Stockholms universitet

Isomorphism testing of level-1 networks in linear time using the AHU algorithm.

Isomorfitestning av level-1 nätverk i linjär tid med hjälp av AHU algoritmen.

Anton Alfonsson

Handledare: Marc Hellmuth
Examinator: Lars Arvestad
Inlämningsdatum: August 29, 2024

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

The graph isomorphism problem is a fascinating problem due it being a well known problem in NP for which we do not know if it belongs to either P or the class of NP-complete problems. There do however exist polynomial time algorithms for checking isomorphism of certain subsets of graphs. One of those subsets is rooted trees, where the AHU algorithm for checking isomorphism of rooted trees has a linear runtime. The subset of level-1 networks is a bit more complex than that of rooted trees. They can very roughly be described as rooted trees where branches are allowed to merge together again creating cyclic blocks in the network, but only in such a way that those blocks do not overlap. This paper presents a linear time complexity method for transforming a level-1 network into a labeled rooted tree. The transformation is done in such a way that two of these trees are isomorphic if and only the networks they were created from are isomorphic. This way the problem of checking for isomorphism of level-1 networks is reduced to that of checking for isomorphism of their corresponding labeled rooted trees, which can be done in linear time with the AHU algorithm.

Sammanfattning

Grafisomorfiproblemet är ett fascinerande problem då det känt tillhör NP men inte är känt att tillhöra varken P eller de NP-fullständiga problemen. Om man inför begränsningar på vilken typ av grafer som ska undersökas så existerar dock algoritmer med polynomiell körtid för att avgöra om graferna isomorfa. En sådan begränsad typ av grafer är rotade träd, för vilka AHU algoritmen kan avgöra om två rotade träd är isomorfa i linjärtid. En något mer komplex typ av grafer är level-1 nätverk. Dessa kan slarvigt beskrivas som rotade träd där grenar tillåts växa ihop igen så att cykliska block uppstår i nätverket, men bara på ett sådant sätt att dessa block inte överlappar. Denna uppsats presenterar en linjärtidsmetod för att transformera ett level-1 nätverk till ett uppmärkt rotat träd. Transformationen görs på ett sådant sätt att två av dessa träd är isomorfa om och endast om nätverken de skapades ifrån är isomorfa. På detta sätt reduceras problemet att avgöra om två level-1 nätverk är isomorfa till att avgöra om två uppmärkta rotade träd är ismorfa, vilket kan göras i linjärtid med AHU algoritmen.

Contents

Abstract 2
Sammanfattning 2

1 Introduction **4**
1.1 Graph isomorphism problem 4
1.2 The AHU algorithm 4
1.3 Level-1 networks 6
1.4 Isomorphism checking of level-1 networks and planar graphs 6
1.5 Goal of this paper 7
1.6 Outline 7

2 Preliminaries **8**
2.1 DAGs and Level-1 networks 8
2.2 The AHU algorithm 11

3 Isomorphism checking of level-1 networks in linear time. **12**
3.1 Untangling a level-1 network 12
3.2 Time complexity of solution 26

4 Conclusion **30**

1 Introduction

1.1 Graph isomorphism problem

The graph isomorphism problem is of theoretical interest due to it being one of few problems in NP that is not known to belong to either P or the class of NP-complete problems, while also having practical applications in for example bio chemistry [1, 2]. Roughly speaking the graph isomorphism problem is the problem of determining whether two different graphs have the same underlying structure, and two graphs have the same structure if it is possible to move the nodes around in one of the graphs to make it look like the other graph without adding or removing any edges.

While the complexity class of the graph isomorphism problem is not known, there exists polynomial time algorithms for determining if two graphs are isomorphic if you restrict the class of input graphs to some subset of graphs such as rooted trees [3] or planar graphs [4].

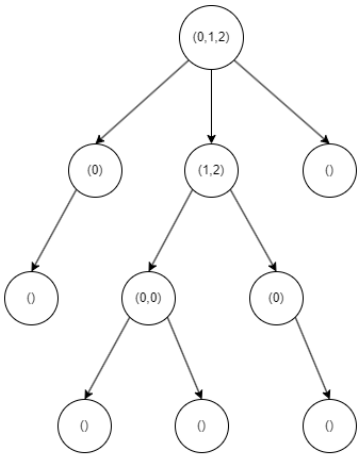
1.2 The AHU algorithm

When the graph isomorphism problem is restricted to rooted trees as the input graphs, a linear time algorithm has been known since 1974. The AHU algorithm is an algorithm for determining if two rooted trees are isomorphic that was first introduced in [3] as an example application for lexicographical sorting. The algorithm is named after its creators Aho, Hopcraft and Ullman. A formal proof of the correctness and linear runtime of the algorithm can be found in [5]. For an in depth explanation of how and why the algorithm works the reader is encouraged to look at [6], but here follows a rough overview.

For two rooted trees of height h, every vertex is first marked with it's level such that the roots are the only vertices with level h, their children have level $h - 1$ and so on. Since the vertices on the final level can not have any children, level 0 will be filled with nothing but leaves. All leaves in the trees are given the same initial label "()", an empty tuple. The algorithm then checks one level at a time in both trees and make sure the multiset of labels in both trees are equal. In practice this is usually done with lexicographically sorted lists instead of multisets, though it has been shown in [7] that products of prime numbers can instead be used while retaining linear time complexity. For level 0 the algorithm just checks if the trees have an equal number of leaves on that level and then moves on to the next level.

On level 1 any non-leaf vertex is given an initial label determined by the labels of it's children, such that if a vertex v has children $c_1, c_2, ..., c_n$ lexicographically ordered based on their labels $l_1, l_2, ..., l_n$, then v has initial label $(l_1, l_2, ..., l_n)$. If the initial labels are the same on this level in both trees, the initial labels are compressed using a compression function such that whichever initial label on the level is the lowest (with regard to lexicographical ordering) corresponds to the compressed label (0), the second to lowest initial label corresponds to the compressed label (1) and so on. Every vertex on the level is then assigned the compressed label that corresponds to its initial label. The process then repeats for the next level, using the compressed labels of the children to create the initial labels for the new level. This process continues until either two levels have non-equal multisets of labels, in which case the trees are not isomorphic, or until the roots have been assigned the same labels, in which case the trees are isomorphic. While the compression step might seem counter intuitive or overly complicated, it's necessary to achieve linear runtime in the algorithm.

At the end of the section in [3] where the algorithm is introduced the authors also note that it can be applied to labeled trees by appending the label of each vertex to the tuples assigned to the vertices during the runtime of the algorithm.



AHU with lexicographically sorted lists		
Level	Initial Label	Compressed label
3	(0,1,2)	(0)
2	() (0) (1,2)	(0) (1) (2)
1	() (0) (0,0)	(0) (1) (2)
0	()	(0)

Figure 1.1: A graph where the vertices of each level are marked with the initial labels they get assigned during runtime of the AHU algorithm when implemented with lexicographically sorted lists. Next to the graph is shown which compressed label that corresponds to each initial label at every level.

1.3 Level-1 networks

While rooted trees are a commonly occurring type of graph, sometimes more general types of networks are needed. An example is when using phylogenetic trees to model evolutionary history, but evolutionary events such as horizontal gene transfer and hybridisation can't be modelled without allowing some nodes in the graph to have more than one parent [8]. To solve this it's instead better to use a phylogenetic network [9]. While some literature restricts the term phylogenetic network to only include so-called binary networks (e.g. [9] and [10]), in this paper we will not impose this restriction, any results will however still generalize to binary phylogenetic networks as well.

In [9] they also introduce a classification of phylogenetic networks based on the maximum number of nodes with more than one parent (called *hybrid nodes*) that exist within any biconnected component or *block* of the network. A network is called a level-k[1] network based on this number. So a rooted tree has zero hybrid vertices and is a level-0 network, while a network where every biconnected component contains at most one hybrid-vertex is a level-1 network. Level-1 networks are useful in biology, and modelling your data such that the level of the networks will be restricted to level-1 allows the use of effective software tools for inferring a network from the data [11].

1.4 Isomorphism checking of level-1 networks and planar graphs

In [4] the authors show that it's possible to create a linear time algorithm to check for isomorphism of planar graphs. It has also been shown in [10] that binary level-1 networks are always (outer) planar. Therefore, any algorithm that can check for isomorphism of planar graphs could also be used to check for isomorphism of binary level-1 networks.
To demonstrate that it's possible for all *all* level-1 networks to check for isomorphism in linear time, one approach would be to show that all level-1 networks are planar and not just binary ones. However, in this paper this approach will not be used. One of the main reasons for this is that the linear time algorithm for isomorphism checking in [4] is, in the authors' own words, "mostly theoretical, demonstrating existence rather than providing a practical algorithm" due to a large constant factor in the running time. A more practical quadratic time algorithm for isomorphism testing of planar graphs is given in [12].

The aim here is to provide a practical linear time algorithm that works for all level-1 networks, thus avoiding the reliance on planarity arguments.

[1] In the original paper they refer to it as a level-f network.

1.5 Goal of this paper

In this paper the goal is to introduce a practical algorithm for checking isomorphism of level-1 networks with a linear time bound. That is, for two level-1 networks $N = (V, E)$, and $N' = (V', E')$, the runtime of the algorithm is in $\mathcal{O}(|V| + |E|)$. To achieve this goal, a method of converting a level-1 network N into a rooted labeled tree T_N is given such that N is isomorphic to a network N' if and only if T_N is isomorphic to $T_{N'}$. I will then show that the size of T_N is bounded linearly by the size of N and the construction can be done in linear time. This makes it so that the problem of checking isomorphism of two level-1 networks is reduced to the problem of checking isomorphism of their corresponding labeled trees, which can be done in linear time with the AHU algorithm.

1.6 Outline

Here follows a brief description of the organization of this paper. Section 2 states the definitions needed for the rest of the paper. It contains definitions for directed graphs, networks, and isomorphism of graphs and labeled networks. In section 3 the correctness and time complexity bound of the algorithm is shown. It first presents the method for converting a level-1 network N into it's corresponding labeled tree (T_N, t_N). That is then followed by proofs that two output trees are isomorphic if and only if the input networks are isomorphic. Proofs are also given that the size of the output trees are linearly bound by the size of the input networks and that the transformation can be done in linear time. In section 4 the results and possible ideas for future investigation are discussed.

