, etc.) networks [1], all of which can be naturally modeled by graphs, graph representation and also algorithms on graphs are becoming increasingly important. Applications of graphs arise in various areas, ranging from chemistry [2], [3], economy [4], politics [5], to popular culture [6].

In this paper, we focus on a general technique for speeding up algorithms that search for patterns in graphs. The main idea of our technique is to exploit symmetries of a graph, i.e., to find equivalent vertices in such a way that if two vertices are equivalent then the search algorithm could process only one (and deduce the information about the other one).

The problem of finding equivalent vertices of a graph has already appeared in the literature; see, for example, papers on regular and structural equivalence [7], [8]. To the best of our knowledge, our definition introduces a new form of equivalence. We call it *exploratory equivalence* (EE), since its primary intent is to be utilized in graph search algorithms; see [9] for the introductory paper. Nevertheless, the exploitation of symmetries of a problem to reduce the amount of time for exploring the solution search space is not new. See, for example, a method for solving 0/1 integer linear programs having a large symmetry [10] or [11] for a similar method. Another equivalence similar to ours, defined in [12], can also be used for pruning the search space. However, our equivalence is more general and, hence, has a greater pruning power.

To find the symmetries in a graph, the usual approach is to find all graph isomorphisms (GI), i.e., structure preserving mappings. In particular, given two graphs, the graph isomorphism problem asks whether they are the same. Similarly, the graph automorphism problem asks whether a graph can be (non-trivially) mapped to itself. The GI problem has a special place in the complexity theory, as it is a canonical example of a possible candidate for an NP-intermediate problem. Ladner's theorem [13] tells us that if P is not equal to NP, then the class of NP-intermediate problems is not empty. As a result, a polynomial-time algorithm is unlikely for the GI problem. Nevertheless, in practice, there are several efficient algorithms and software packages for finding automorphisms of graphs, e.g., NAUTY [14], [15], BLISS [16], [17], SAUCY [18], [19], [20], and NISHE [21]. For some special cases of graphs, e.g., tree graphs [22], polynomial-time algorithms exist.

Based on automorphisms, one could define *automorphic equivalence*, where two vertices are equivalent if and only if there exists an automorphism that maps one to another. Such equivalence classes are also called *orbits*. Notice that exploratory equivalence is similar but not the same as *automorphic equivalence*. Indeed, exploratory equivalence is more restrictive.

In our preceding paper [9], we already presented the definition of exploratory equivalences and the corresponding problem of finding maximum exploratory equivalent partition of graph vertices, i.e., the MAXEXPLOREQ problem. In this paper, we show that the MAXEXPLOREQ is GI-hard, which means that a polynomial-time algorithm (in terms of the number of graph vertices) is unlikely to exist. Hence, it is reasonable to restrict the input to MAXEXPLOREQ to selected subclasses of graphs. As the second contribution of this paper, we present a polynomial-time algorithm for solving MAXEXPLOREQ on an arbitrary tree. Thereby we show that the restriction of MAXEXPLOREQ to trees is in P. Additionally, we also show that for trees, a maximum exploratory equivalent partition leads to a globally optimal set of subgraph isomorphism search constraints. In particular, when searching for a given tree in a given host graph using the constraints derived from a maximum exploratory equivalent partition of the tree, each of the occurrences of the tree in the host graph will be discovered exactly once. For general graphs, this is not necessarily the case.

The rest of this paper is structured as follows. In the next section, we present mathematical notions needed for the rest of the paper. In Section III, we present a motivational example (based on the subgraph isomorphism problem) for exploratory

978-83-60810-66-8/$25.00©2015, IEEE

equivalence. The definition of the MAXEXPLOREQ problem is presented in Section IV. In Section V, we show that the MAXEXPLOREQ problem is GI-hard. In Section VI, we present a polynomial-time algorithm for solving the MAXEXPLOREQ problem on trees and determine its computational complexity. Section VII presents an empirical demonstration of the presented algorithm on the set of all small trees. In Section VIII, we elaborate on the connection between exploratory equivalence and subgraph isomorphism, with a particular emphasis on trees. Finally, Section IX concludes the paper.

II. PRELIMINARIES

Let $G = (V, E)$ denote a simple undirected graph, where $V = \{1, 2, \ldots, n\}$ is a set of vertices and $E \subseteq V \times V$ is a set of edges. The graph can be labeled; let Σ denote a set of labels, and let $\ell_V \colon V \to \Sigma$ and $\ell_E \colon E \to \Sigma$ denote the functions that assign labels to individual vertices and edges, respectively. An unlabeled graph can be viewed as a labeled graph where all vertices and edges have the same label. A *tree* is a connected acyclic undirected graph.

A (graph) *homomorphism* from a graph $G = (V, E)$ to a graph $H = (U, F)$ is a mapping $h \colon V \to U$ such that for each $(i, j) \in E$ it also holds that $(h(i), h(j)) \in F$. To simplify notation, the homomorphism $h \colon V \to U$ will be denoted $h \colon G \to H$. An *endomorphism* is a homomorphism whose domain is equal to its codomain, i.e., $h \colon G \to G$.

An *isomorphism* is a bijective homomorphism, i.e., a mapping $h \colon G \to H$ such that $(i, j) \in E$ if and only if $(h(i), h(j)) \in F$. We write $G \simeq H$ if there exists an isomorphism from G to H; such graphs G and H are called isomorphic. A *subgraph isomorphism* $G \to H$ is an isomorphism between the graph G and a subgraph of the graph H. A subgraph in H that is isomorphic to G is called an *occurrence* of G in H.

An *automorphism* is both an endomorphism and an isomorphism, i.e., a mapping $h \colon G \to G$ such that $(i, j) \in E$ if and only if $(h(i), h(j)) \in E$. Note that every automorphism is a permutation. The set of all automorphisms of a graph G can be defined as

$$\mathrm{Aut}(G) = \{a \in \Pi[n] \mid G \simeq a(G)\} \tag{1}$$

where $\Pi[P]$ denotes the set of all permutations of a set P and $\Pi[n] \equiv \Pi[\{1, 2, \ldots, n\}]$. For example, the set of automorphisms of the graph G in Fig. 1 can be denoted as {123456, 123465, 124356, 124365, 215634, 215643, 216534, 216543}, where, e.g., 215643 denotes an automorphism h such that $h(1) = 2$, $h(2) = 1$, $h(3) = 5$, $h(4) = 6$, $h(5) = 4$, and $h(6) = 3$.

Given a (finite) set S, a family $\{P_1, P_2, \ldots, P_s\}$ of nonempty subsets of S is a *partition* of S if every element in S is exactly in one of the subsets, i.e., $P_i \subseteq S$ and $P_i \neq \emptyset$, where $1 \leq i \leq s$, $\bigcup_{1 \leq i \leq s} P_i = S$, and $P_i \cap P_j = \emptyset$ for all $1 \leq i, j \leq s$ with $i \neq j$. When the partition $\{P_1, P_2, \ldots, P_s\}$ is given explicitly, we usually use $\{i \in P_1 \mid i \in P_2 \mid \ldots \mid i \in P_s\}$ as a short form, e.g., $\{\{1, 2\}, \{3\}, \{4\}\}$ is shortened to $\{1, 2 \mid 3 \mid 4\}$. In what follows, the order of the sets in a partition is often important. To denote such an *ordered partition*, we use the form $\langle i \in P_1 \mid i \in P_2 \mid \ldots \mid i \in P_s \rangle$, e.g., $\langle 1, 2 \mid 3 \mid 4 \rangle$.

III. MOTIVATION

Given a pattern graph and a host graph, the goal of the *subgraph isomorphism problem* is to find all (or at least one, depending on the definition) occurrences of the pattern graph in the host graph, i.e., the subgraphs of the host graph that are isomorphic to the pattern graph.

Unfortunately, the decision version of the subgraph isomorphism problem is NP-complete [23], while its counting version is $\#P$-complete, since the counting version of the clique problem is $\#P$-complete [24]. Furthermore, not only that it is unlikely that a polynomial-time algorithm exists, but so far no exponential-time algorithm with a lower bound better than what can be achieved by the naive enumeration of the occurrences has been devised [25]. Most algorithms are therefore based on a backtracking approach (e.g., [26], [27]). In particular, the vertices of the pattern graph are matched with those of the host graph until a match is found, using the vertex neighborhood information to prune the search space.

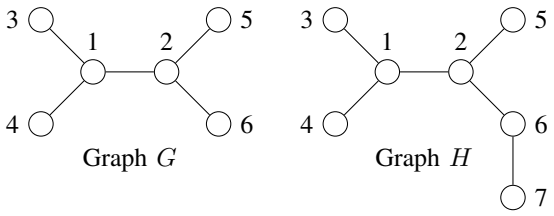

Fig. 1. A sample pattern graph G and host graph H.

Let us assume that a given pattern graph G has m nontrivial automorphisms. When searching for the occurrences of G in a given host graph H, a search algorithm that establishes all valid matches between G and subgraphs of H discovers *each* of G's occurrences m times, because the vertices of G can be isomorphically mapped to the vertices of each of G's occurrences in m different ways. As an example, consider the pattern graph G and the host graph H in Fig. 1. An algorithm that is unaware of the eight automorphisms of G will find the single occurrence of G in H eight times. In other words, it will establish eight subgraph isomorphisms $h \colon G \to H$:

i	$h_i(1)$	$h_i(2)$	$h_i(3)$	$h_i(4)$	$h_i(5)$	$h_i(6)$
1	1	2	3	4	5	6
2	1	2	3	4	6	5
3	1	2	4	3	5	6
4	1	2	4	3	6	5
5	2	1	5	6	3	4
6	2	1	5	6	4	3
7	2	1	6	5	3	4
8	2	1	6	5	4	3

However, by imposing the constraints $h(1) < h(2)$, $h(3) < h(4)$, and $h(5) < h(6)$ while performing the exhaustive subgraph isomorphism search, the sole occurrence of the graph G in the graph H will be discovered exactly once, and this *regardless of the numbering of H's vertices*.

Motivated by this observation, we recently introduced the so-called *exploratory equivalence* [9], on the basis of which such constraints can be defined and safely imposed during

the subgraph isomorphism search. Exploratory equivalence is an automorphisms-based equivalence relation on the vertices of a given graph; an *exploratory equivalent partition* (EE partition) is a partition of the graph vertex set into a set of exploratory equivalence classes. Every EE partition can be directly translated into a safe set of search constraints, where 'safe' means that the constraints never lead the algorithm to miss any occurrence, regardless of the numbering of the host graph vertices. In particular, an EE partition $\{u_{11}, \ldots, u_{1k_1} \mid u_{21}, \ldots, u_{2k_2} \mid \ldots \mid u_{s1}, \ldots, u_{sk_s}\}$ (the vertices u_{11}, …, u_{1k_1} constitute the first equivalence class, etc.) gives rise to the constraint set $\{h(u_{11}) < \ldots < h(u_{1k_1}), h(u_{21}) < \ldots < h(u_{2k_2}), \ldots, h(u_{s1}) < \ldots < h(u_{sk_s})\}$. In the example of Fig. 1, one of the EE partitions of the graph G is $\{1, 2 \mid 3, 4 \mid 5, 6\}$, and it leads to the above-mentioned set of constraints.

If a graph G has nontrivial automorphisms, it has several nontrivial EE partitions. All of them lead to a safe set of search constraints. However, of particular interest is one that gives rise to the set of constraints that results in the largest speedup when searching for the occurrences of G. Such an EE partition is called a *maximum EE partition* ('a' instead of 'the' because there can be several of them), and the problem of finding such a partition for a given graph is denoted MAXEXPLOREQ. In our previous paper [9], we defined the problem and showed two algorithms, both of which are polynomial only in the number of automorphisms, rather than in the number of graph vertices. Besides that, the algorithms fail to find a maximum EE partition for all graphs, although counterexamples appear to be very rare; for example, the second algorithm finds a maximum EE partition for all but 2 graphs out of 261080 connected unlabeled undirected 9-vertex graphs.

IV. PROBLEM DESCRIPTION

Since the MAXEXPLOREQ problem is defined and thoroughly explained in our FedCSIS 2014 paper [9], we provide a relatively brief review of the main definitions.

Definition 1 (cover): A set of permutations $A \subseteq \Pi[n]$ *covers* a set $P \subseteq \{1, \ldots, n\}$ if for every permutation σ of the set P there exists a permutation $a \in A$ such that $a(i) = \sigma(i)$ for all $i \in P$:

$$\text{cover}(A, P) \equiv \forall \sigma \in \Pi[P]\ \exists a \in A\ \forall i \in P\colon \sigma(i) = a(i). \quad (2)$$

For example, the set Aut(G) for the graph G of Fig. 1 covers the set $\{3, 5\}$, since it contains both an automorphism for which $a(3) = 3$ and $a(5) = 5$ (12**3**4**5**6) and an automorphism for which $a(3) = 5$ and $a(5) = 3$ (21**5**6**3**4). For the graph G' of Fig. 2, the set Aut(G') covers the set $\{1, 3, 5\}$, since it contains an automorphism for each of the 3! permutations of the set $\{1, 3, 5\}$ (**1**2**3**4**5**6 for the permutation 135, **1**6**5**4**3**2 for the permutation 153, **3**2**1**6**5**4 for the permutation 315, etc.)

Definition 2 (stabilizer): The *stabilizer* of a set $A \subseteq \Pi[n]$ with respect to a set $P \subseteq \{1, \ldots, n\}$ is the set of all permutations in A that fix all elements of P:

$$\text{Stab}(A, P) = \{a \in A \mid \forall i \in P\colon a(i) = i\}. \quad (3)$$

For the graph G of Fig. 1, we have Stab(Aut(G), $\{1, 2\}$) = $\{\mathbf{12}3456, \mathbf{12}3465, \mathbf{12}4356, \mathbf{12}4365\}$, Stab(Aut($G$), $\{3, 4\}$) = $\{12\mathbf{34}56, 12\mathbf{34}65\}$, and Stab(Aut($G$), $\{3, 5\}$) = $\{12\mathbf{3}4\mathbf{5}6\}$.

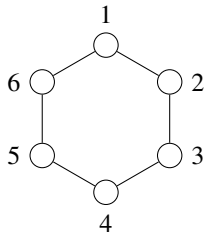

Fig. 2. A sample graph G'.

Definition 3 (EE ordered partition): For a given graph $G = (V, E)$, an ordered partition $\langle P_1, P_2, \ldots, P_s\rangle$ of V is *exploratory equivalent* if for all $i \in \{1, \ldots, s\}$ we have

$$\text{cover}(A_{i-1}, P_i) \text{ and } A_i = \text{Stab}(A_{i-1}, P_i),$$

where $A_0 = \text{Aut}(G)$.

For the graph G of Fig. 1, one of the EE ordered partitions is $\langle 1, 2 \mid 3, 4 \mid 5, 6\rangle$; the corresponding stabilizers are $A_1 = \{123456, 123465, 124356, 124365\}$, $A_2 = \{123456, 123465\}$, and $A_3 = \{123456\}$.

Definition 4 (EE partition): For a given graph $G = (V, E)$, a partition $\langle P_1, P_2, \ldots, P_s\rangle$ of V is *exploratory equivalent* if there exists an exploratory equivalent ordered partition $\langle P_{i_1}, P_{i_2}, \ldots, P_{i_s}\rangle$ for a set of distinct indices $i_j \in \{1, \ldots, s\}$.

Figure 3 shows all EE partitions of the graph G in Fig. 1.

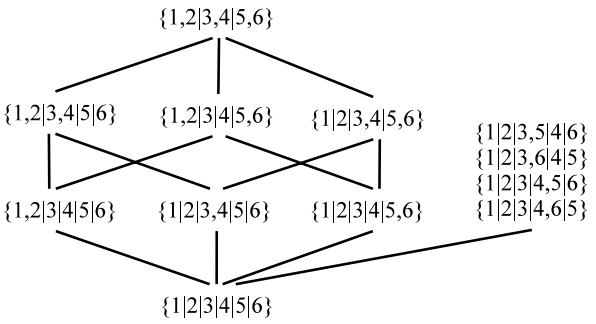

Fig. 3. The Hasse diagram of all EE partitions of the graph G of Fig. 1. (The four partitions on the right-hand side are actually four separate vertices in the diagram.)

As we mentioned in the introduction, an EE partition determines a set of subgraph isomorphism search constraints. For example, the EE partition $\{1 \mid 2 \mid 3, 4 \mid 5, 6\}$ determines the constraints $h(3) < h(4)$ and $h(5) < h(6)$, which can be safely used when searching for subgraph isomorphisms $h\colon G \to H$ in an arbitrary host graph H. Since an EE partition set with k vertices represents $k!$ permutations of those vertices, the corresponding constraint reduces the number of discoveries of each occurrence of G in H by a factor of $k!$. The score of an EE partition can thus be defined as follows:

Definition 5 (score of an EE partition): The *score* of an EE partition $\{P_1, \ldots, P_s\}$ is $\prod_{i=1}^{s} |P_i|!$.

The goal of the MAXEXPLOREQ problem is to find a maximum EE partition, i.e., one with the highest score:

*Definition 6 (*MAXEXPLOREQ*):* Given a graph G, find an EE partition with the maximum score.

The sole maximum EE partition of the graph G of Fig. 1 is $\{1,2 \mid 3,4 \mid 5,6\}$, with the score of $2!\,2!\,2! = 8$. One of the two maximum EE partitions of the graph G' of Fig. 2 $\{1,3,5 \mid 2 \mid 4 \mid 6\}$, with the score of $3!\,1!\,1!\,1! = 6$. The other is $\{2,4,6 \mid 1 \mid 3 \mid 5\}$. Note that $\{1,3,5 \mid 2,4,6\}$ is *not* an EE partition, since $\mathrm{Stab}(\mathrm{Aut}(G'),\{1,3,5\}) = \{123456\}$ and the resulting set of automorphisms covers only singletons.