2 Preliminaries

In this chapter we will introduce the concepts of DAGs, level-1 networks, rooted trees and the AHU algorithm.

2.1 DAGs and Level-1 networks

A *directed graph* $G = (V, E)$ consists of a vertex set $V(G) := V$ and an edge set $E(G) := E \subseteq V \times V \setminus \{(v, v) : v \in V\}$. Every directed graph G also has a corresponding undirected graph $G' = (V, E')$ where $E' = \{\{u, v\} \subseteq V \mid (u, v) \in E\}$. For the rest of this paper the word *graph* is used to mean directed graph unless otherwise specified.

For an edge $(u, v) \in E$ in a graph $G = (V, E)$, the vertex u is called the *tail* of the edge and v is called the *head* of the edge. Both u and v are said to be *incident* with (u, v). The *degree* of a vertex $v \in V$, $deg_G(v)$ is the number of edges $e \in E$ such that v is incident with e. The *indegree* of a vertex $v \in V$, $indeg_G(v)$ is the number of edges $e \in E$ where v is the head of e, and the *outdegree* of v, $outdeg_G(v)$, is the number of edges $e \in E$ where v is the tail of e. The subscript G is dropped when it's clear from context in what graph the degree is counted.

A *path* $P = v_0 \rightsquigarrow v_n$ in a graph $G = (V, E)$ is a sequence of vertices $v_0, v_1, v_2, ..., v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for every $i \in \{0, 1, 2, ..., n-1\}$, and for $j, k \in \{0, 1, 2, ..., n-1\}$, if $j \neq k$ then $v_j \neq v_k$, and $v_j \neq v_n$ for $j \neq 0$. The last condition allows for the final vertex and the first vertex to be the same, but no other repeats of vertices are allowed. A path $v_0, v_1, ...v_n$ has *length* n. A *cycle* is a path $v \rightsquigarrow v$ of length at least 2. A graph without cycles is called *acyclic*. For every vertex there is a trivial path $v \rightsquigarrow v$ from v to v of length zero. For a path $P = v_0, v_1, ..., v_n$, the path $P' = v_i, v_{i+1}, ..., v_{i+k}$ such that $i + k \leq n$ and $i \geq 0$ is a *subpath* of P.

Subgraph

A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

For a graph $G = (V, E)$ and a subset $X \subseteq V$ of the vertex set, the *induced subgraph* $G[X] = (X, E')$ is the graph with vertex set X and edge set $E' = \{(u, v) \in E \mid u \in X \text{ and } v \in X\}$.

Connectedness

A graph $G = (V, E)$ is *connected* if for every pair of vertices $u, v \in V$ there is a path $u \rightsquigarrow v$ in the corresponding undirected graph G'. The graph is *biconnected* if it's connected and, for every set $X = V \setminus \{v\}$ where v is some vertex in V, the induced subgraph $G[X]$ is connected. Graphs with only a single vertex are considered biconnected.
A *connected component* (also referred to as just a *component*) $C = (V', E')$ of a graph G is an inclusion-maximal connected induced subgraph $G[V']$. That is, it is connected and for any set V'' such that $V' \subset V'' \subseteq V$, the induced subgraph $G[V'']$ is not connected.
A *biconnected component* $B = (V', E')$ is an inclusion-maximal biconnected induced subgraph. A biconnected component is also called a *block*. A block with more than one edge is called a *non-trivial block*.

Network

A *DAG* is a directed acyclic graph. A Network N is a DAG $N = (V, E)$ with a vertex $\rho \in V$ called the *root* such that $indeg(\rho) = 0$, and no other vertex has indegree zero. In a network, a vertex v is called a *tree-vertex* if $indeg(v) = 1$ and $outdeg(v) > 0$, a *hybrid-vertex* if $indeg(v) > 1$, and a *leaf* if $outdeg(v) = 0$.[1] If a network has no hybrid-vertices it's called a *rooted tree*.
In a network $N = (V, E)$, if there is an edge $(u, v) \in E$, then u is called a *parent* of v, and v is a *child* of u.
If for vertices $u, v \in V$ there is a path $u \rightsquigarrow v$ then u is called an *ancestor* of v, and v is a *descendant* of u. If it's neither the case that u is an ancestor of v nor v is a ancestor of u, then u and v are *incomparable*.

Lemma 2.1. For a network $N = (V, E)$ with root ρ, for every vertex $v \in V$ there is a path $\rho \rightsquigarrow v$.

Proof. Since ρ is the only vertex with indegree 0 in N, it must either be the case that $v = \rho$ (in which case we know that we have the trivial path $\rho \rightsquigarrow \rho$) or v has a parent u in N. We then know that either $u = \rho$ or u will also have parent w in N. This argument can then be repeated for w, but since N is acyclic no vertex can be a parent to one of it's ancestors, so repeating will eventually encounter a vertex whose parent is ρ since N is finite. □

Lemma 2.2. Every block B in a network N has a unique vertex ρ_B such that ρ_B is the only vertex in B with $indeg_B(\rho_B) = 0$.

For a proof of this, refer to Lemma 8 in [13].

Corollary 2.3. Every block B in a network N is a network.

Proof. Since N is a DAG and every block B is a subgraph of N, it follows that every block B is a DAG, and since every block has a unique root ρ_B it also follows that every block B is a network. □

[1] Note that a hybrid-vertex can be a leaf.

Level-1 Networks

A *Level-k network* is a network $N = (V, E)$ such that every block B in N contains at most k hybrid-vertices distinct from ρ_B. In particular, a *level-1 network* is a network N such that every non-trivial block B in N contains at most one hybrid-vertex η_B distinct from ρ_B. Here follows some properties of non-trivial blocks in level-1 networks that will be used for the proofs in this paper. There will be extensive references to [13] for proofs of these properties.

Lemma 2.4. *For a level-1 network N with distinct non-trivial blocks B and B', if there is a vertex $v \in V(N)$ such that $v \in V(B)$ and $v \in V(B')$, then $v \in \{\rho_B, \rho_{B'}\}$.*

For a proof of this refer to lemma 9 in [13].

Lemma 2.5. *Every non-trivial block B in a level-1 network N contains exactly one hybrid-vertex η_B distinct from ρ_B.*

Proof.
In a non-trivial block B there must be some vertex $v \in V(B)$ such that $outdeg_B(v) = 0$, otherwise B would contain a directed cycle. From the section defining directed acyclic graphs in [13] we can see that these vertices with outdegree zero in B will be hybrid vertices. From the definition of a level-1 network we have that a non-trivial block B in a level-1 network can have at most one hybrid-vertex distinct from ρ_B, and we just noted that every non-trivial block B must have at least one hybrid-vertex v with $outdeg_B(v) = 0$. Since $outdeg_B(\rho_B) \neq 0$ this means that we have exactly one hybrid-vertex distinct from ρ_B in B. □

For a non-trivial block B in a level-1 network N, the vertex η_B will be referred to as its *designated hybrid* or *unique hybrid*.

Lemma 2.6. *For distinct non-trivial blocks B and B' in a level-1 network N, the designated hybrids η_B and $\eta_{B'}$ are distinct.*

Proof. From lemma 2.4 we get that if we have some network N with non-trivial blocks B and B' and some vertex $v \in V(N)$ such that $v = \eta_B = \eta_{B'}$, then it must also be the case that $v = \rho_B$ or $v = \rho_{B'}$. But since $\eta_B \neq \rho_B$ and $\eta_{B'} \neq \rho_{B'}$ this is impossible. So we can't have distinct non-trivial blocks B and B' in N that share the same vertex as their designated hybrids. □

Since every non-trivial block in a level-1 network must have a designated hybrid and two blocks can not share the same designated hybrid we get the following corollary.

Corollary 2.7. *The number of non-trivial blocks B in a level-1 network $N = (V, E)$ will be bound by $|V|$.*

We define the *inner vertices* B^0 of a block B in a level-1 network as $B^0 = V(B) \setminus \{\rho_B, \eta_B\}$.

Isomorphism and labeled networks

Two networks $N = (V, E)$ and $N' = (V', E')$ with roots ρ and ρ' are *isomorphic*, written $N \simeq N'$, if there is a bijection $\phi : V \to V'$ such that:

- $\phi(\rho) = \rho'$.
- $(u, v) \in E$ if and only if $(\phi(u), \phi(v)) \in E'$.

A *labeled network* (N, t) is a network $N = (V, E)$ with a function $t : V \to \mathcal{L}$ for some label set $\mathcal{L}$. Two labeled networks (N, t) and (N', t') with roots ρ and ρ' are isomorphic, $(N, t) \simeq (N', t')$, if there is a bijection $\phi : V \to V'$ such that:

- $\phi(\rho) = \rho'$.
- $(u, v) \in E$ if and only if $(\phi(u), \phi(v)) \in E'$.
- $t(v) = t'(\phi(v))$ for all $v \in V$.

The notation $N \simeq_\phi N'$ means that N is isomorphic to N' and the function ϕ is a bijection that fulfills the criteria listed above. We then say that ϕ is an isomorphism.

2.2 The AHU algorithm

As was mentioned in the introduction, the AHU algorithm [3] takes two (possibly labeled) rooted trees as input and then outputs whether or not the trees are isomorphic. For two rooted trees T and T' the algorithm runs in $\mathcal{O}(n)$ where $n = |V(T)| = |V(T')|$ [5][7].

3 Isomorphism checking of level-1 networks in linear time.

This chapter contains two sections. In the first section a method for creating a labeled tree (T_N, t_N) from a level-1 network N is given such that $(T_N, t_N) \simeq (T_{N'}, t_{N'})$ if and only if $N \simeq N'$. The second section is devoted to proving that the method given in the first section can be implemented with linear runtime.

3.1 Untangling a level-1 network

To recall, the AHU algorithm for isomorphism testing takes two, possibly labeled, rooted trees as input. A rooted tree is a network with no hybrid-vertices. The goal of this section is to introduce a method to transform a level-1 network N into a labeled tree (T_N, t_N), such that for two networks N and N' we have that $(T_N, t_N) \simeq (T_{N'}, t_{N'})$ if and only if $N \simeq N'$. This will reduce the testing of whether two level-1 networks are isomorphic to testing whether two labeled trees are isomorphic, which can be done in linear time with the AHU algorithm. We also want the size of $V(T_N)$ to be in $\mathcal{O}(V(N))$ to preserve linear run time with regards to the size of the network N.
We will use a label set $\mathcal{L} = \{s, p, h, \star\}$. The s label will represent the position of the source ρ_B in a block B, the p label will denote the position of a parent to the hybrid η_B in a block B, the label h denotes the position of the hybrid η_B and the label $\star$ is given to the original vertices in the network.

For every block B in the network N, let $child_B$ denote the children of ρ_B that are in B and do the following:

- Step 1 - Expand ρ_B:
  - Remove the edges (ρ_B, c) for all $c \in child_B$.
  - Add a vertex w_B with label $t_B(w_B) = s$.
  - Add the edges (w_B, c) for all $c \in child_B$.
  - Add the edge (ρ_B, w_B).
- Step 2 - Move η_B:
  - Remove the edges (u, η_B) and collect all such u in the set P_B.
  - Add a new vertex v_B^h with label $t_B(v_B^h) = h$ and the edges (w_B, v_B^h) and (v_B^h, η_B).

- Step 3 - Mark parents to hybrid:
  - For every $u \in P_B$, add a new vertex v_B^u with label $t_B(v_B^u) = p$ and a new edge (u, v_B^u).
- Step 4 - For every vertex v in B, if v have not received a label, assign $t_B(v) = \star$.

The network obtained from B, with the application of Step i together with the respective previous steps will be denoted (T_B^i), where $i \in \{1, 2, 3, 4\}$. We also define $T_{B-\rho}^i := T_B^i[V(B) \setminus \{\rho_B\}]$, that is T_B^i without the root ρ_B. We say that $T_B := T_B^4$ and $T_{B-\rho} := T_{B-\rho}^4$.

The network obtained by applying Step 1-4 to every non-trivial block B in a level-1 network N is called T_N. We define the labeling function $t_N(v) = t_B(v)$ if v is in some subtree T_B, and otherwise $t_N(v) = \star$.

Figures 3.1 - 3.4 contain a visual example showing how a non-trivial block is changed by these steps.

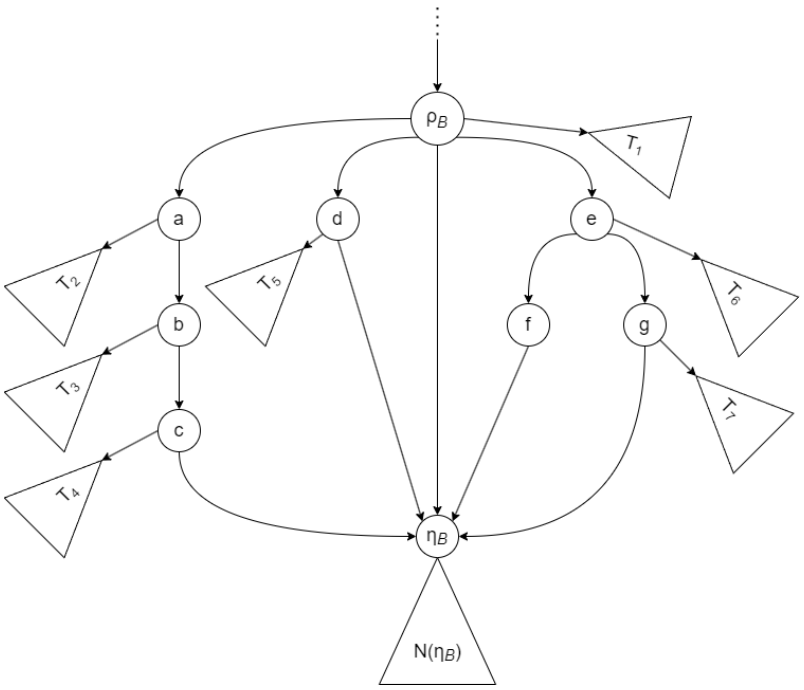


Figure 3.1: A zoomed in picture showing part of a level-1 network N with a non-trivial block B. The vertices that are part of B are marked as circles and the remaining parts of the network are abstracted as sub networks T_i.

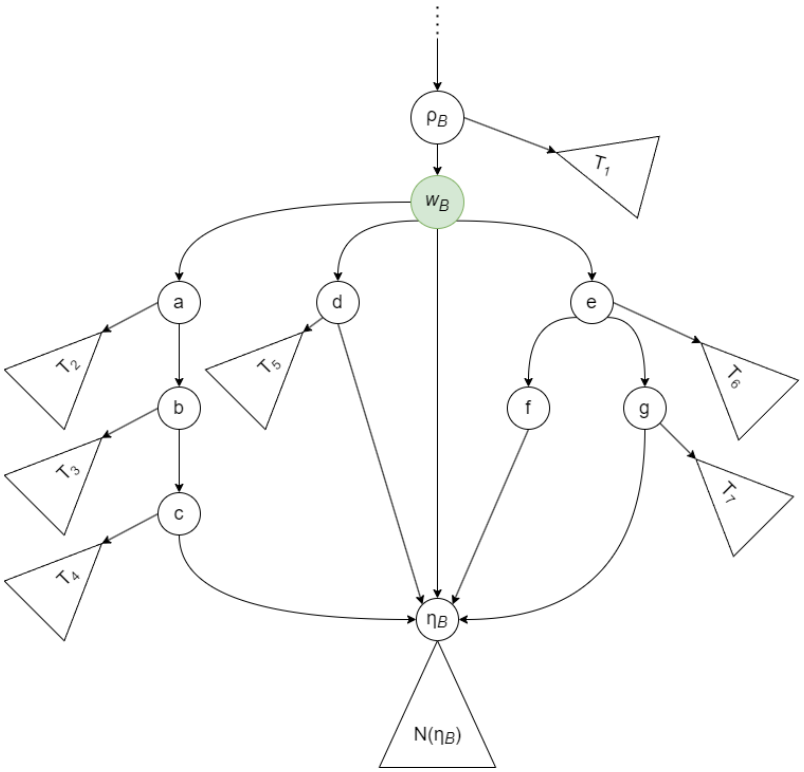


Figure 3.2: The network T_B^1 obtained by applying Step 1 to B. The green color denotes the label s

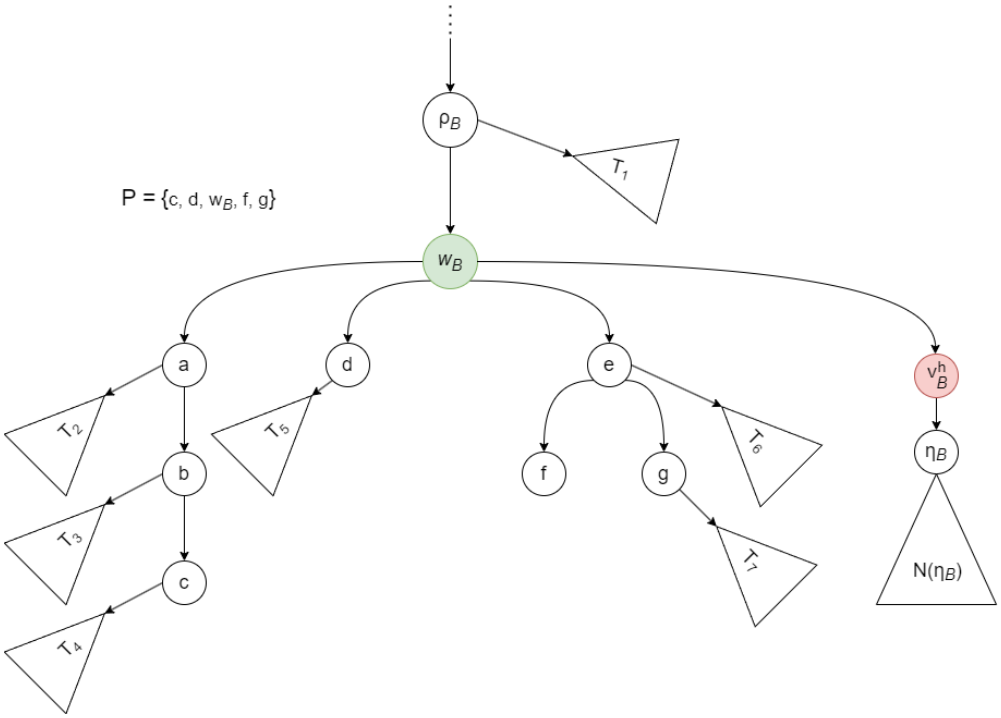


Figure 3.3: The network T_B^2 obtained by applying Step 2 to T_B^1. The red color denotes the label h and the set P is being tracked.

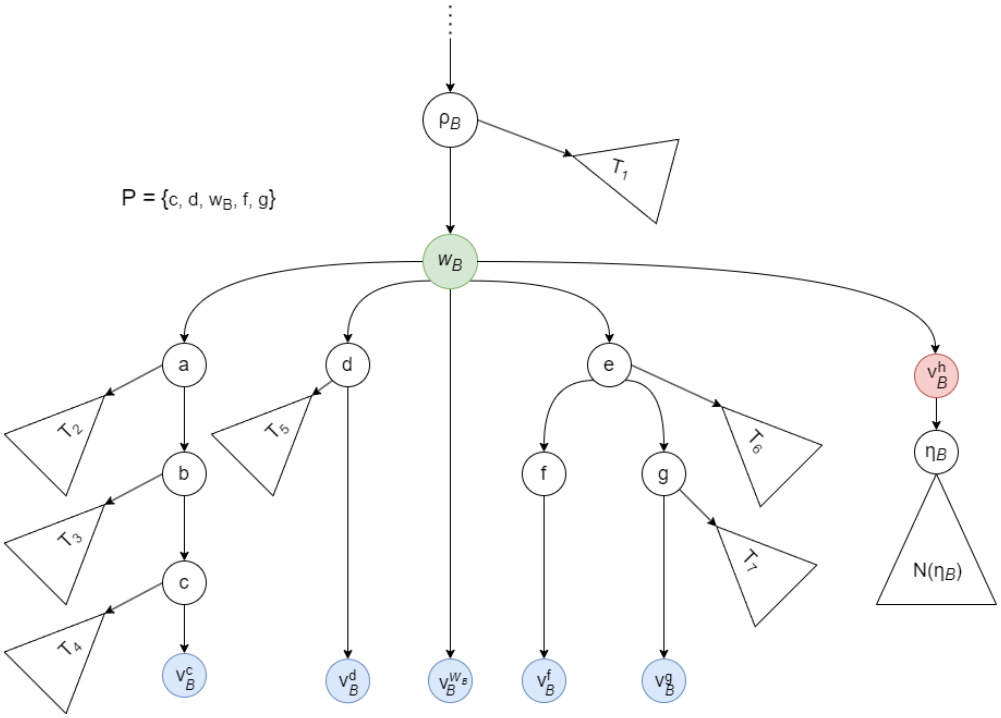


Figure 3.4: The network T_B^3 obtained by applying Step 3 to T_B^2. The blue color denotes the label p.