For convenience, let us also define an exploratory equivalent set and exploratory equivalent vertices:

Definition 7 (EE set): For a graph $G = (V,E)$, a set $P \subseteq V$ is *exploratory equivalent* if there exists an EE partition that contains P.

For example, in the graph G of Fig. 1, the sets $\{3,5\}$, $\{5,6\}$, and $\{3,6\}$ are all exploratory equivalent. However, the set $\{3,5,6\}$ is not.

Definition 8 (EE vertices): Vertices v_1, ..., v_k are *exploratory equivalent* if the set $\{v_1,\ldots,v_k\}$ is exploratory equivalent.

We will now present an alternative interpretation of exploratory equivalence that will be used in some proofs. Let $V' = \{v_1, v_2, \ldots, v_k\} \subseteq V$ be a subset of vertices of an unlabeled graph $G = (V,E)$, and let Z_1, Z_2, ..., Z_k be mutually distinct labels. Let G^j denote a copy of the graph G in which the vertex v_i (for $i \in \{1,\ldots,k\}$) is labeled $\sigma_j(Z_i)$, where σ_j (for $j \in \{1,\ldots,k!\}$) represents the j-th permutation of the set $\{Z_1,\ldots,Z_k\}$. Now, the set V' is exploratory equivalent if the graphs G^1, ..., $G^{k!}$ are all mutually isomorphic. For instance, in the case of the graph G' of Fig. 2, the set $\{1,3,5\}$ is exploratory equivalent because all 3! graphs in Fig. 4 are mutually isomorphic.

Let $\mathcal{P} = \langle P_1, P_2, \ldots, P_s\rangle$ with $P_i = \{v_{i1}, v_{i2}, \ldots, v_{ik_i}\}$ be an ordered partition of the vertex set of an unlabeled graph G, and let Z_{11}, Z_{12}, ..., Z_{1k_1}, Z_{21}, Z_{22}, ..., Z_{2k_2}, ..., Z_{s1}, Z_{s2}, ..., Z_{sk_s} be mutually distinct labels that do not occur at any vertex in the graph G. Let $G^{r,j}$ $(1 \le r \le s)$ denote a copy of the graph G in which the vertices v_{i1}, v_{i2}, ..., v_{ik_i} (for $i \in \{1,\ldots,r-1\}$) are labeled Z_{i1}, Z_{i2}, ..., Z_{ik_i}, respectively, while the vertices v_{r1}, v_{r2}, ..., v_{rk_r} are labeled $\sigma_j(Z_{r1})$, $\sigma_j(Z_{r2})$, ..., $\sigma_j(Z_{rk_r})$, respectively, where σ_j (for $j \in \{1,\ldots,k_r!\}$) represents the j-th permutation of the set $\{Z_{r1},\ldots,Z_{rk_r}\}$. Now, the partition $\mathcal{P}$ is exploratory equivalent if we have $G^{i,1} \simeq G^{i,2} \simeq \ldots \simeq G^{i,k_i!}$ for each $i \in \{1,\ldots,s\}$. For instance, in the case of the graph G of Fig. 1, the ordered partition $\langle 1,2 \mid 3,4 \mid 5,6\rangle$ is exploratory equivalent because we have $G^{1,1} \simeq G^{1,2}$, $G^{2,1} \simeq G^{2,2}$, and $G^{3,1} \simeq G^{3,2}$ for the graphs of Fig. 5.

Alternatively, the partition $\mathcal{P} = \langle P_1, P_2, \ldots, P_s\rangle$ is exploratory equivalent if the set P_1 is exploratory equivalent and if for each $i \in \{2,\ldots,s\}$, the set P_i remains exploratory equivalent after the labels of the vertices of the sets P_j (for all $1 \le j < i$) have been fixed to Z_{j1}, ..., Z_{jk_j}.

V. THE COMPLEXITY OF MAXEXPLOREQ

In this section, we show that MAXEXPLOREQ is at least as hard as the graph isomorphism problem. We have the following theorem:

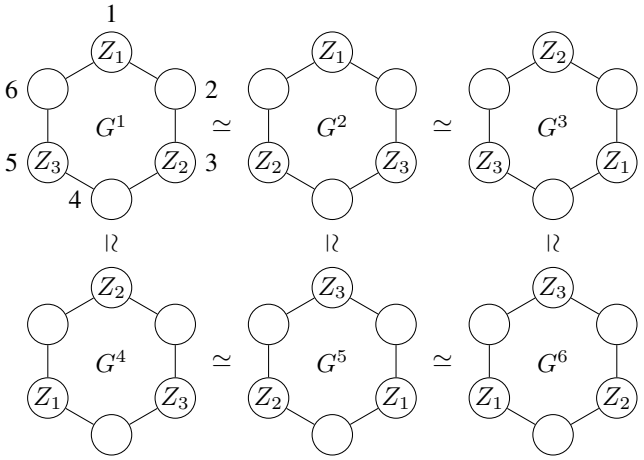

Fig. 4. These 6 isomorphic graphs prove that the set $\{1,3,5\}$ is exploratory equivalent for the graph G' of Fig. 2.

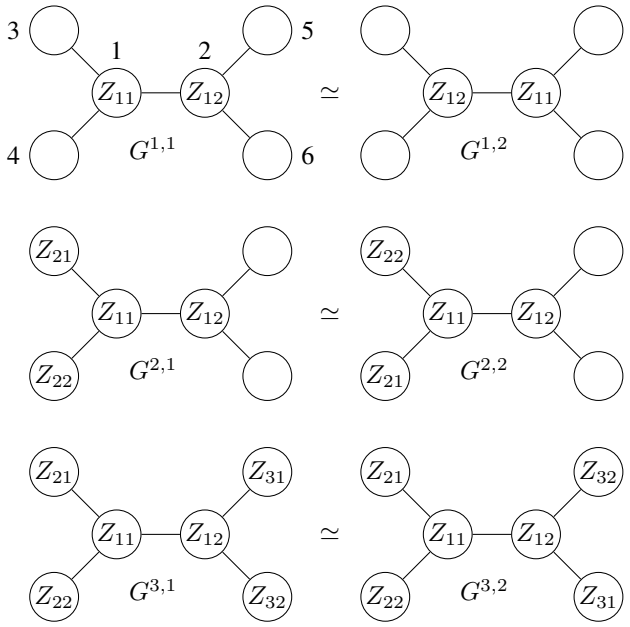

Fig. 5. The three pairs of isomorphic graphs proving that the ordered partition $\langle\{1,2\},\{3,4\},\{5,6\}\rangle$ is exploratory equivalent for the graph G of Fig. 1.

Theorem 1: The MAXEXPLOREQ problem is GI-hard.

Proof: The theorem can be proved by a polynomial-time reduction of the graph isomorphism problem to the MAXEXPLOREQ problem. Let G and H be graphs for which one would like to determine whether they are isomorphic. Let us form a graph G' by adding a vertex u_0 to the graph G and connecting it with all the vertices of G. In an analogous way, let us form a graph H' from the graph H (we call the added vertex v_0). Now we solve the MAXEXPLOREQ problem on the graph $G' \cup H'$, i.e., on the disjoint union of the graphs G' and H'. We claim that the graphs G and H are isomorphic if and only if any maximum EE partition contains a set with at least one vertex from G' and at least one vertex from H'. Let us prove this.

(If) If a maximum EE partition for the graph $G' \cup H'$ contains a set with vertices $u \in V(G')$ and $v \in V(H')$, then there exists an automorphism that maps u to v and v to u. Since the graphs G' and H' are both connected, such an automorphism can exist only if the graphs are isomorphic. This implies that the graphs G and H are isomorphic, too.

(Only if) If the graphs G are H isomorphic, then an EE partition for the graph $G' \cup H'$ cannot be maximum unless it contains at least one set with at least one vertex from both G' and H'. Indeed, in an EE partition that contains no such set, one can always join the singletons $\{u_0\}$ and $\{v_0\}$ into an EE set $\{u_0, v_0\}$ and thus obtain an EE partition with a higher score, since the vertices u_0 and v_0, owing to their degree, can only be exploratory equivalent with each other (and they are, if the graphs G' and H', and of course also G and H, are isomorphic). ■

Because of its GI-hardness, the MAXEXPLOREQ problem for general graphs is unlikely to be solvable in polynomial time. In the rest of this paper, we therefore restrict the problem to trees.

VI. SOLVING THE MAXEXPLOREQ PROBLEM ON TREES

A. Prerequisites

Let a graph $T = (V, E)$ be an arbitrary tree. For the sake of simplicity, let us assume that the tree is unlabeled; the algorithm could be fairly straightforwardly generalized to labeled trees. Since T is an arbitrary unrooted tree, we will only speak of *leaves* (vertices with degree 1) but not of the root, parents, and children. Before showing an algorithm for finding a maximum EE partition on T, let us present some auxiliary definitions and claims.

Definition 9 (distance): The *distance* between vertices u and v in a tree (denoted $d(u, v)$) is the number of edges on the (unique) path from u to v.

Definition 10 (neighborhood): In a tree, the *neighborhood* of a vertex u at a distance d is the subtree composed of all vertices v such that $d(u, v) \leq d$.