The rest of this section will be devoted to showing that for two level-1 networks N and N', we have that $N \simeq N'$ if and only if $(T_N, t_N) \simeq (T_{N'}, t_{N'})$.

Lemma 3.1. $(T_B, t_B) \simeq (T_{B'}, t_{B'})$ if and only if $(T_{B-\rho}, t_B) \simeq (T_{B'-\rho}, t_{B'})$.

Proof. For any isomorphism ϕ such that $(T_B, t_B) \simeq_\phi (T_{B'}, t_{B'})$ it must be the case that $\phi(w_B) = w_{B'}$ since $t_B(w_B) = s = t_{B'}(w_{B'})$ and no other vertex in either tree has label s. So $\phi(\rho_B) = \rho_{B'}$ since ρ_B is the only parent of w_B. Removing ρ_B from T_B and $\phi(\rho_B)$ from $T_{B'}$ creates two new isomorphic graphs and these are exactly the graphs $(T_{B-\rho}, t_B)$ and $(T_{B'-\rho}, t_{B'})$. Likewise if $(T_{B-\rho}, t_B) \simeq_\phi (T_{B'-\rho}, t_{B'})$ then $\phi(w_B) = w_{B'}$, so adding a parent vertex (with label $\star$) to w_B in $T_{B-\rho}$ and to $\phi(w_B)$ in $T_{B'-\rho}$ will create the two isomorphic trees (T_B, t_B) and $(T_{B'}, t_{B'})$. □

Lemma 3.2. For every non-trivial block B in N, $B \simeq T^1_{B-\rho}$.

Proof. Let us define a function $\phi : V(B) \to V(T^1_{B-\rho})$ such that $\phi(\rho_B) = w_B$ and for all $v \in V$, if $v \neq \rho_B$ then $\phi(v) = v$. Applying Step 1 to B makes it so that there is an edge $(w_B, v) = (\phi(\rho_B), \phi(v))$ in $T^1_{B-\rho}$ if and only if there is an edge (ρ_B, v) in B. For all $u, v \in V(T^1_{B-\rho}) \setminus \{w_B\}$ the edges (u, v) are unchanged after applying Step 1, so it must be the case that $B \simeq_\phi T^1_{B-\rho}$. □

By looking at Step 1-4 above it's clear that none of the inner vertices of B are removed and none of the edges between inner vertices are changed. So we have the following observation.

Observation 3.3. For every non-trivial block B in N and its inner vertices $u, v \in B^0$, we have that $(u, v) \in E(B)$ if and only if $(u, v) \in E(T_B)$ if and only if $(u, v) \in E(T_{B-\rho})$.

Lemma 3.4. *Let N be a level-1 network and B be a non-trivial block in N. Then both (T_B, t_B) and $(T_{B-\rho}, t_B)$ are labeled trees.*

Proof. Let B be a non-trivial block in the level-1 network N. Then B is a connected rooted DAG with root ρ_B. After step 1, T^1_B is still a connected rooted DAG with unique root ρ_B and thus T^1_B is a network. In step 2, all edges (u, η_B) are removed and the vertices u are saved in a set P_B. By [13] Lemma 7, all the parents of η_B are located in B so for all edges (u, η_B), all u are in B. The only edges added in Step 2 are (w_B, v^h_B) and (v^h_B, η_B), since these are the only edges in T^2_B where v^h_B and η_B are incident no cycles are introduced. We also have that there's a path $\rho_B \rightsquigarrow u$ for every u in P_B, and a path $\rho_B \rightsquigarrow \eta_B$, so all of T^2_B is connected. So T^2_B remains acyclic and is a network with root ρ_B. In step 3, a vertex v_u is introduced for every vertex $u \in P_B$. An edge (u, v_u) is then introduced for every vertex v_u, making sure the graph is still connected. Since every vertex v_u is a leaf, these edges do not introduce any cycles. So T^3_B is still a connected DAG with ρ_B as a unique root and is therefore a network.

The indegree of all the original vertices in B distinct from η_B remain unchanged, and all the added vertices w_B, v^h_B and v^u_B for every $u \in P_B$ have indegree 1. So the only vertex that can have an indegree greater than 1 in B is η_B. But η_B has only one edge

where it's the head, (v_B^h, η_B), so it has indegree 1. So there are no hybrid-vertices in T_B^3 and it is therefore a tree.

In step 4, any vertex v in $V(T_B^3)$ that has not already received a label is given the label $t_B(v) = \star$, so all v in $V(T_B^3)$ have a label in $\mathcal{L}$ and t_B is therefore a valid labeling function. No changes are made to the vertices or edges in step 4 so T_B^4 is still a tree and (T_B, t_B) is a labeled tree.

We have that w_B is the only child of ρ_B and ρ_B is the only parent of w_B in T_B. So removing ρ_B from T_B will make it so that w_B is the only vertex with indegree zero in the graph. So $T_B[V(T_B) \setminus \{\rho\}]$ will be a rooted tree with root w_B. Since $V(T_{B-\rho}) \subset V(T_B)$ and $t_B(v)$ is defined for all $v \in V(T_B)$ we have that $(T_{B-\rho}, t_B)$ also is a labeled tree. □

Lemma 3.5. *For distinct non-trivial blocks B and B' in a level-1 network N, the trees $T_{B-\rho}$ and $T'_{B-\rho}$ are vertex disjoint.*

Proof. Let B and B' be distinct non-trivial blocks in the network N. If B and B' were vertex-disjoint, then every vertex in T_B was either a vertex in B or it got added to T_B during it's construction, and any vertex added during construction is not added into any other trees $T_{B''}$, in particular not into $T_{B'}$. So T_B and $T_{B'}$ will be vertex disjoint if B and B' are vertex disjoint. Since $V(T_{B-\rho}) \subset V(T_B)$ and $V(T_{B'-\rho}) \subset V(T_{B'})$, $T_{B-\rho}$ and $T_{B'-\rho}$ will also be vertex-disjoint if B and B' are vertex-disjoint.

We have from lemma 2.4 that if there is a vertex v such that v is in B and v is in B', then $v = \rho_B$ or $v = \rho_{B'}$. From this it follows that they can't share two or more vertices, cause then ρ_B would be in B' and $\rho_{B'}$ would be in B and $\rho_B \neq \rho_{B'}$, so we would have non-trivial paths $\rho_B \rightsquigarrow \rho_{B'}$ and $\rho_{B'} \rightsquigarrow \rho_B$ so N would not be a network since it has a cycle. So if B and B' are not vertex-disjoint, we can without loss of generality assume that they share a single vertex ρ_B.

By construction ρ_B is not in $T_{B-\rho}$. As in the case when B and B' were vertex-disjoint, every vertex in $T_{B-\rho}$ was either already in B (and therefore not in B' since the blocks only shared ρ_B and ρ_B is not in $T_{B-\rho}$) or it got added to $T_{B-\rho}$ during construction and is therefore not in $T_{B'-\rho}$. Likewise every vertex in $T_{B'}$ was either in B' (and therefore not in $T_{B-\rho}$ since the the blocks only share ρ_B which is not in $T_{B-\rho}$), or it got added to $T_{B'-\rho}$ during construction and is therefore not in $T_{B-\rho}$. So for distinct non-trivial blocks B and B', the trees $T_{B-\rho}$ and $T_{B'-\rho}$ are vertex disjoint. □

Lemma 3.6. *For different non-trivial blocks B and B' in a level-1 network N, if there is a vertex v such that $v \in V(T_B)$ and $v \in V(T_{B'})$, then $t_B(v) = t_B(v') = \star$.*

Proof. For a level-1 network N with distinct non-trivial blocks B and B' we have that $T_{B-\rho}$ and $T_{B'-\rho}$ are vertex disjoint. So if there is a vertex v such that $v \in V(T_B)$ and $v \in V(T_{B'})$ then $v = \rho_B$ or $v = \rho_{B'}$. So we have either $v \in B$ or $v \in B'$, both of which imply that $v \in V(N)$. For a non-trivial block $\mathcal{B}$ in N and a vertex $u \in V(T_{\mathcal{B}})$, we can see from step 1-4 that $t_{\mathcal{B}}(u) \in \mathcal{L} \setminus \{\star\}$ only if $u \notin V(N)$, since the labels s, p, h are

only assigned to vertices that get added to the tree $T_{\mathcal{B}}$ during construction. So since $v \in V(N)$, we have that $t_B(v) = t_{B'}(v) = \star$. □

Lemma 3.7. *For a level-1 network N, (T_N, t_N) is a labeled tree.*

Proof. To recall, a labeled tree (T, t) is a connected DAG T with no hybrids and a unique root ρ, and a labeling function t such that $t(v)$ is defined for all v in $V(T)$.

From lemma 3.6 we have that for blocks B and B' in N, if $t_B(v)$ and $t_{B'}(v)$ are both defined for a vertex v, then $t_B(v) = t_{B'}(v)$. Every vertex $u \in V(N)$ is either in one more non-trivial blocks B, and therefore has a label $t_N(u) = t_B(u)$, or it is not in any non-trivial block and therefore gets assigned $t_N(u) = \star$, so t_N is a valid labeling function since each vertex u gets assigned exactly one label $t_N(u)$.

From [13] lemma 11 we have that for a hybrid vertex η and a block B in a network N, if η and one of it's parents are contained in B, then all of it's parents are contained in B and $\eta \neq \rho_B$. From this it follows that every hybrid-vertex in a level-1 network N is the designated hybrid η_B of some non-trivial block B. Since every non-trivial block B is replaced by a tree (hybrid-free network) T_B, there are no hybrids in T_N.

We have from lemma 2.1 that for every vertex $v \in V(N)$, there is a path $P^v = \rho \rightsquigarrow v$ in N. For every non-trivial block B in N, the path P^v either traverses no edges contained in B or it includes a subpath contained in B. If P^v traverses no edges contained in B, then the path P^v is unchanged when you replace B with T_B in N.

If the path P^v contained a subpath S_B^v contained in B, then S_B^v is either of the form $\rho_B \rightsquigarrow \eta_B$ or $\rho_B \rightsquigarrow u$ for some $u \in B^0$. In every tree T_B, there is by construction a path $\rho_B \rightsquigarrow \eta_B = \rho_B, v_B^h, \eta_B$, so if the path S_B^v was of the form $\rho_B \rightsquigarrow \eta_B$, then there will be a path $P_B^v = \rho \rightsquigarrow v$ in N after you replace B with T_B, where P_B^v is the path obtained by replacing S_B^v in P^v with the subpath ρ_B, v_B^h, η_B.

If S_B^v is of the form $\rho_B \rightsquigarrow u$ for some $u \in B^0$, then S_B will start along an edge (ρ_B, c) for some child of ρ_B in B and the remaining edges that S_B follow will be of the form (u_i, u_j) where $u_i, u_j \in B^0$. Since all the edges (u_i, u_j) remain unchanged in T_B, we only need to show that there is still a path $\rho_B \rightsquigarrow c$ in T_B to show that there will be a path $\rho \rightsquigarrow v$ in N after replacing B with T_B. By construction there will be a path $\rho_B \rightsquigarrow c$ in T_B along the edges $(\rho_B, w_B), (w_B, c)$. So there will be a path $P_B^v = \rho \rightsquigarrow v$ in N after replacing B with T_B, where P_B^v is the path obtained by replacing the first two vertices ρ_B, c in the subpath S_B^v with the sequence ρ_B, w_B, c.

So changing any non-trivial blocks B for the tree T_B in N does not affect the existence of a path $\rho \rightsquigarrow v$ for any vertex v in $V(N)$. So if there is a path $\rho \rightsquigarrow v$ in N, then there will be a path $\rho \rightsquigarrow v$ in T_N. And since N was a level-1 network, it had a path $p_v = \rho \rightsquigarrow v$ for every $v \in V(N)$, and therefore T_N will also have a path p_v for every $v \in V(N)$.

Now for the vertices $u \in V(T_N) \setminus V(N)$. Each of these vertices are in some subgraph

T_B. Since there is a path $P_1 = \rho \rightsquigarrow \rho_B$ for each T_B in T_N, we just need to note that T_B is a connected network with unique source ρ_B to know that there will be a path $P_2 = \rho_B \rightsquigarrow u$, and therefore appending the paths P_1, P_2 will yield a path $\rho \rightsquigarrow u$ in T_N. So T_N will have a path $\rho \rightsquigarrow v$ for every $v \in V(T_N)$, and therefore T_N is connected.

Since T_N is connected with a unique root, every vertex except the root ρ has indegree at least 1, and since T_B has no hybrids, those vertices have indegree exactly 1, so there are $|V(T_N)| - 1$ edges in T_N. The number of edges compared to the number of vertices together with the fact that T_N is connected implies that T_N is a tree. So (T_N, t_N) is labeled tree. □

Lemma 3.8. *Let N be a level-1 network and B and B' be non-trivial blocks in N. If $B \simeq B'$, then $(T_B, t_B) \simeq (T_{B'}, t_{B'})$.*

Proof. Let us assume that we have a level-1 network N with non-trivial blocks B and B', such that $B \simeq_\phi B'$ for some function ϕ. It must then be the case that $\phi(\rho_B) = \rho_{B'}$ since these are the only vertices with indegree equal to 0 in their respective blocks. It must also be the case that $\phi(\eta_B) = \eta_{B'}$ since these are the only vertices with more than one parent in their respective blocks.

Let us extend ϕ to $\phi^1 : V(T_B^1) \to V(T_{B'}^1)$ by defining $\phi^1(v) = \phi(v)$ for $v \in V(B)$ and $\phi^1(w_B) = w_{B'}$. It should be clear that ϕ^1 is a bijective function from $V(T_B^1)$ to $V(T_{B'}^1)$. Note that all of the following is true:

- There is an edge (w_B, c) in T_B^1 if and only if there is an edge (ρ_B, c) in B.
- There is an edge $(\phi(\rho_B), \phi(c))$ in B' if and only if there is an edge (ρ_B, c) in B.
- There is an edge $(w_{B'}, \phi(c))$ in $T_{B'}^1$ if and only if there is an edge $(\phi(\rho_{B'}), \phi(c))$ in B'.

Therefore, there is an edge (w_B, c) in T_B^1 if and only if there is an edge $(\phi^1(w_B), \phi^1(c))$ in $T_{B'}^1$. We also know that the only edge to which ρ_B is incident in T_B^1 is the edge (ρ_B, w_B) and the only edge to which $\phi^1(\rho_B) = \rho_{B'}$ is incident in $T_{B'}^1$ is the edge $(\rho_{B'}, w_{B'}) = (\phi^1(\rho_B), \phi^1(w_B))$. It's also the case that w_B has no other parent in T_B^1 and $w_{B'}$ has no other parent in $T_{B'}^1$.

For vertices $u, v \in V(B) \setminus \{\rho_B\}$, any edges (u, v) are unchanged in T_B^1 and the same is true for non-root vertices and their incident edges in B'. This means that there is an edge (w_1, w_2) in T_B^1 if and only if there is an edge $(\phi^1(w_1), \phi^1(w_2))$ in $T_{B'}^1$, so $T_B^1 \simeq_{\phi^1} T_{B'}^1$.

Now let us extend ϕ^1 to $\phi^2 : V(T_B^2) \to V(T_{B'}^2)$ in the following way. For $v \in V(T_B^1)$ we define $\phi^2(v) = \phi^1(v)$, and $\phi^2(v_B^h) = v_{B'}^h$. This way ϕ^2 is defined for all vertices in T_B^2 and is a bijection from $V(T_B^2)$ to $V(T_{B'}^2)$.
Note that all of the following is true:

- An edge in T_B^1 is removed when creating T_B^2 if and only if it's on the form (u, η_B) for some $u \in V(T_B^1)$.
- There is an edge (u, η_B) in $V(T_B^1)$ if and only if there is an edge $(\phi^1(u), \phi^1(\eta_B)) = (\phi^1(u), \eta_{B'})$ in $V(T_{B'}^1)$.
- An edge in T_B^1 is removed when creating T_B^2 if and only if it's on the form $(v, \eta_{B'})$ for some $v \in V(T_B^1)$.

Therefore, an edge (u, η_B) for some $u \in V(T_B^1)$ is removed from T_B^1 when creating T_B^2 if and only if the edge $(\phi^2(u), \phi^2(\eta_B))$ is removed from $T_{B'}^1$ when creating $T_{B'}^2$.

We know that the only edges added to T_B^1 when constructing T_B^2 are the edges (w_B, v_B^h) and (v_B^h, η_B). Likewise the only edges added to $T_{B'}^1$ when constructing T_B^2 are the edges $(w_{B'}, v_{B'}^h) = (\phi^2(w_B), \phi^2(v_{B'}^h))$ and $(v_{B'}^h, \eta_{B'}) = (\phi^2(v_B^h), \phi^2(\eta_B))$.

The edges mentioned above are the only edges added and removed when constructing T_B^2 from T_B^1 and $T_{B'}^2$ from $T_{B'}^1$. So we have that there is an edge (u, v) in T_B^2 if and only if there is an edge $(\phi^2(u), \phi^2(v))$ in $T_{B'}^2$ and therefore $T_B^2 \simeq_{\phi^2} T_{B'}^2$.

Let us now once again extend ϕ^2 to $\phi^3 : V(T_B^3) \to V(T_{B'}^3)$. We noted above that for $u \in V(T_B^1)$ there is an edge (u, η_B) in $V(T_B^1)$ if and only if there is an edge $(\phi^1(u), \phi^1(\eta_B))$ in $V(T_{B'}^1)$, so $u \in P_B$ if and only if $\phi^1(u) \in P_{B'}$. We define ϕ^3 as follows:

- For $v \in V(T_B^2)$ we make it so $\phi^3(v) = \phi^2(v)$.
- Every vertex v in T_B^3 added during step 3 has exactly one parent p_v. We make it so that $\phi^3(v) = u$ where u is the vertex added during step 3 of the construction of $T_{B'}^3$ that has $\phi^2(p_v)$ as its parent.

Since there is a vertex $u \in P_B$ if and only if $\phi^1(u) \in P_{B'}$, we know that for every vertex $v \in P_B$ both v and $\phi^3(v)$ will have a child added to them during step 3 of the construction of their respective trees, and these are the only vertices added during step 3, so ϕ^3 is a well-defined bijection from $V(T_B^3)$ to $V(T_{B'}^3)$. It's also clear that an edge (u, v) is added to T_B^3 during step 3 if and only if a corresponding edge $(\phi^3(u), \phi^3(v))$ is added in $T_{B'}^3$. So it must be the case that $T_B^3 \simeq_{\phi^3} T_{B'}^3$.

No changes are made to the edges or vertices of T_B^3 and $T_{B'}^3$ during step 4, so $T_B \simeq_{\phi^3} T_{B'}$. Now note that for all $v \in V(B)$, $t_B(v) = \star$, for all vertices w_B added during step 1 we have $t_B(w_B) = s$, for all vertices v_B^h added during step 2 we have $t_B(v_B^h) = h$, and finally for all vertices v_B^u added during step 3 we have $t_B(v_B^u) = p$.
This gives us that $(T_B^1, t_B) \simeq_{\phi^1} (T_{B'}^1, t_{B'})$, $(T_B^2, t_B) \simeq_{\phi^2} (T_{B'}^2, t_{B'})$ and $(T_B^3, t_B) \simeq_{\phi^3} (T_{B'}^3, t_{B'})$. So $(T_B, t_B) \simeq_{\phi^3} (T_{B'}, t_{B'})$ □

Lemma 3.9. *Let N be a level-1 and B and B' be non-trivial blocks in N. If $(T_B, t_B) \simeq (T_{B'}, t_{B'})$, then $B \simeq B'$.*

Proof. Let N be a level-1 network with non-trivial blocks B and B' and assume that $(T_B, t_B) \simeq (T_{B'}, t_{B'})$. By lemma 3.1 we have that $(T_{B-\rho}, t_B) \simeq_\phi (T_{B'-\rho}, t_{B'})$ for some function ϕ. Since there is a bijection ϕ from $V(T_{B-\rho})$ to $V(T_{B'-\rho})$, there are an equal number of vertices in the two trees, and since it's a labeled isomorphism, there are an equal number of vertices with each label in each tree. Let us perform the following procedure on both trees.