Definition 11 (eccentricity, center): The *eccentricity* of a vertex u in a tree is the maximum distance between u and any other vertex, i.e., $e(u) = \max_{v \in V} d(u, v)$. A *center* of the tree is a vertex with minimum eccentricity.

Theorem 2: Any tree has either one or two centers. If it has two, they are adjacent.

Proof: Let us focus on a longest path (several such paths are possible) in the tree, and let u and v be the two extreme vertices on that path. The distance between u and v is therefore the greatest possible in the tree. The eccentricity of any vertex w on the path is $e(w) = \max\{d(u, w), d(v, w)\}$; it obviously cannot be lower, but if it were greater, we could form a strictly longer path in the tree (passing through w, one of u and v, and the most remote vertex from w), contradicting our assumption. Any center c of the tree has to be located somewhere on the path from u to v, for if we had, say, a putative center c' outside of that path, then $e(c') = \max\{d(u, c'), d(v, c')\}$ would be greater than $e(c) = \max\{d(u, c), d(v, c)\}$. Since a center is a vertex with the lowest eccentricity, we have only two possibilities:

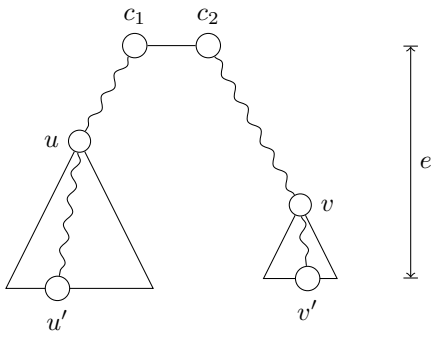

Fig. 6. An illustration of the proof of Lemma 3.

- If $d(u, v)$ is odd, the tree has exactly one center c, and it is located halfway between u and w, such that $d(c, u) = d(c, v)$.
- If $d(u, v)$ is even, the tree has two adjacent centers c_1 and c_2 such that $d(c_1, u) = d(c_2, v)$.

■

Lemma 3: Let c_1 and c_2 be the center(s) of the tree (by Theorem 2, we may have $c_1 = c_2$). If vertices u and v are exploratory equivalent, we have $d(u, c_1) = d(v, c_2)$.

Proof: Let us assume that $d(u, c_1) \neq d(v, c_2)$. Without loss of generality, we may further assume that $d(u, c_1) < d(v, c_2)$. Consider Fig. 6. Let u' be a leaf such that $d(c_1, u') = e(c_1) = e(c_2) = e$ (since c_1 is a center, such a leaf must exist). Likewise, let v' be a leaf such that $d(c_2, v') = e$. Since $d(u, c_1) < d(v, c_2)$, we have $d(u, u') > d(v, v')$. This means that there is a leaf at the distance of $d(u, u')$ from u, but there cannot be any leaf at the same distance from v. The neighborhoods of u and v at the distance $d(u, u')$ are therefore non-isomorphic, which implies that the vertices u and v cannot be automorphically mapped to each other. Consequently, the vertices u and v are not exploratory equivalent. ■

Definition 12 (centrifugal subtree): Let u be a vertex connected with vertices $v_1, \ldots, v_k$, and let v_i, for some $i \in \{1, \ldots, k\}$, be the sole vertex on the path from u to the center(s) of the tree. The *centrifugal subtree* of the vertex u is the tree composed of the vertex u and of all vertices on the paths starting at u, passing through $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_k$, respectively, and finishing at leaves.

Informally, the centrifugal tree of a given vertex u contains the vertex u and all vertices 'below' it in the direction away from the center(s). The triangles in Fig. 6 represent the centrifugal subtrees of the vertices u and v.

Lemma 4: If vertices u and v of the tree T are exploratory equivalent, they have isomorphic centrifugal subtrees.

Proof: If the centrifugal subtrees are not isomorphic, the vertices u and v cannot be automorphically mapped to each other, since there exists some distance d at which their neighborhoods are not isomorphic. Therefore, the vertices cannot be exploratory equivalent. ■

Lemma 5: If vertices $v_1, \ldots, v_k$ of the tree T are connected to the same vertex and all their centrifugal subtrees are

mutually disjoint and isomorphic, then the vertices v_1, ..., v_k are exploratory equivalent.

Proof: It is easy to see that the vertices v_1, ..., v_k (and with them the entire corresponding centrifugal subtrees) can be automorphically mapped to each other in all $k!$ possible ways, which means that they are exploratory equivalent. ■

Lemma 6: Let the tree have two distinct centers, c_1 and c_2. If the centrifugal subtrees of c_1 and c_2 are isomorphic, then the centers c_1 and c_2 are exploratory equivalent.

Proof: Here, the same argument applies as in the proof of Lemma 5. ■

Lemma 7: Let $\mathcal{P} = \langle P_1, \ldots, P_s \rangle$ be an ordered partition of the vertex set V such that the following holds:

- The set P_1 is exploratory equivalent in the sense of Lemma 5 or Lemma 6.
- Every set P_i with $i > 1$ is exploratory equivalent in the sense of Lemma 5.
- If the vertices of P_j, together with their common neighbor, all belong to the centrifugal subtree of some vertex $v \in P_i$, then $j > i$.

If all the above conditions are met, the partition $\mathcal{P}$ is exploratory equivalent.

Proof: By Lemmas 5 and 6, the set $P_1 = \{u_{11}, \ldots, u_{1k_1}\}$ is exploratory equivalent. Let us fix the labels of u_{11}, ..., u_{1k_1} to Z_{11}, ..., Z_{1k_1}. Is the set $P_2 = \{u_{21}, \ldots, u_{2k_1}\}$ still exploratory equivalent? Yes, owing to the third condition in the lemma, it holds that for each vertex u_{1i} ($i \in \{1, \ldots, k_1\}$) the centrifugal subtrees of the vertices $u_{21}, \ldots, u_{2k_2}$ are either disjoint from the centrifugal tree of u_{1i} or completely contained within it. In both cases, the vertices u_{21}, ..., u_{2k_2} (and with them the entire centrifugal subtrees) can be automorphically mapped to each other in all $k_2!$ possible ways, even if the vertices u_{11}, ..., u_{1k_1} have distinct unique labels. The second case is illustrated in Fig. 7. Since the same reasoning applies all the way to the set P_s, we can conclude that the partition is indeed exploratory equivalent. ■

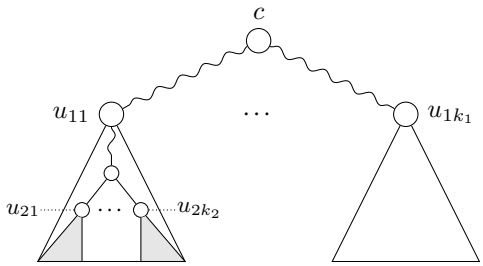

Fig. 7. An illustration of the second case in the proof of Lemma 7. The subtrees represented by the two large triangles are isomorphic, and so are those represented by the two small ones.

For the tree of Fig. 8, an ordered EE partition in the sense of Lemma 7 is $\langle 17 \mid 15, 16 \mid 11 \mid 12 \mid 13 \mid 14 \mid 1, 2, 3 \mid 4, 5 \mid 6, 7 \mid 8, 9, 10 \rangle$. Non-singleton sets are represented by different colors.

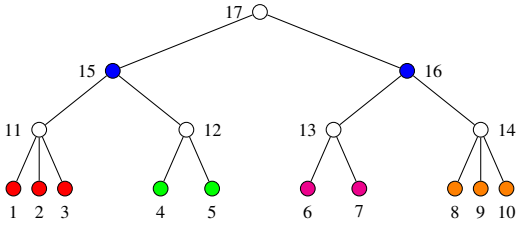

Fig. 8. A sample tree.

Lemma 8: If u and v are EE vertices at a distance greater than 2, then there also exist EE vertices u' and v' at a distance of at most 2.

Proof: Let us first assume that $d = d(u, v)$ is even. Let w be the sole vertex such that $d(u, w) = d(v, w) = d/2$. Now let u' and v' be the neighbors of w on the paths from w to u and w to v, respectively. The distance between u' and v' is therefore 2. We claim that the vertices u' and v' are exploratory equivalent if so are u and v. Indeed! The automorphism that maps u to v and vice versa maps the entire centrifugal subtree of u' to the centrifugal subtree of v' and vice versa. In particular, u' is mapped to v' and vice versa, which means that u' and v' are exploratory equivalent, too. (However, note that the sets $\{u, v\}$ and $\{u', v'\}$ cannot *both* be part of the same EE partition!)

If $d(u, v)$ is odd, then it follows from Lemma 3 that the tree T has two distinct centers, c_1 and c_2, and that $d(u, c_1) = d(v, c_2)$. An automorphism that maps u to v (and v to u) also maps c_1 to c_2 (and c_2 to c_1), implying that the vertices c_1 and c_2 are exploratory equivalent, too. ■

Lemma 9: If u_1, ..., u_k are EE vertices, then there also exist EE vertices u'_1, ..., u'_k such that $d(u_i, u_j) \leq 2$ for all distinct pairs $i, j \in \{1, \ldots, k\}$. Furthermore, if $k > 2$, then $d(u_i, u_j) = 2$ for all distinct pairs $i, j \in \{1, \ldots, k\}$.

Proof: The first part of the lemma is a straightforward generalization of Lemma 8. As for the second part, observe that if distinct vertices u, v, and w are exploratory equivalent, we cannot have $d(u, v) = d(v, w) = 1$ and $d(u, w) = 2$; such vertices would then form a 3-vertex line subgraph $u - v - w$, which can never have more than 2 automorphisms, but a set of three vertices can be exploratory equivalent only if at least 3! automorphisms exist. The case $d(u, v) = d(u, w) = d(v, w) = 1$ is clearly impossible in a tree, and so $d(u, v) = d(u, w) = d(v, w) = 2$ remains as the only possibility. ■

Lemma 10: There exists a maximum EE partition $\mathcal{P} = \{P_1, \ldots, P_s\}$ of the tree T such that for each $i \in \{1, \ldots, s\}$ the distance between each pair of vertices in P_i is at most 2.

Proof: Let $\mathcal{R} = \{R_1, \ldots, R_s\}$ be a maximum EE partition such that a set $R \in \mathcal{R}$ does not conform to the conditions in the lemma. By Lemma 9, the set R can be replaced by the corresponding EE set R' of vertices at a distance of at most 2. We now claim that the partition $\mathcal{R}' = \mathcal{R} \setminus \{R\} \cup \{R'\}$ is also exploratory equivalent. To see this, consider the operation performed in the proof of Lemma 8. Let v be the vertex such that $d(v, u_1) = \ldots = d(v, u_k)$. (If $k = 2$, such a vertex might not exist, but then we have $d(c_1, u_1) = d(c_2, u_2)$, and

the same logic applies.) By replacing the EE set $R = \{u_1, \ldots, u_k\}$ with the EE set $R' = \{u'_1, \ldots, u'_k\}$ (where $u'_1, \ldots, u'_k$ are the neighbors of v on the paths from v to $u_1, \ldots, u_k$, respectively), the resulting partition remains exploratory equivalent, since an automorphism that maps u_i to u_j also maps the entire path from v to u_i to the path from v to u_j and since the selection of R into an EE partition precludes the selection of any other set containing vertices on different paths from v to $u_1, \ldots, u_k$. However, the opposite is not necessarily the case. While the choice of R' does, of course, preclude the selection of R, it might not rule out everything 'between' R' and R. For instance, in the example shown in Fig. 8, it would be unwise to select the EE set $R_1 = \{1, 8\}$ into an EE partition, since the most we could then possibly attain would be the partition $\{1, 8 \mid 2, 3 \mid 4, 5 \mid 6, 7 \mid 9, 10 \mid \text{singletons}\}$. However, if we instead choose the set $R'_1 = \{15, 16\}$, we can obtain the clearly better (and indeed maximum) partition $\{15, 16 \mid 1, 2, 3 \mid 4, 5 \mid 6, 7 \mid 8, 9, 10 \mid \text{singletons}\}$. ■

The above lemma tells us that when searching for a maximum EE partition, we may safely ignore any pairs of vertices at the distance greater than 2. This important fact is the basis for the algorithm we show below.

B. The algorithm

We are now ready to present a polynomial-time algorithm that constructs a maximum EE partition for a given tree. The algorithm is shown as Alg. 1. At the beginning, the algorithm assigns the so-called *ornament* $*$ to each leaf of the given tree; all other vertices are assigned the ornament ϵ. The algorithm then proceeds in a reverse breadth-first fashion: in each iteration, the vertices connected to the (current) leaves that have at most one ϵ-ornamented neighbor receive their ornaments, constructed from the ornaments of their leaf neighbors. Simultaneously, the leaves are removed from the tree. The output of the algorithm is a partition of the vertex set of the original tree.

Figures 9 and 10 provide two examples for Alg. 1. The numbers beside the vertices indicate the order in which the ornaments are assigned to the vertices (of course, the order within each iteration is arbitrary), while the boxes show the ornaments. The vertices with the same non-white color belong to the same set in the returned partition. For the tree of Fig. 9, the algorithm thus produces the partition $\langle 9, 10 \mid 7, 8 \mid 5, 6 \mid 4 \mid 3 \mid 2 \mid 1 \rangle$. The partition for the tree of Fig. 10 is $\langle 8 \mid 6, 7 \mid 1 \mid 5 \mid 4 \mid 3 \mid 2 \rangle$.

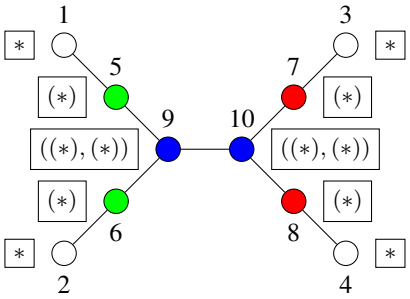

Fig. 9. The execution of Alg. 1 on a sample tree.

Algorithm 1 An algorithm for solving the MAXEXPLOREQ problem on a given tree T.

```
1:  function FindMaxPartition(T = (V, E))
2:      for all v ∈ V do orn(v) ← ϵ
3:      L ← the set of leaves of T
4:      for all v ∈ L do orn(v) ← *
5:      t ← 0
6:      while |V| > 2 do
7:          W = {w ∈ V | (∃v ∈ L: (w, v) ∈ E) and
8:              only one neighbor w' of w has orn(w') = ϵ}
9:          for all w ∈ W do
10:             ⟨r_1, ..., r_m⟩ ← the leaves connected to w,
11:                 lexicographically sorted by their ornaments
12:             orn(w) ← (orn(r_1), ..., orn(r_m))
13:             R ← {r_1, ..., r_m}
14:             while R ≠ ∅ do
15:                 r ← any vertex from R
16:                 t ← t + 1
17:                 P_t ← {r' ∈ R | orn(r') = orn(r)}
18:                 R ← R \ P_t
19:                 remove each vertex in P_t from T
20:         L ← the set of leaves of T
21:     if |V| = 1 then return ⟨V, P_t, P_{t-1}, ..., P_1⟩
22:     else
23:         {u, v} ← V
24:         if orn(u) = orn(v) then return ⟨V, P_t, ..., P_1⟩
25:         else return ⟨{u}, {v}, P_t, ..., P_1⟩
```

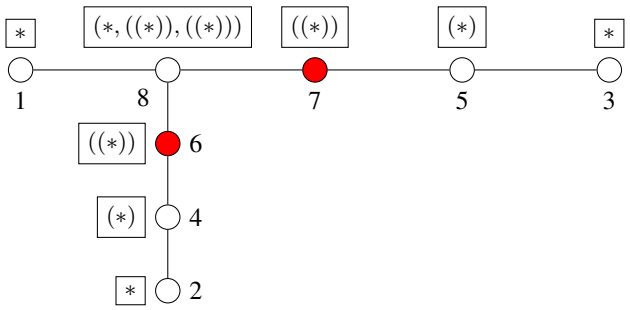

Fig. 10. The execution of Alg. 1 on a sample tree.

Lemma 11: In each iteration, the algorithm removes from the current tree all vertices that are farthest from the center(s) of the current tree.

Proof: In each iteration, the algorithm removes all leaves except those whose neighbor is connected to at least two non-leaves. However, such leaves cannot be at the greatest distance from the center(s). ■

Corollary 12: After each iteration, the center(s) of the resulting tree are coincident with those of the original tree.

Corollary 13: The (at most two) vertices that remain in the set V after the main loop of the algorithm are exactly the center(s) of the tree.

Lemma 14: At the end of the algorithm, two vertices have equal ornaments if and only if their centrifugal subtrees are isomorphic.

Proof: Since the algorithm proceeds from the leaves towards the centers, it builds the ornaments of individual vertices from the ornaments of their centrifugal subtrees. By construction, the ornament of a vertex reflects the structure of its centrifugal subtree. The lexicographical ordering ensures that two vertices with isomorphic centrifugal subtrees also have equal ornaments. As for the 'only if' part, consider that vertices with non-isomorphic centrifugal subtrees cannot possibly have equal ornaments; any valid ornament takes the form $(s_1, s_2, \ldots, s_k)$, where s_1, ..., s_k are individual subornaments, and there is exactly one way to split a valid ornament into valid constituents. Therefore, it cannot happen that two non-isomorphic subtrees 'accidentally' receive equal ornaments. ■

Lemma 15: For a given tree, the ordered partition produced by the algorithm is exploratory equivalent.

Proof: First, the properties from Lemmas 5, 6, and 14 ensure that each set from the EE partition is *individually* exploratory equivalent. Indeed, the algorithm adds a non-singleton set to the partition only if all vertices from that set have the same neighbor and isomorphic centrifugal subtrees. Second, does the returned ordered partition $\langle P_s, P_{s-1}, \ldots, P_1\rangle$ (where $s = t + 1$ or $t + 2$, depending on the situation after the main loop) conform to the conditions in Lemma 7, which guarantee 'EE-ness'? It does! The first two conditions are clearly met; the two centers, if they exist and if they are exploratory equivalent, constitute the set P_s. As for the third condition, consider that the algorithm 'peels' the tree from the leaves towards the centers and that the EE sets are stacked into the partition in the reverse order. These facts ensure that the centrifugal subtree of a vertex in P_j can be a subtree of the centrifugal subtree of a vertex in P_i only if $j > i$. ■

Lemma 16: The EE partition produced by the algorithm contains all possible EE sets $P = \{u_1, \ldots, u_k\}$ such that for each distinct pair $i, j \in \{1, \ldots, k\}$ we have $d(u_i, u_j) \leq 2$.