In labeled tree $(T_{\mathcal{B}-\rho}, t_{\mathcal{B}})$ do the following:

- Step I: remove p labeled vertices:
  - For every vertex $v \in V(T_{\mathcal{B}-\rho})$ such that $t_{\mathcal{B}}(v) = p$, put it's parent u into a set $Q_{\mathcal{B}}$.
  - For every $u, v \in V(T_{\mathcal{B}-\rho})$ remove any edges (u, v) such that $t_{\mathcal{B}}(v) = p$, and remove those vertices v.
- Step II: Move $\eta_{\mathcal{B}}$:
  - Remove the edges $(v^h_{\mathcal{B}}, \eta_{\mathcal{B}})$ and $(w_{\mathcal{B}}, v^h_{\mathcal{B}})$, and remove the vertex $v^h_{\mathcal{B}}$.
  - For every vertex in $u \in Q_{\mathcal{B}}$, add an edge $(u, \eta_{\mathcal{B}})$.

The network obtained from $(T_{\mathcal{B}-\rho}, t_{\mathcal{B}})$ with the application of step (i) together with respective previous steps will be denoted B_i where $i \in \{I, II\}$.

We note that $v \in Q_B$ if and only if $\phi(v) \in Q_{B'}$, since an edge (v, u) such that $t_B(u) = p$ is in $E(T_{B-\rho})$ if and only if $(\phi(v), \phi(u)) \in E(T_{B'-\rho})$.

Let $V = V(T_{B-\rho})$ and $V' = V(T_{B'-\rho})$, then $B_I = T_{B-\rho}[V \setminus Q_B]$ and $B'_I = T_{B'-\rho}[V' \setminus Q'_B]$.

So we know that $T_{B-\rho} \simeq_\phi T_{B'-\rho}$ and a vertex $v \in V(T_{B-\rho})$ is in Q_B if and only if $\phi(v) \in Q_{B'}$. This gives us that a vertex v is removed from $T_{B-\rho}$ in B_I if and only if $\phi(v)$ is removed from $T_{B'-\rho}$ in B_I, and all edges to which u is incident are removed in B_I if and only if all edges to which $\phi(u)$ is incident are removed in B_I. So we have that $B_I \simeq_\psi B_I$ where $\psi : V(B_I) \to V(B'_I)$ and for every $v \in V(B_I)$ we have $\psi(v) = \phi(v)$.

Next we note that $T_{B-\rho}$ and $T_{B'-\rho}$ both only have one vertex with label h and one with label s, and these vertices are also in B_I and B'_I. So it must be the case that $\psi(w_B) = w_{B'}$ and $\psi(v^h_B) = v^h_{B'}$. We also note that the vertices η_B and $\eta_{B'}$ both have label $\star$ and are therefore not removed from their respective trees during step I. Due to the construction of $T_{B-\rho}$ there are exactly two edges where v^h_B is incident, those are $(w_B v^h_B)$ and (v^h_B, η_B). Similarly $v^h_{B'}$ is incident in the edges $(w_{B'}, v^h_{B'})$ and $(v^h_{B'}, \eta_{B'})$. These edges are also left intact in B_I and B'_I. So a vertex $v \in V(T_{B-\rho})$ or an edge $(u_1, u_2) \in E(T_{B-\rho})$ is removed from B_I when constructing B_{II} if and only if the vertex $\psi(v)$ or the edge $(\psi(u_1), \psi(u_2))$ is removed in B'_I when constructing B'_{II}.

We noted above that a vertex $v \in Q_B$ if and only if $\phi(v) \in Q_{B'}$. We should also note that during the construction of $T_{B-\rho}$ all vertices with label p are added as leaves, so

no vertex added to Q_B is then also removed during step I. So the edge (v, η_B) will be added in B_{II} if and only if the edge $(\psi(v), \psi(\eta_B) = \eta_{B'})$ is added in B_{II}.

No vertices get added to either B_{II} or B'_{II} during step II.

Combining what have now shown we get that $B_I \simeq_\psi B_I$, and vertices v and edges (u_1, u_2) gets added or removed to B_{II} if and only if the vertices $\psi(v)$ and edges $(\psi(u_1), \psi(u_2))$ gets added or removed in B_{II}. So we have that $B_{II} \simeq B_{II}$.

Now note that in constructing B_{II} we have undone step 3 and step 2 from the procedure of constructing $T_{B-\rho}$ from B, so $B_{II} = T^1_{B-\rho}$. Also note that from lemma 3.2 we have $T^1_{B-\rho} \simeq B$ for any non-trivial block B. So since $B_{II} \simeq B'_{II}$, this means that $B \simeq B'$. We have shown that if $(T_B, t_B) \simeq (T_{B'}, t_{B'})$, then $B \simeq B'$. □

Theorem 3.10. *Let N be a level-1 network and B and B' be non-trivial blocks in N. Then $B \simeq B'$ if and only if $(T_B, t_B) \simeq (T_{B'}, t_{B'})$.*

Proof. This follows directly from the combination of lemmas 3.8 and 3.9. □

Lemma 3.11. *Let N and N' be level-1 networks. If $N \simeq N'$, then $(T_N, t_N) \simeq (T_{N'}, t_{N'})$.*

Proof. Let N and N' be level-1 networks and assume that $N \simeq_\phi N'$ for some function ϕ. For every non-trivial block B in N there is a corresponding non-trivial block B' in N' such that $\phi(\rho_B) = \rho_{B'}$ and $\phi(\eta_B) = \eta_{B'}$.

Let us perform steps 1-4 on B in N and call the new network N_B. Do the same on B' in N' and call the new network $N'_{B'}$. We know that no edges in N that weren't properly contained in B got changed when replacing B with T_B through steps 1-4, so for any vertices $u, v \in V(N)$ such that $u, v \notin V(B)$ we know that $(u, v) \in E(N_B)$ if and only if $(\phi(u), \phi(v)) \in E(N'_{B'})$. We also know from theorem 3.10 that $T_B \simeq_\psi T'_B$ for some function ψ, so for $u, v \in V(T_B)$ we know that $(u, v) \in E(N_B)$ if and only if $(\psi(u), \psi(v)) \in E(N'_{B'})$.

In the proof of lemma 3.8 we can see that if $B \simeq_\chi B'$, then $T_B \simeq_{\chi^3} T_{B'}$ where χ^3 is such that if $v \in V(B)$ then $\chi^3(v) = \chi(v)$. So we can without loss of generality say that our function ψ is such that for $v \in V(B)$, we have that $\psi(v) = \phi(v)$. So for vertices $u, v \in V(N)$ we have that $(u, v) \in E(N_B)$ if and only if $(\phi(u), \phi(v)) \in E(N'_{B'})$.

Let us now extend the function ϕ to $\phi^* : V(N_B) \to V(N'_{B'})$ such that for $v \in V(N)$ we have $\phi^*(v) = \phi(v)$ and for $v \in (V(T_B) \setminus V(N))$ we have $\phi^*(v) = \psi(v)$. We then have that for any vertex $v \in V(N_B)$ such that both $\psi(v)$ and $\phi(v)$ are defined, $\psi(u) = \phi(u) = \phi^*(u)$.

For vertices $u, v \in V(N)$ we have that $(u, v) \in E(N_B)$ if and only if $(\phi^*(u), \phi^*(v)) \in E(N'_{B'})$.

Now note that for any edge $(u,v) \in E(N_B)$ or $(v,u) \in E(N_B)$ such that $u,v \in V(N_B)$ but $u \notin V(N)$, we have that all such vertices u got added to T_B during it's construction. For a vertex u added to T_B during it's construction, we can see in steps 1-4 that any edges u is incident to are properly contained in T_B. So for such edges we have that $(u,v) \in E(N_B)$ if and only if $(\psi(u) = \phi^*(u), \psi(v) = \phi^*(v)) \in E(N'_{B'})$. So for all vertices $u,v \in V(N_B)$ we have that $(u,v) \in E(N_B)$ if and only if $(\phi^*(u), \phi^*(v)) \in E(N'_{B'})$. So $N_B \simeq_{\phi^*} N'_{B'}$. If we also define a labeling function $temp_B$ such that for $v \in V(T_B)$ we have $temp_B(v) = t_B(v)$ and for all other vertices u we have $t_B(u) = \star$, we can note that $(N_B, temp_B) \simeq_{\phi^*} (N'_{B'}, temp_{B'})$ since we know that $(T_B, t_B) \simeq_{\psi} (T_{B'}, t_B)$ and $\psi(v) = \phi^*(v)$ for $v \in V(T_B)$.

We now have level-1 networks N_B and $N'_{B'}$ such that $(N_B, temp_B) \simeq (N'_{B'}, temp_{B'})$ and N_B has one fewer non-trivial blocks than N. Repeating this process by applying step 1-4 on a non-trivial block C in N_B and it's corresponding block C' in $N'_{B'}$ will then once again yield isomorphic labeled level-1 networks that we can call N_C and $N'_{C'}$. If we now instead define $temp_C$ such that for $v \in V(T_B)$ we have $temp_C(v) = t_B(v)$, for $v \in V(T_C)$ we have $temp_C(v) = t_C(v)$, and for all other vertices u we have $temp_C(u) = \star$, then we also have $(N_C, temp_C) \simeq (N'_{C'}, temp_{C'})$.

If we repeat this process until there are no more non-trivial blocks in the resulting networks and we extend the *temp* labeling functions with the labels of all previously replaced blocks, then when the final non-trivial block D is processed like this we will get $(N_D, temp_D) = (T_N, t_N)$. It will also be the case that $(N_D, temp_D) \simeq (N_{D'}, temp_{D'})$, so $(T_N, t_N) \simeq (T_{N'}, t_{N'})$.
We have now showed that if $N \simeq N'$, then $(T_N, t_N) \simeq (T_{N'}, t_{N'})$. □

Lemma 3.12. *Let N and N' be level-1 networks. If $(T_N, t_N) \simeq (T_{N'}, t_{N'})$, then $N \simeq N'$.*

Proof. Let N and N' be level-1 networks and assume that $(T_N, t_N) \simeq_{\phi} (T_{N'}, t_{N'})$ for some bijection ϕ. Then for every subtree T_B in T_N there is a subtree $T_{B'}$ in $T_{N'}$ such that $\phi(w_B) = w_{B'}$ and $T_B \simeq_{\phi} T_{B'}$ when ϕ is restricted to the vertices in $V(T_B)$. From lemma 3.10 we have that if $T_B \simeq T_{B'}$ then $B \simeq B'$. If we replace the subtree T_B in T_N with the block B and call this new network T_N^B, and replace the subtree $T_{B'}$ in $T_{N'}$ with B' and call this new network $T_{N'}^{B'}$. Then we have that $(T_N^B, t_N) \simeq_{\phi} (T_{N'}^{B'}, t_{N'})$ when ϕ is restricted to $V(T_N^B)$. The proof for this follows below.

We first note that $V(T_N^B) \subseteq V(T_N)$, so $\phi : V(T_N^B) \to V(T_{N'}^{B'})$ is still a well defined bijection. Next we have from observation 3.3 that for $u,v \in B^0$, the edge $(u,v) \in E(T_B)$ if and only if $(u,v) \in E(B)$. Since $T_B \simeq_{\phi} T_{B'}$ and $B \simeq B'$, this means that for $u,v \in B^0$, $(u,v) \in E(T_N^B)$ if and only if $(\phi(u), \phi(v)) \in E(T_{N'}^{B'})$. For vertices $u,v \notin V(B)$, the edge $(u,v) \in E(T_N^B)$ if and only if $(\phi(u), \phi(v)) \in E(T_{N'}^{B'})$ since $T_N \simeq_{\phi} T_{N'}$ and any edges not contained in T_B (or $T_{B'}$) remain unchained when T_B got replaced B (and $T_{B'}$ by B'). So now we just need to make sure that $(u,v) \in E(T_N^B)$ if and only if $(\phi(u), \phi(v)) \in E(T_{N'}^{B'})$ when either u or v is in $V(B)$ but not in B^0. This means we have one of the following cases:

- $u = \rho_B$, $v = \eta_B$. There is an edge $(\rho_B, \eta_B) \in E(T_N^B)$ if and only if there was an edge $(\rho_B, v_B^{\eta_B})$ in T_B. Since ϕ is a bijection that preserves labels and w_B only has one parent in T_B it must be that $\phi(w_B) = w_{B'}$ and $\phi(\rho_B) = \rho_{B'}$. It must also be the case that $\phi(\eta_B) = \eta_{B'}$ since v_B^h only has one child in T_B, and finally w_B can have at most one child with label p in T_B so $\phi(v_B^{\eta_B}) = v_{B'}^{\eta_{B'}}$. So there's an edge $(\phi(\rho_B), \phi(\eta_B)) \in E(T_{N'}^{B'})$ if and only if there was an edge $(\phi(\rho_B), \phi(v_B^{\eta_B}))$ in $T_{B'}$. And since $T_B \simeq_\phi T_{B'}$ this means that there's an edge $(\rho_B, \eta_B) \in E(T_N^B)$ if and only if there is an edge $(\phi(\rho_B), \phi(\eta_B)) \in E(T_{N'}^{B'})$.

- $u = \rho_B$, $v \in B^0$. In this case there is an edge $(\rho_B, v) \in E(T_N^B)$ if and only if there was an edge (w_B, v) in T_B. Since $T_B \simeq_\phi T_{B'}$ there was an edge (w_B, v) in T_B if and only if there was an edge $(\phi(w_B), \phi(v))$ in $T_{B'}$. We noted above that $\phi(w_B) = w_{B'}$ and $\phi(\rho_B) = \rho_{B'}$. So there was an edge $(\phi(w_B), \phi(v))$ in $T_{B'}$ if and only if there was an edge $(\phi(\rho_B), \phi(v))$ in B'. This gives us that there is an edge $(\rho_B, v) \in E(T_N^B)$ if and only if there is an edge $(\phi(\rho_B), \phi(v)) \in E(T_{N'}^{B'})$.

- $u \notin B$, $v = \rho_B$. In this case there was an edge $(u, \rho_B) \in E(T_N)$ if and only if there was an edge $(\phi(u), \phi(\rho_B)) \in E(T_{N'})$ and no such edges got added or removed when replacing T_B with B (or when replacing $T_{B'}$ with B'). So $(u, \rho_B) \in E(T_N^B)$ if and only $(\phi(u), \phi(\rho_B)) \in E(T_{N'}^{B'})$.

- $u = \rho_B$, $v \notin B$. In this case there was an edge $(\rho_B, v) \in E(T_N)$ if and only if there was an edge $(\phi(\rho_B), \phi(v)) \in E(T_{N'})$ and no such edges were added or removed when replacing T_B with B (or when replacing $T_{B'}$ with B'). So $(\rho_B, v) \in E(T_N^B)$ if and only if $(\phi(\rho_B), \phi(v)) \in E(T_{N'}^{B'})$.

- $u \in B^0$, $v = \eta_B$. In this case there is an edge $(u, \eta_B) \in E(T_N^B)$ if and only if u had child c with label $t_B(c) = h$ in T_B. And $T_B \simeq_\phi T_{B'}$, there was an edge $(u, c) \in E(T_B)$ if and only if there was an edge $(\phi(u), \phi(c)) \in E(T_{B'})$, and since ϕ preserves labels it must be that $t_{B'}(\phi(c)) = h$. So there is an edge $(\phi(u), \phi(c)) \in E(T_{B'})$ if and only there is an edge $(\phi(u), \phi(\eta_B)) \in E(T_{N'}^{B'})$. So we have that there is an edge $(u, \eta_B) \in E(T_N^B)$ if and only if there is an edge $(\phi(u), \phi(\eta_B)) \in E(T_{N'}^{B'})$.

- The case where $u \notin B$, $v = \eta_B$ is not possible, see [13] lemma 7.

Combining all of the cases above we can see that for $u, v \in V(T_N^B)$, there is an edge $(u, v) \in E(T_N^B)$ if and only if there is an edge $(\phi(u), \phi(v)) \in E(T_{N'}^{B'})$, so $T_N^B \simeq_\phi T_{N'}^{B'}$. Since no vertices got added when constructing T_N^B from T_N (or $T_{N'}^{B'}$ from T_N') we also have $(T_N^B, t_N) \simeq (T_{N'}^{B'}, t_{N'})$. This procedure can then be repeated by replacing another subtree T_{B^*} in T_N^B and the corresponding subtree $T_{B'^*}$ in $T_{N'}^{B'}$ until all such subtrees have been replaced, at which point we have the networks N and N'. So we have shown that if $(T_N, t_N) \simeq (T_{N'}, t_{N'})$, then $N \simeq N'$. □

Theorem 3.13. *Let N and N' be level-1 networks. Then $N \simeq N'$ if and only if $(T_N, t_N) \simeq (T_{N'}, t_{N'})$.*

Proof. This follows from the combination of lemma 3.11 and lemma 3.12. □

3.2 Time complexity of solution

Here comes first a proof that the size of the output trees are linearly bounded by the size of the input networks. Then follows a proof that the algorithm can be implemented such that it can be performed in linear time with regard to the size of the input graph.

Theorem 3.14. *For a level-1 network N, $|V(T_N)| \in \mathcal{O}(|V(N)|)$.*

Proof. For a non-trivial block B in N, vertices get added to a tree T_B in 3 cases:

- The vertex w_B gets added between ρ_B and all its children in B.
- The vertex v_B^h gets added between w_B and η_B.
- A vertex v_B^u gets added as a leaf beneath every vertex that was a parent of η_B.

Let us assume the worst case scenario for the all three cases.

From the lemma 2.4 we get that for every vertex $v \in V$ there can at most be one non-trivial block B such that $v \in B$ but $v \neq \rho_B$. So every vertex $v \in V$ can be the child of at most one designated root ρ_B in N. So the first case adds a maximum of $|V|$ vertices.

The second case adds one vertex per non-trivial block in N. From lemma 2.7 we have that the number of non-trivial blocks in N is bound by $|V|$, so the second case adds a maximum of $|V|$ vertices.

For the third case we once again consider the fact that for every vertex $v \in V(N)$ there is at most one non-trivial block B such that $v \in V(B)$ but $v \neq \rho_B$. So for any vertex v in V, that vertex can be parent to at most one designated hybrid η_B in non-trivial blocks B where v is not the designated root ρ_B. So there can be a maximum of $|V|$ edges in N of the form (u, η_B) where B is a non-trivial block in N and $u \neq \rho_B$. Now we consider the remaining edges for this case, those that are of the form (ρ_B, η_B) for some non-trivial block B in N. We have from lemma 2.6 that distinct non-trivial blocks can't share designated hybrids, so every hybrid-vertex η_B can be incident to at most one edge of the form (ρ_B, η_B) for some designated root ρ_B. So the number of edges in N of the form (ρ_B, η_B) for some non-trivial block B in N is also bound by $|V|$. So the third case will add a maximum of $2 \cdot |V|$ vertices.

So $|V(T_N)| \leq |V(N)| + |V(N)| + |V(N)| + 2 \cdot |V(N)|$ which is in $\mathcal{O}(|V(N)|)$. □

Now to prove that the construction can be performed in linear time. Let's consider the following implementation `Algorithm 1`.

Lemma 3.15. *When given a level-1 network N as input,* `Algorithm 1` *correctly returns the labeled tree (T_N, t_N)*

Proof. Let N be a level-1 network given to `Algorithm 1` as input.

Algorithm 1 Transforming a level-1 network N to a labeled tree (T_N, t_N).