Proof: By Lemma 3, the vertices $\{u_1, \ldots, u_k\}$ of an EE set all have the same distance from the center. If the distance between each pair of them is at most 2, then they must be connected with the same vertex or, if $k = 2$, the vertices u_1 and u_2 can also be the two centers of the tree. By Lemma 3, EE vertices are always located at the same distance from the tree center(s); by Lemma 4, they must also have isomorphic centrifugal subtrees. The algorithm produces *all* such sets that fulfill these conditions: (1) it considers all sets of vertices that have the pairwise distance of exactly 2 and are located at the same distance from the tree center(s); (2) if the tree has two centers, the algorithm will, at the very end, certainly check whether they have the same ornaments; (3) the algorithm adds each such set of vertices to the output partition provided that the vertices have isomorphic centrifugal subtrees. ■

Theorem 17: For a given tree, Alg. 1 produces a maximum exploratory equivalent (ordered) partition.

Proof: By Lemma 15, the partition is exploratory equivalent. By Lemma 10, every tree has a *maximum* EE partition such that all pairwise distances in each EE set are at most 2. Since, by Lemma 16, the algorithm constructs an EE partition out of *all* such EE sets in the input tree, and since each of these sets contains as many vertices as possible (owing to line 17 in Alg. 1), the output EE partition certainly has the maximum score. ■

We have just proved that the algorithm indeed solves the MAXEXPLOREQ problem for trees. To get an estimate on the algorithm's complexity, we proceed as follows. For each vertex of the tree, one has to sort the signatures of its children (the neighbors of that vertex in its centrifugal subtree). Each vertex has $O(n)$ children, and the length of each ornament is $O(n)$, giving $O(n^2 \log n)$ to sort the signatures of the children of each vertex. Since there are $O(n)$ vertices, the total complexity of the algorithm is $O(n^3 \log n)$. We have the following theorem:

Theorem 18: Algorithm 1 is a polynomial-time algorithm for tree graphs.

VII. MAXEXPLOREQ ON SMALL TREES

In order to give us some insight into the problem, we performed a small empirical study of MAXEXPLOREQ on the set of small trees. This analysis will show us how the symmetries (that we can detect and exploit with MAXEXPLOREQ) are present in the studied set of trees.

Since trees are a ubiquitous structure, there are a lot of applications that can benefit from the symmetries found with our algorithm. An application that directly relates to the set of small trees is graphlet counting. In [28], the authors present a method for counting graphlets by exploiting many symmetries of small graphs (up to 5 nodes). Their method is currently considered one of the best, and with the use of MAXEXPLOREQ some of those symmetries could also be used to count larger graphlets much faster than with the straightforward approach.

For the analysis of this set of trees, we generated all nonisomorphic unlabeled trees of sizes 2 to 20 (let us call the set T_2^{20}) and computed MAXEXPLOREQ on every generated tree. Table I gives the number of trees for each size; in parentheses, we give the number of trees that have only the trivial automorphism, i.e., all sets in the MAXEXPLOREQ partition are singletons. Figure 11 shows the distribution of MAXEXPLOREQ score in T_2^{20}. This histogram shows that maximum EE partitions are non-trivial in a vast majority of trees.

To view the potential of MAXEXPLOREQ in more detail, let us examine trees of different sizes separately. For each separate set, we computed the median MAXEXPLOREQ score. The resulting chart is shown in Fig. 12. From this chart, we can see the potential speedup of at least half of the trees in the set of all trees with the same size. For example, for the trees of size 15, half of the trees have the potential speedup of 12, and for larger trees the median value is even larger, implying an almost exponential growth of the median value.

The two charts shown above demonstrate features of the MAXEXPLOREQ score, but they do not show the structure of individual partitions in any way. To show a feature of the MAXEXPLOREQ partitions, we measured the frequencies of the largest set in the partition (for each tree in T_2^{20}). Figure 13 shows the frequencies of these sets. In this histogram, we can see that most of the partitions are composed of pairs and triplets; the frequency of other partitions drops exponentially.

TABLE I. NUMBER OF NONISOMORPHIC TREES OF A SPECIFIED SIZE. IN PARENTHESES, THE NUMBER OF TREES WITHOUT AUTOMORPHISMS IS GIVEN.

size	2	3	4	5	6	7	8	9	10	11
#trees	1 (0)	1 (0)	2 (0)	3 (0)	6 (0)	11 (1)	23 (1)	47 (3)	106 (6)	235 (15)
size	12	13	14	15	16	17	18	19	20	
#trees	551 (29)	1301 (67)	3159 (139)	7741 (313)	19320 (671)	48629 (1487)	123867 (1487)	317955 (7264)	823065 (16137)	

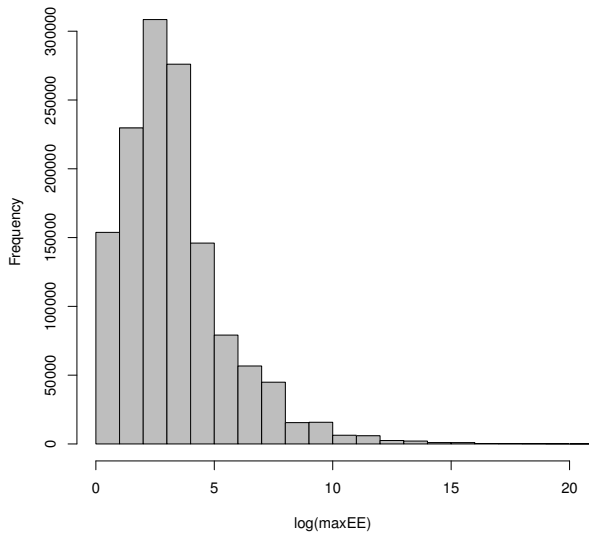

Fig. 11. The histogram of the frequencies of MAXEXPLOREQ values (logarithmic x axis) in T_2^{20}

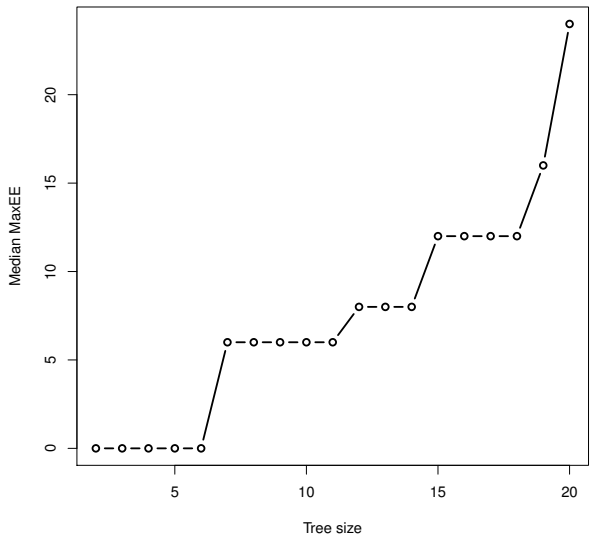

Fig. 12. The median values of MAXEXPLOREQ for all trees in T_2^{20} computed separately for all trees of the same size.

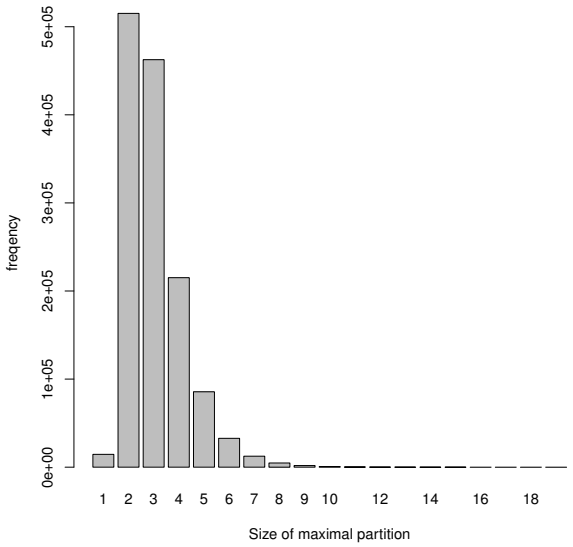

Fig. 13. Frequencies of the largest set in the MAXEXPLOREQ partition for each tree.

VIII. EXPLORATORY EQUIVALENCE AND THE SUBGRAPH ISOMORPHISM PROBLEM

We motivated exploratory equivalence by its application to the subgraph isomorphism problem. In this section, we will establish the relationship between these two concepts in more depth. First, note that there is a bijection between the set of automorphisms of a pattern graph G and the set of isomorphisms between G and each occurrence of G in an arbitrary host graph H:

Lemma 19: If G' is an occurrence of a graph G in a graph H, then for each automorphism of G there exists an isomorphism $G \to G'$, and vice versa.

Proof: A subgraph isomorphism between the graphs G and H is an isomorphism between two copies of the graph G (G and G' in our case). An automorphism of G can be interpreted in exactly the same way: as an isomorphism between two copies of G. ■

Lemma 20: Let $G = (V, E)$ be a pattern graph, and let G' be its occurrence in a host graph H. If a set $\{v_1, v_2, \ldots, v_k\} \subseteq V$ is exploratory equivalent, then there exists an isomorphism $h' \colon G \to G'$ such that $h'(v_1) < h'(v_2) < \ldots < h'(v_k)$.

Proof: Without loss of generality, we may assume that the graph G' consists of the vertices $u_1, u_2, \ldots, u_k$ such that $u_1 < u_2 < \ldots < u_k$. Let $h_0 \colon G \to G'$ be an isomorphism such that $h_0(v_i) = u_{\sigma(i)}$ (for each $i \in \{1, \ldots, k\}$), where σ is some permutation of the set $\{1, \ldots, k\}$. Since the set $\{v_1, v_2, \ldots,$

$v_k\}$ is exploratory equivalent, there exists an automorphism of G for each of the $k!$ permutations of the set. One of those automorphisms (e.g., h) has the property that $h(v_i) = v_{\sigma^{-1}(i)}$ for each $i \in \{1, \ldots, k\}$. Now, let us define the isomorphism $h' = h_0 \circ h$. For each $i \in \{1, \ldots, k\}$, we have $h'(v_i) = h_0(h(v_i)) = h_0(v_{\sigma^{-1}(i)}) = u_i$ and hence $h'(v_1) < h'(v_2) < \ldots < h'(v_k)$. ■

Theorem 21: Let G be a pattern graph, and let G' be its occurrence in a host graph H. If $\mathcal{P} = \langle P_1, \ldots, P_s\rangle$ (where $P_i = \{v_{i1}, \ldots, v_{ik_i}\}$ for $i \in \{1, \ldots, s\}$) is an EE ordered partition, then there exists an isomorphism $h' \colon G \to G'$ such that $h'(v_{i1}) < h'(v_{i2}) < \ldots < h'(v_{ik_i})$ for all $i \in \{1, \ldots, s\}$.

Proof: Since $P_1 = \{v_{11}, \ldots, v_{1k_1}\}$ is an EE set, Lemma 20 ensures the existence of an isomorphism $h_1 \colon G \to G'$ with the property $h_1(v_{11}) < h_1(v_{12}) < \ldots < h_1(v_{1k_1})$. Now, the fact that $\mathcal{P}$ is an EE ordered partition implies that the set P_2 remains exploratory equivalent even if we assign unique labels to the vertices in P_1. In other words, even if the set of all possible isomorphisms $h \colon G \to G'$ is restricted to those that satisfy $h(v_{11}) < h(v_{12}) < \ldots < h(v_{1k_1})$, there will still exist an isomorphism $h_2 \colon G \to H$ such that $h_2(v_{21}) < h_2(v_{22}) < \ldots < h_2(v_{2k_2})$ (in addition to $h_2(v_{11}) < \ldots < h_2(v_{1k_1})$). The same reasoning applies for P_3, P_4, etc., up to P_s. Therefore, there exists an automorphism h' such that $h'(u_{i1}) < \ldots < h'(u_{ik_i})$ for all $i \in \{1, \ldots, s\}$. ■

From Theorem 21, it follows that if $\mathcal{P} = \{P_1, \ldots, P_s\}$ is an EE partition of a pattern graph G, we can safely impose the constraints $h(v_{i1}) < \ldots < h(v_{ik_i})$ (for each $i \in \{1, \ldots, s\}$) while searching for subgraph isomorphisms $h \colon G \to H$ in an arbitrary host graph H. However, these constraints are not necessarily optimal in the sense of redundancy elimination: they might not reduce the number of residual isomorphisms between G and each of its occurrences in H to 1. For example, consider the graphs G and H in Fig. 14. The set of automorphisms of G (and simultaneously the set of G-to-H isomorphisms) is $\{1234, 2341, 3412, 4123, 4321, 3214, 2143, 1432\}$, and the maximum EE partition is $\{1, 3 \mid 2, 4\}$, giving the set of constraints $\{h(1) < h(3),\ h(2) < h(4)\}$. However, these constraints still retain two isomorphisms, 1234 and 2143. (In fact, two isomorphisms would remain regardless of how the vertices of H were numbered.) This means that in this case, the set of constraints resulting from the maximum EE partition does *not* eliminate the entire redundancy in subgraph isomorphism search and hence cannot be regarded as optimal. Incidentally, the optimal set of constraints is $\{h(1) < h(3) < h(4)\}$, but the partition $\{1, 3, 4 \mid 2\}$ is not exploratory equivalent.

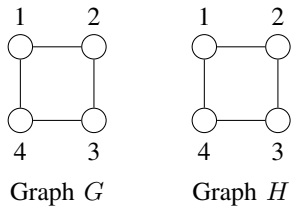

Fig. 14. A sample pattern graph G and host graph H.

In the case of general graphs, a maximum EE partition might lead to a suboptimal set of constraints because the number of automorphisms might be greater than the score of a maximum EE partition. In a tree, however, these two values are exactly the same.

We will state the following lemma without a formal proof. We can employ the same techniques as in the proofs of lemmas and theorems of Section VI. In addition, note that the set of automorphisms forms a group: if h and h' are automorphisms, then $h \circ h'$ and $h' \circ h$ are automorphisms, too.

Lemma 22: For any tree T, the following properties hold:

- If T contains an automorphism that maps a vertex u to a vertex v, then the vertices u and v are exploratory equivalent.
- Let c_1 and c_2 be the centers of the tree (we may also have $c_1 = c_2$). If the tree has an automorphism h such that $h(u) = v$, then (1) $d(u, c_1) = d(v, c_2)$ and (2) the centrifugal subtrees of u and v are isomorphic.
- For each nontrivial tree automorphism h, there exists a pair of vertices u and v such that $1 \le d(u, v) \le 2$ and $h(u) = v$.
- Let the set $P = \{v_1, \ldots, v_k\}$ be exploratory equivalent. Let h and h' be automorphisms such that $h(v_1) = h'(v_1) = \sigma(v_1)$, …, $h(v_k) = h'(v_k) = \sigma(v_k)$ for some permutation σ of the set P. If, on top of that, $h(u) = w_1$ and $h'(u) = w_2$ with $\{u, w_1, w_2\} \cap \{v_1, \ldots, v_k\} = \emptyset$ and $w_1 \neq w_2$, then the set $W = \{w_1, w_2\}$ is exploratory equivalent independently of the set P, which means that the sets P and W can both be part of the same EE partition.
- If the sets of tree vertices $P = \{u_1, \ldots, u_p\}$ and $Q = \{v_1, \ldots, v_q\}$ are both exploratory equivalent and if they cannot be extended by any other vertices without becoming non-EE, then the number of distinct automorphisms permuting the sets P and Q is equal to $p!\,q!$ only if (1) the centrifugal subtrees of the vertices in P are all pairwise disjoint from the centrifugal subtrees of the vertices in Q or (2) the vertices v_1, …, v_q are all part of the centrifugal subtree of a vertex u_i for some $i \in \{u_1, \ldots, u_p\}$ (or vice versa). Otherwise, the number of distinct automorphisms permuting the sets P and Q is $p! = q!$. In this case, we must have $p = q$, and each of the vertices of Q is located in the centrifugal subtree of a different vertex of P (or the other way around).

Theorem 23: Let $\mathcal{P} = \langle P_1, P_2, \ldots, P_s\rangle$ be the maximum EE ordered partition obtained by Alg. 1 for a given tree T. Then the number of automorphisms of T is $|P_1|! \ldots |P_s|!$.

Proof: If $s = 1$, the vertices of P_1 can be mapped to each other in all $|P_1|!$ ways, which means that there are at least $|P_1|!$ automorphisms. However, the number of automorphisms is exactly $|P_1|!$. Suppose there were two automorphisms, h and h', for the same permutation of the set P_1. In particular, if $P_1 = \{v_1, \ldots, v_k\}$, suppose that $h(v_i) = h'(v_i) = \sigma(v_i)$ for all $i \in \{1, \ldots, k\}$ and for some permutation σ of P_1. For h and h' to be distinct, we must have, say, $h(u) = w_1$ and $h'(u) = w_2$ with $w_1 \neq w_2$ and $u, w_1, w_2 \in V \setminus P_1$. However, in this case, the set $\{w_1, w_2\}$ is exploratory equivalent independently

of the set P_1 (Lemma 22) and is therefore part of the same maximum EE partition as the set P_1.

Now, let us assume that the theorem holds for some $s > 1$, and let us verify that it also holds for $s + 1$. Indeed: for any fixed permutation of the vertices in the set $P_1 \cup \ldots \cup P_s$, there are exactly $|P_{s+1}|!$ automorphisms, one for each permutation of the set P_{s+1}, which, together with the inductive assumption, gives the property stated in the theorem. The number of automorphisms cannot possibly be more than that; if it were, that would imply the exploratory equivalence of some set not present in $\mathcal{P}$ (independently of the sets in $\mathcal{P}$) or the fact that the set P_s is not at the bottom of the centrifugal subtree containment hierarchy (which would, in turn, imply that $\mathcal{P}$ is not an EE ordered partition). ■

Theorem 23 implies the optimality of the subgraph isomorphism constraints derived from a maximum EE partition for an arbitrary tree. In the search for occurrences of a pattern tree T in an arbitrary host graph H, the use of these constraints reduces the number of generated isomorphisms between T and each of its occurrences in H to 1, thus eliminating the automorphism-induced redundancy completely.

IX. Conclusion

Recently, we defined the so-called MaxExplorEq problem, the goal of which is to find a maximum exploratory equivalent (EE) partition of the vertex set of a given graph G. This problem is closely related to the problem of finding occurrences of a graph G in a graph H (the subgraph isomorphism problem), since every EE partition of G determines a set of redundancy reduction constraints that can be safely imposed during the subgraph isomorphism search. In the MaxExplorEq problem, we try to find an EE partition that gives rise to the optimal set of constraints in terms of redundancy elimination in subgraph isomorphism search.

In this paper, we proved that MaxExplorEq is GI-hard, which means that it is unlikely to be solvable in polynomial time. For this reason, we restricted the MaxExplorEq problem to an important subclass of graphs — the class of trees. By devising a polynomial-time algorithm, we showed that the restricted MaxExplorEq problem belongs to the complexity class P. Our algorithm finds a maximum EE partition in time $O(n^3 \log n)$, where n is the number of vertices of the input tree. Note that in contrast to the algorithms presented in our previous paper [9], Alg. 1 does not require or enumerate the set of automorphisms of the given tree. If it did, it could not possibly run within polynomial-time bounds, since the number of automorphisms for a tree with n vertices can be up to $(n-1)!$.

Besides that, we showed that the score of the maximum EE partition is equal to the number of automorphisms in the case of trees, but not necessarily in the case of general graphs. For any tree, a maximum EE partition thus gives rise to an optimal set of subgraph isomorphism search constraints.

To demonstrate the large potential of MaxExplorEq, we performed a small empirical study on the set of all trees of sizes 2 to 20. The study demonstrates that large speedups could be obtained in various search algorithms, especially for finding tree-shaped patterns in larger structures. The automorphisms on trees have been, of course, well known for a long time; however, our algorithm finds the partition of nodes that can be completely interchanged in search algorithms, and thus we give an explicit recipe on how to exploit these symmetries.

Could we apply the approach presented in this paper to general graphs? The GI-hardness of the MaxExplorEq problem does not offer much hope to find a polynomial-time algorithm for arbitrary graphs. Nevertheless, the lemmas and theorems of Section VI — if, of course, they could really be extended to arbitrary graphs in some way — might at least give rise to a relatively efficient branch-and-bound algorithm for finding a maximum EE partition. However, a number of problems will have to be solved before arriving at a viable algorithm. A general graph might have an arbitrary number of centers, and it is not yet clear whether the concepts such as 'centrifugal subtree' could be generalized at all.

References

[1] J. Leskovec and E. Horvitz, "Geospatial structure of a planetary-scale social network," *IEEE Transactions on Computational Social Systems*, vol. 1, no. 3, pp. 156–163, 2014. doi: 10.1109/TCSS.2014.2377789

[2] D. K. Agrafiotis, V. S. Lobanov, M. Shemanarev, D. N. Rassokhin, S. Izrailev, E. P. Jaeger, S. Alex, and M. Farnum, "Efficient Substructure Searching of Large Chemical Libraries: The ABCD Chemical Cartridge," *Journal of Chemical Information and Modeling*, pp. 3113–3130, 2011. doi: 10.1021/ci200413e

[3] J. M. Barnard, "Substructure searching methods: Old and new," *Journal of Chemical Information and Computer Sciences*, vol. 33, no. 4, pp. 532–538, 1993. doi: 10.1021/ci00014a001

[4] M. O. Jackson, *Social and Economic Networks*. Princeton, NJ, USA: Princeton University Press, 2008. ISBN 0691134405, 9780691134406

[5] D. Knoke, *Political Networks: The Structural Perspective*, ser. Structural Analysis in the Social Sciences. Cambridge University Press, 1994. ISBN 9780521477628. [Online]. Available: http://books.google.si/books?id=9djTlBLaOccC

[6] B. Hopkins, "Kevin Bacon and graph theory," *Problems, Resources, and Issues in Mathematics Undergraduate Studies (PRIMUS)*, vol. 14, no. 1, pp. 5–11, 2004. doi: 10.1080/10511970408984072

[7] L. D. Sailer, "Structural equivalence: Meaning and definition, computation and application," *Social Networks*, vol. 1, pp. 73–90, 1978. doi: 10.1016/0378-8733(78)90014-X

[8] M. G. Everett and S. P. Borgatti, "Regular equivalence: General theory," *Journal of mathematical sociology*, vol. 19, no. 1, pp. 29–52, 1994. doi: 10.1080/0022250X.1994.9990134

[9] J. Mihelič, L. Fürst, and U. Čibej, "Exploratory equivalence in graphs: Definition and algorithms," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems (FedCSIS), Warsaw, Poland, September 7-10, 2014*, 2014. doi: 10.15439/2014F352 pp. 447–456.

[10] F. Margot, "Pruning by isomorphism in branch-and-cut," *Mathematical Programming*, vol. 94, pp. 71–90, 2002. doi: 10.1007/s10107-002-0358-2

[11] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, "Orbital branching," in *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference, Ithaca, NY, USA, June 25–27, 2007, Proceedings*, 2007. doi: 10.1007/978-3-540-72792-7_9 pp. 104–118.

[12] D. Erwin and F. Harary, "Destroying automorphisms by fixing nodes," *Discrete mathematics*, vol. 306, pp. 3244–3252, 2006. doi: 10.1016/j.disc.2006.06.004

[13] R. Ladner, "On the structure of polynomial time reducibility," *Journal of the ACM*, vol. 22, pp. 155–171, 1975. doi: 10.1145/321864.321877

[14] B. D. McKay, "Practical graph isomorphism," in *Congressus Numerantium 30: 10th Manitoba Conference on Numerical Mathematics and Computing, Winnipeg, Canada, 1980*, 1981, p. 45–87.

[15] B. D. McKay and A. Piperno, "Practical graph isomorphism, II," *Journal of Symbolic Computation*, vol. 60, pp. 94–112, 2013. doi: 10.1016/j.jsc.2013.09.003

[16] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments, ALENEX 2007, New Orleans, LA, USA, January 6, 2007*, 2007. doi: 10.1137/1.9781611972870.13

[17] ——, "Conflict propagation and component recursion for canonical labeling," in *Theory and Practice of Algorithms in (Computer) Systems – First International ICST Conference, TAPAS 2011, Rome, Italy, April 18–20, 2011, Proceedings*, ser. LNCS 6595. Springer, 2011. doi: 10.1007/978-3-642-19754-3_16 pp. 151–162.

[18] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov, "Exploiting structure in symmetry detection for CNF," in *Proceedings of the 41st Design Automation Conference, DAC 2004, San Diego, CA, USA, June 7-11, 2004*, 2004. doi: 10.1145/996566.996712 pp. 530–534.

[19] P. T. Darga, K. A. Sakallah, and I. L. Markov, "Faster symmetry discovery using sparsity of symmetries," in *Proceedings of the 45th Design Automation Conference, DAC 2008, Anaheim, CA, USA, June 8-13, 2008*, 2008. doi: 10.1145/1391469.1391509 pp. 149–154.

[20] H. Katebi, K. A. Sakallah, and I. L. Markov, "Symmetry and satisfiability: An update," in *Theory and Applications of Satisfiability Testing – SAT 2010, 13th International Conference, Edinburgh, UK, July 11-14, 2010, Proceedings*, ser. LNCS 6175. Springer, 2010. doi: 10.1007/978-3-642-14186-7_11 pp. 113–127.

[21] M. N. Velev and R. E. Bryant, "Effective use of boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors," in *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, 2001. doi: 10.1145/378239.378469 pp. 226–231.

[22] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974. ISBN 978-0201000290

[23] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC), Shaker Heights, Ohio, USA, May 3-5, 1971*, 1971. doi: 10.1145/800157.805047 pp. 151–158.

[24] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009. ISBN 9780521424264

[25] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*. Springer, 2011. ISBN 978-3-642-16532-0

[26] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism," *Journal of the ACM*, vol. 23, pp. 31–42, 1976. doi: 10.1145/321921.321925

[27] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004. doi: 10.1109/TPAMI.2004.75

[28] T. Hočevar and J. Demšar, "A combinatorial approach to graphlet counting," *Bioinformatics*, vol. 30, no. 4, pp. 559–565, 2014. doi: 10.1093/bioinformatics/btt717