Input: Level-1 network $N = (V, E)$
Output: Labeled tree (T_N, t_N)

Find all biconnected components of N using DFS and name each non-trivial biconnected component with a number i starting at 0. For each vertex v mark it so that $v.blocks$ is a list containing the numbers for all non-trivial blocks that v is part of.
Create a table *parents* such that for every vertex v we have that $parents[v]$ is a list containing all vertices that have v as a child.
Mark all vertices v in N such that $v.label = \star$
Create an empty array ws and an empty array *root* both with length equal to the number of non-trivial blocks in N.
Create a set of vertices V^* that is a copy of V.
for $v \in V$ in DFS order starting at ρ **do**
    for $B \in v.blocks$ **do**
        if $roots[B]$ is not empty **then** ▷ ρ_B has been visited
            if $parents[v]$ contains $roots[B]$ **then**
                Remove $roots[B]$ from $parents[v]$
                Add $ws[B]$ to $parents[v]$
            end if
            if Size of $parents[v]$ is > 1 **then** ▷ v is the hybrid of B, η_B
                for all u in $parents[v]$ **do**
                    Create a vertex v_B^u and add v_B^u to V^*.
                    Set $parents[v_B^u] \leftarrow [u]$ and $v_B^u.label \leftarrow p$.
                end for
                Create a new vertex v_B^h and add v_B^h to V^*.
                Set $parents[v_B^h] \leftarrow [ws[B]]$ and $w_B^h.label \leftarrow h$.
                Set $parents[v] \leftarrow [v_B^h]$
            end if
        end if
        if $roots[B]$ is empty **then** ▷ v is ρ_B
            Create a new vertex w_B and add w_B to V^*.
            Set $parents[w_B] \leftarrow [v]$ and $w_B.label \leftarrow s$.
            Set $ws[B] \leftarrow w_B$
            Set $roots[B] \leftarrow v$
        end if
    end for
end for
Create a network T_N such that $V(T_N)$ equals V^* and (u, v) is and edge in $E(T_N)$ if and only if u is in $parents[v]$.
Define t_N as a function such that $t_N(v) = v.label$ for each vertex v in T_N
return (T_N, t_N)

Step 1 requires that for every non-trivial block B, we add a vertex w_B with label $t_N(w_B) = s$ and ρ_B as its only parent. This is handled on lines 23-28 of the algorithm. Since there are no paths from ρ to a vertex v in B that does not include ρ_B, it must be the case that the first vertex in B visited with DFS starting at ρ is the vertex ρ_B. We also have that $roots[B]$ is empty when visiting a vertex v in B if and only if v is the first vertex visited in B, since $roots[B]$ gets assigned v on line 25 if it was empty and can't be changed or emptied again later. So the if-block starting at line 21 is entered exactly once per non-trivial block B when visiting ρ_B. In this if-block the vertex w_B is added and it's label set on line 24, and the parent of w_B is set to ρ_B on line 25.

Step 1 also requires that for every vertex v in B that is a child of ρ_B, we remove the edge (ρ_B, v) and add the edge (w_B, v). This is handled on lines 9-12. Since ρ_B is the first vertex to be visited in B, $roots[B]$ will contain ρ_B by the time any vertex in B that is a child of ρ_B gets visited. So any vertex v in B will have the edge (ρ_B, v) removed at line 10 and the edge (w_B, v) added at line 11, if and only if it had ρ_B as it's parent.

So Step 1 is handled correctly.

In Step 2 and Step 3 a new vertex v_B^h with label $t_N(v_B^h) = h$ is added together with the edges (w_B, v_B^h) and (v_B^h, η_B) and all other edges where η_B is the head of the edge are removed. This is handled on lines 18-20. Note once again that by the time η_B is visited, ρ_B must already have been visited, so the if-statement at line 8 will be true, and if the currently visited vertex is η_B it will have a number of parents greater than 1 and this will still be true after lines 9-12. So the if statement on line 13 will be true if and only if the currently visited vertex is η_B.

Step 2 and Step 3 also requires for any vertex u in a non-trivial block B that if u is a parent of η_B then a vertex v_B^u with label $t_N(v_B^u) = p$ and edge (u, v_B^u) are added. This is handled on lines 13-17. As mentioned above the if-statement on line 13 will be true if and only if the currently visited vertex is η_B, and for each parent u of η_B a vertex v_B^u with label p and edge (u, v_B^u) is added on lines 15-16.

So step 2 and 3 is also handled correctly.

And since all vertices added are given a label, and all vertices that were already in N at the start are given the label $\star$ at line 3, we have that Step 4 is also handled correctly.

So the algorithm performs all changes required for Step 1-4 and these are the only changes the algorithm makes. So `Algorithm 1` correctly outputs (T_N, t_N) when given a level-1 network N as input. □

Theorem 3.16. *For a level-1 network $N = (V, E)$,* `Algorithm 1` *runs in $O(|V|+|E|)$.*

Proof. Let $N = (V, E)$ be a level-1 network.

Finding all biconnected components in a graph $N = (V, E)$ can be done in $\mathcal{O}(|V|+|E|)$ with the help of DFS [14, p. 621-622]. So line 1 can be performed in linear time.
A parent table can also be setup in linear time by storing some extra information while performing DFS. So line 2 can be performed in linear time by storing some extra information during the DFS for line 1.
Line 3 can obviously be done in $|V|$ steps and can therefore be done in linear time.
Line 4 can be done in linear time since the number of non-trivial blocks in N is bound by $|V|$.
Line 5 can be done by simply iterating over all vertices in V, so it can be done in $|V|$ steps and therefore done in linear time.
Line 6-7 will iterate over each non-trivial block that each vertex is part of. From lemma 3.5 it follows that a vertex can only be part of more than one non-trivial block if it's the unique root of at least all but one of those non-trivial blocks, and the number of non-trivial blocks in N in bound by $|V|$ from corollary 2.7. Therefore there are at most $|V|$ pairs (u, B) where B is a non-trivial block and $u \in V(B)$ but $u \neq \rho_B$. There will also be at most $|V|$ pairs (ρ_B, B) where B is a non-trivial block. So a maximum of $2 \cdot |V|$ iterations will be started. For all code inside these loops, the following paragraphs show that they will run in amortized linear time.
Lines 23-28 can be performed in constant time, and will only be performed once per non-trivial block B in N. We have from corollary 2.7 that hte number of non-trivial blocks B is bound by $|V|$, so lines 23-28 will contribute at most $\mathcal{O}(|V|)$ to the runtime.
Lines 9-12 can be run in time proportional to the number of parents of the evaluated vertex v, since $roots[B]$ needs to be found and removed in $parents[v]$. Lines 8-22 will not be performed for a vertex v and non-trivial block B where v is the root ρ_B of B. We also noted above that no vertex v will be part of more than one non-trivial block B where $v \neq \rho_B$. So no parent list will be iterated over more than once for this part of the algorithm. So lines 9-12 can be done in $\mathcal{O}(|E|)$ steps.
Finally lines 13-21 will be performed once per hybrid-vertex η_B. Lines 15-16 can be performed in constant time, and since two distinct non-trivial blocks can't share their unique hybrid, no parent list will be iterated over more than once for this part of the algorithm. So lines 14-17 will contribute at most $\mathcal{O}(|E|)$ to the runtime. Lines 18-20 can be done in constant time and will be performed once per non-trivial block, and can therefore be done in $\mathcal{O}(|V|)$ steps.
So lines 6-30 can be performed in $\mathcal{O}(|V| + |E|)$ steps.
Since we showed in theorem 3.14 that the size of T_N is linear with regards to N, line 31 can be performed in linear time.
Line 32 and 33 can performed in constant time.
So with a level-1 networks $N = (V, E)$ as input, the run time of `Algorithm 1` is in $\mathcal{O}(|V| + |E|)$. □

4 Conclusion

The goal of this thesis was to present a practical algorithm for checking isomorphism of level-1 networks in linear time. This was achieved by the method presented at the start of section 3.1 to transform a level-1 network N into a labeled tree (T_N, t_N) wich was proved by theorem 3.13 to be such that $N \simeq N'$ if and only if $(T_N, t_N) \simeq (T_{N'}, t_{N'})$. It was then also showed in section 3.2 that the labeled tree (T_N, t_N) can be constructed from N in linear time and that the size of T_N is linear with regard to the size of N. Since the AHU algorithm can check for isomorphism of labeled rooted trees in linear time, one can check the isomorphism of level-1 networks by constructing their corresponding labeled trees and then use those as input for the AHU algorithm, all of which can be done in linear time.

One question that remains is how this algorithm performs when properly implemented. The implementation Algorithm 1 in section 3.2 is just pseudo code to prove that the method can be implemented in linear time, so empirical tests on how the method performs when properly implemented, especially compared to the performance of existing methods, would be a good starting point for further investigations.

Bibliography

[1] L. Chen, "Graph isomorphism and identification matrices: Sequential algorithms," *Journal of Computer and System Sciences*, vol. 59, no. 3, pp. 450–475, 1999.

[2] S. Fortin, "The graph isomorphism problem," Tech. Rep. TR96-20, The University of Alberta, 1996. `https://doi.org/10.7939/R3SX64C5K`.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, Mass.: Addison-Wesley, 1974.

[4] J. E. Hopcroft and J. K. Wong, "Linear time algorithm for isomorphism of planar graphs (preliminary report)," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, (New York, NY, USA), p. 172–184, Association for Computing Machinery, 1974.

[5] A. Lindeberg, "Isomorphism testing of rooted trees in linear time," *arXiv preprint arXiv:2401.07636*, 2024.

[6] D. M. Campbell and D. Radford, "Tree isomorphism algorithms: Speed vs. clarity," *Mathematics Magazine*, vol. 64, no. 4, pp. 252–261, 1991.

[7] F. Ingels, "Revisiting tree isomorphism: Ahu algorithm with primes numbers," *arXiv preprint arXiv:2309.14441*, 2023.

[8] P. Gambette, V. Berry, and C. Paul, "The structure of level-k phylogenetic networks," in *Combinatorial Pattern Matching* (G. Kucherov and E. Ukkonen, eds.), (Berlin, Heidelberg), pp. 289–300, Springer Berlin Heidelberg, 2009.

[9] C. Choy, J. Jansson, K. Sadakane, and W.-K. Sung, "Computing the maximum agreement of phylogenetic networks," *Theoretical Computer Science*, vol. 335, no. 1, pp. 93–107, 2005. Pattern Discovery in the Post Genome.

[10] V. Moulton and T. Wu, "Planar rooted phylogenetic networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 2, pp. 1289–1297, 2022.

[11] L. Knüver, M. Fischer, M. Hellmuth, and K. Wicke, "The weighted total cophenetic index: A novel balance index for phylogenetic networks," 2024.

[12] J. Kukluk, L. Holder, and D. Cook, "Algorithm and experiments in testing planar graphs for isomorphism," *J. Graph Algorithms Appl.*, vol. 8, pp. 313–356, 01 2004.

[13] M. Hellmuth, D. Schaller, and P. F. Stadler, "Clustering systems of phylogenetic networks," *Theory in Biosciences*, vol. 142, no. 4, pp. 301–358, 2023.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

Datalogi
www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm