mathematics

MDPI

Article

Constructing an Evolutionary Tree and Path–Cycle Graph Evolution along It

Konstantin Gorbunov * and Vassily Lyubetsky †

Institute for Information Transmission Problems of the Russian Academy of Sciences, 127051 Moscow, Russia; lyubetsk@iitp.ru

* Correspondence: gorbunov@iitp.ru; Tel.: +7-910-439-0597

† Partial support of RFBR grant 20-01-00670 acknowledged.

Abstract: The paper solves the problem of constructing an evolutionary tree and the evolution of structures along it. This problem has long been posed and extensively researched; it is formulated and discussed below. As a result, we construct an exact cubic-time algorithm which outputs a tree with the minimum cost of embedding into it and of embedding it into a given network (Theorem 1). We construct an algorithm that outputs a minimum embedding of a tree into a network, taking into account incomplete linear sorting; the algorithm depends linearly on the number of nodes in the network and is exact if the sorting cost is not less than the sum of the duplication cost and the loss cost (Theorem 3). We construct an exact approximately quadratic-time algorithm which, for arbitrary costs of *SCJ* operations, solves the problem of reconstruction of given structures on any two-star tree (Theorem 4). We construct an exact algorithm which reduced the problem of *DCJ* reconstruction of given structures on any star to a logarithmic-length sequence of SAT problems, each of them being of approximately quadratic size (Theorem 5). The theorems have rigorous and complete proofs of correctness and complexity of the algorithms, and are accompanied by numerical examples and numerous explanatory illustrations, including flowcharts.

Keywords: exact algorithm; low computation complexity algorithm; discrete optimization; discrete evolution; tree reconciliation; path-cycle graph reconstruction; minimum embedding of a tree into a network

MSC: 05C15; 05C40; 05C70; 05C85; 05C90

Citation: Gorbunov, K.; Lyubetsky, V. Constructing an Evolutionary Tree and Path–Cycle Graph Evolution along It. *Mathematics* **2023**, *11*, 2024. https://doi.org/10.3390/math11092024

Academic Editor: Andrea Scozzari

Received: 29 March 2023
Revised: 19 April 2023
Accepted: 21 April 2023
Published: 24 April 2023

1. General Introduction

Tree graphs arise very widely, both in fundamental problems of discrete mathematics and in applied problems from very different subject areas; for particular examples of their use, see [1,2]. Such a problem is the classification problem and well-known methods for constructing graph trees, such as the ID3 algorithm; see, e.g., [3,4]. Note the applications of random trees and random forests in [5].

In the present paper, we consider a specific problem of constructing the evolution of structures along a tree or network G; G is understood as the discrete time in which the evolution progresses. This problem naturally decomposes into two parts, the construction of this G (Theorems 1–3) and the construction along it of the evolution of X, itself from the given X_0 at the leaves of the tree (Theorems 4 and 5). The construction of G also requires some kind of a priori data; in particular, G is constructed as a tree/network that is optimally coordinated with already known trees or networks P and S so that there are optimal embeddings $P \to G \to S$. This problem setting is well known, and hundreds of papers, surveys, and books are devoted to various versions of it, of which we note the survey ([6] Section 5) and the book [7]. The present paper is not a survey and contains proofs of the four theorems mentioned above. Algorithms for constructing the discrete

evolution of structures along a tree are often heuristic. In our purely mathematical study, all algorithms have complete and rigorous proofs of their correctness and of low estimates of their computational complexity.

Evolution analysis is one of the most fundamental mathematical problems, having, at the same time, a broad practical importance. In this respect, we have considered *discrete evolution*, where a rooted tree acts as time. This understanding of time is particularly important in connection with biological evolution, although, in this paper, evolution is considered purely mathematically, without any relation to various applications. We consider two phases of reconstruction to necessarily following each other, first, a tree acting as time is constructed, and then evolution itself is sought for along the constructed tree. Namely, the first problem consists of constructing a tree as the "average" between a given tree and a given network, which reduces to minimizing a given functional defined on all trees of a fixed size. In other words, the first problem involves creating a new tree that represents an "average" between a tree and a network (in particular, also a tree). This is completed by finding a tree that minimizes a given functional, which assigns a value to each possible tree of a fixed size. The goal is to find the tree minimizing this value that will be the best representation of the "average" between these two given graphs. The second problem consists of describing the evolution of structures defined at leaves of a given tree. Evolution is understood as the continuation of these structures onto internal nodes of the tree, where the continuation is the minimum of a given functional defined on all arrangements along the tree. An *arrangement* is a one-to-one mapping of structures to internal nodes of the tree, together with already fixed structures at its leaves. In other words, the second problem aims to explain how structures that are defined at the leaves of a tree evolve over time by continuing onto internal nodes of the tree. The continuation of these structures is determined by finding the arrangement that minimizes a given functional, which is defined for all possible arrangements along the tree. Essentially, the goal is to find the best possible way for the structures to evolve along the tree. In addition to the functional, the notion of a structure must be defined. In this paper, a *structure* is a loaded directed graph consisting of paths and cycles (a path–cycle graph). Of course, both problems can be considered in the framework of other formulations as well.

In Section 2, "Tree Construction," we prove Theorem 1, where we construct an exact cubic-time algorithm which, given a binary tree P, binary tree or network S, and a collection B of sets, outputs a binary tree G minimizing the sum of costs of minimum continuations that preserve the natural ordering of leaf mappings from P to G and from G to S, provided that G contains only clades that belong to B. Here, a *clade* means the set of all leaves below any given node $g \in G$. A continuation preserving the natural ordering, $P \to G$ or $G \to S$, is called an *embedding*; thus, here we minimize the sum of costs of two minimum embeddings. Our mathematical results (Theorems 1–5), which develop the problem of discrete optimization, are used by us, in particular, in applied biological works to construct the evolution of genomes and genomic rearrangement. For example, the algorithm from Theorem 1 allows us to trace the evolution of proteins with respect to the genes that encode them, and, at the same time, the evolution of these genes with respect to the species that contain them. The example described in Section 2.6 (and in our applied works) show that this algorithm significantly improves the intermediate tree G compared to what the algorithm in [8] produces for precisely the same data. That algorithm is based on building the optimal gene tree by applying modifications on the input tree P guided by the tree S. Thus, the algorithm in [8] is based on descending to the local minimum of a functional starting from a tree G equal to the given tree P. The algorithm in [8] is quite different from ours and, moreover, is heuristic; the latter also makes it drastically different from our algorithm. In [8], no proofs or considerations about the exactness of that algorithm are given, but only a quadratic estimate of its runtime.

In Section 2, we also construct a minimum embedding of a tree into a network taking into account duplication, loss, and Incomplete Linear Sorting events (Theorem 3). Transition to a network S instead of a tree S significantly extends the average tree problem in

mathematical and applied aspects. Previously, for this problem an algorithm in [9] was known, which took into account only one Incomplete Linear Sorting event.

In Section 3, "Reconstruction of Structures on a Tree," we prove two theorems. In Theorem 4, we construct an exact approximately quadratic-time algorithm which solves the *SCJ* reconstruction problem on any two-star tree for arbitrary costs of *SCJ* operations. In Theorem 5, we construct an exact algorithm which, on any star, for arbitrary costs of *DCJ* operations, reduces a cyclic equal-content *DCJ* reconstruction problem to a logarithmic-length sequence of *SAT* (Boolean satisfiability) problems, each of them being approximately quadratic-size. For example, algorithms from Theorems 4 and 5 make it possible to reconstruct genomes of ancestral organisms and genomic rearrangements. Previously, an exact *SCJ* reconstruction algorithm was developed only for equal cost operations in [10] or on a star-like tree in [11], and the *DCJ* reconstruction algorithm was developed as a reduction in the reconstruction problem to an integer linear programming problem in [12].

The rest of this article is organized as follows. Section 2 contains definitions related to the embedding of a tree into a network (Section 2.1); setting of the intermediate tree problem and the formulation of Theorem 1 (Section 2.2); the definition of embedding a tree into a network (Section 2.3); a description of the algorithm for constructing an intermediate tree and a Proof of Theorem 1 (Section 2.4); a description of the choice of the main parameter of the algorithm (Section 2.5); an example of the algorithm (Section 2.6); a description of the algorithm for constructing an embedding of a tree into a network, taking into account "Incomplete Linear Sorting", along with a Proof of Theorem 3 (Section 2.7); and, finally, an example of the operation of the algorithm (Section 2.8).

Section 3 contains a setting of the structure reconstruction problem and formulations of Theorems 4 and 5 (Section 3.1), a description of the *SCJ* reconstruction algorithm on a two-star tree (Section 3.2), a proof of the exactness of this algorithm and estimates of its runtime (Theorem 4, Section 3.3), an example of the operation of the *SCJ* reconstruction algorithm (Section 3.4), a description of the *SCJ* reconstruction algorithm for the cyclic case of the problem (Section 3.5), a description of the *DCJ* reconstruction algorithm on a star and a Proof of Theorem 5 (Section 3.6), an example of the *DCJ* reconstruction algorithm (Section 3.7), and a description of the choice of parameters for this algorithm (Section 3.8).

Section 4 (Discussion) briefly mentions one of the possible applications of our algorithms, as well as possible directions for further mathematical research related to this work. In general, the issue of applications of these algorithms requires descriptions of the corresponding subject areas and is far in style and content from the presented mathematical study.

2. Tree Construction

2.1. Introduction

We consider rooted trees; a root is entered by an edge which starts at a node of degree 1, referred to as a *super-root*; the edge itself will be called the *root edge*. We will regard the super-root of a tree, and then the root, as located "at the top," and leaves of a tree (all of degree 1) as located "at the bottom", as shown in Figure 1a. Similarly, *subtrees* are regarded with their root edge, which enters the root of the subtree and starts at its super-root, the latter being of degree 1. The *tail* of an edge e is its endpoint located closer to the root, and the second endpoint is the *head* of the edge; they are denoted by e_+ and e_-, respectively. An *ancestor* node/edge is located closer to the root, and a *descendant* node/edge is farther from the root, both lying on the same path; this relation is denoted by $\leq$. The nearest ancestor is called a *parent*, and the nearest descendant is called a *child*; nodes located immediately below the same node are said to be *sibling* nodes, each of them is called a *sister* of any other one. In essence, a tree specifies a parent–child relation. For any set N of leaves, a node which is the lowest among all ancestors of all leaves in N is well defined. Denote it by N'. For two trees T_1 and T_2, there is a well-known *canonical continuation* [13,14] of **any preassigned mapping** s from leaves in T_1 to leaves in T_2, which is denoted here by s''. Namely, for an interior node x in T_1 with a set $x_<$ of leaves lying below it, we let $s''(x) = s(x_<)'$. The set $x_<$

is called the *clade* of x, or the *clade* of the root edge entering x, or the *clade* of the subtree defined by x.

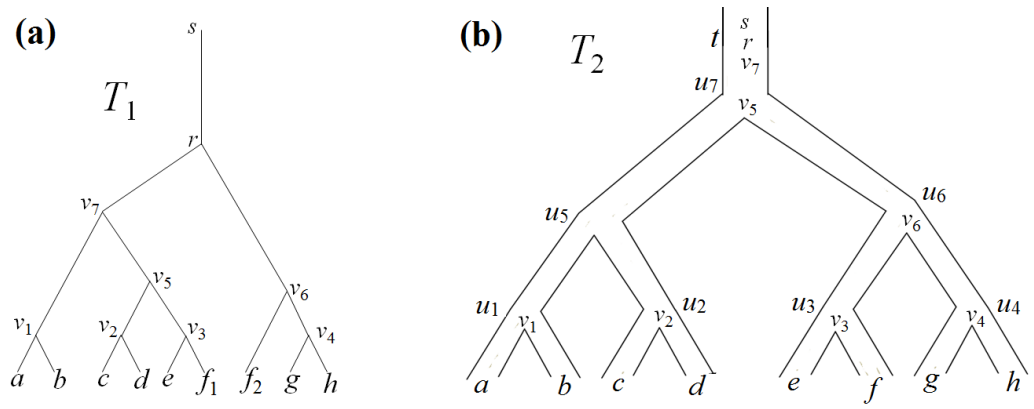

Figure 1. (**a**) All the considered trees are arranged "top to bottom"; in them, s is a super-root, r a root, and (s,r) a root edge. (**b**) The mapping s; $T_1 \to T_2$ is arbitrarily defined on leaves as a correspondence of labels in which indices are disregarded (i.e., f_1 and f_2 are mapped to f). The mapping s canonically continues onto interior nodes in T_1 as s': $T_1 \to T_2$. For instance, $s'(v_i) = u_i$ for $1 \le i \le 4$ or $i = 6$, $s'(v_5) = u_7$, and $s'(v_7) = s'(r) = s'(s) = t$. The nodes v_7, r, and s are mapped to the root edge. Images of nodes of T_1 are shown "inside" T_2, so that is why we refer to edges in T_2 as *pipes*.

If $s''(e_+) = s''(e_-)$, then $s'(e_+)$ is defined as the edge entering $s''(e_-)$ from above; for instance, s' maps the super-root in T_1 into the root edge in T_2. Thus, s' maps nodes in T_1 into nodes and edges in T_2; see Figure 1b. This definition can be generalized to the case where T_2 is a network (see Section 2.3). We will refer to the mapping s' as an *embedding* of T_1 into T_2. Below, when saying "a tree T_1 is mapped to a tree T_2," we always mean precisely this mapping s' and speak about its images and arguments. Note that in this case we have a preassigned mapping s, which is called a *leaf mapping*. Of course, the mapping s' preserves the natural order.

The *path* of an edge $e \in T_1$ in a tree T_2 is a path consisting of nodes in T_2 from $s'(e_+)$ to $s'(e_-)$ excluding its endpoints $s'(e_+)$ and $s'(e_-)$.

The *image tree* of T_1 in T_2 is defined to be a tree isomorphic to T_1 with nodes located at nodes and in edges of T_2 so that v is located either at the node $s'(v)$ or in the pipe $t = s'(v)$ (see Figure 2). In this sense, edges of T_2 will be referred to as *pipes* containing nodes and edges of T_1. Images are *connected* by an edge according to the parent–child relation of their arguments. Thus, we want to consider an *image tree* as the T_1 tree located inside T_2. Our use of mathematical interpretations in embedding is related to our inspiration from biological paradigms, such as the evolution of genomes and genomic rearrangement.

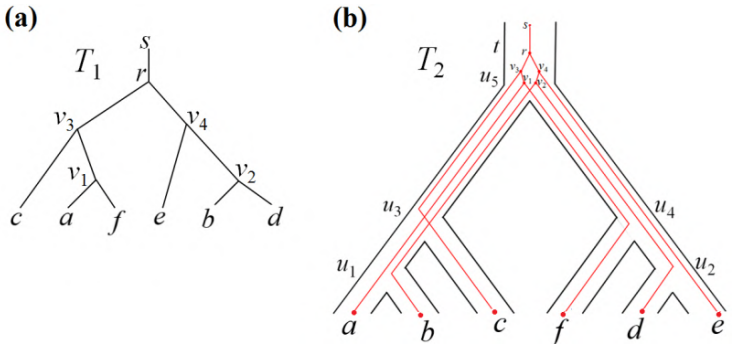

Figure 2. (**a**) Original tree T_1, and (**b**) its image tree in T_2; its nodes are shown in bold, and its edges are shown as polygonal curves. The images of v_1 and v_2 are located at the same node u_5, and the images of v_3 and v_4 are in the same pipe t. For example, for the edge $e' = (v_4,e)$, its path in T_2 is $u_5u_4u_2$.

If tree leaves are assigned unique names, we may not distinguish between a leaf and a name assigned to it. A tree with labelled leaves represents the discrete-time evolution of a matter which is assigned to the root of the tree; the evolution goes from the root to leaves through interior nodes of the tree. For example, this can be evolution of a gene, protein, species, etc., from the root along the whole given tree to its leaves, which represent up-to-date states of the initial (at the root) matter; such up-to-date states could already be actually specified.

Evolution analysis is one of the most fundamental mathematical problems, having at the same time a broad practical importance (see, e.g., [15–18]). Note, also, the work [19], which describes an algorithm for solving the problem of isomorphic tree embedding in a network and provides numerous references to articles related to this topic. A detailed overview of topics related to evolutionary trees and numerous references are given in [6]. Issues related to phylogenetic networks are discussed in the book [7].

Let T_0 be the set of all leaves of a tree or network T. The problem is as follows: given a tree P, a set M, a tree or network S, and mappings s_1: $P_0 \to M$ and s_2: $M \to S_0$, find a tree G (where $G_0 = M$) that reconciles the evolutions s_1' and s_2' in the following sense. We define *costs* $c(s')$ and $c(s_2')$ of the embeddings s_1': $P \to G$ and s_2': $G \to S$, respectively, and then minimize $f(G) = c(s_1') + c(s_2')$ with respect to the argument G (see Figure 3). For the case of a network S, the definition of the embedding cost of s_2': $G \to S$ is given in Section 2.7. Until that section, S will always be a tree, and the embedding costs will be considered according to Definitions 1 and 2 below. However, $f(G)$ can be defined to be an easily computable function with some properties that we will need below.

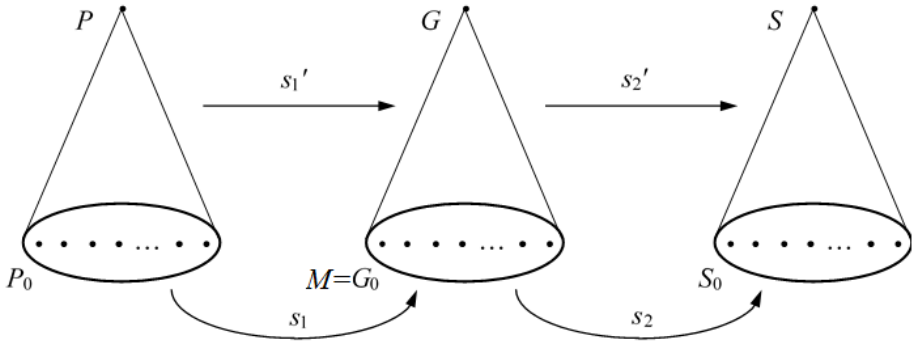

Figure 3. Mappings s_1 and s_2 on leaves and their continuations s_1' and s_2' onto interior nodes. We want to minimize $c(s_1') + c(s_2')$ with respect to the gene tree G and thus find a tree G^* on which $c(s_1') + c(s_2')$ attains its minimum value.

We are dealing here with a general mathematical problem, so all further material is presented independently of any applications. However, as a typical example, let us describe its content using biological terms. Let P be a tree whose leaves are assigned with protein names ("protein evolution tree", or, for short, protein tree), and let S be a network, in particular a tree, whose leaves are assigned with names of species containing these proteins ("species evolution network", or, for short, species network/tree). For more details concerning the notion of a network, see Section 2.3. At this moment, we may assume S to be a tree. Let M be a set of genes from which proteins in leaves of P are formed, and let a leaf mapping s_1: $P_0 \to M$ be fixed where $y = s_1(x)$ is the gene from which a protein x is formed (see Figure 3). On the other hand, a mapping s_2: $M \to S_0$ is fixed where $s_2(y)$ is the species to which a gene y belongs. The composition $s_2(s_1)$: $P_0 \to S_0$ defines a mapping which specifies a species to which a protein x belongs. s_1 and s_2 are specified extrinsically for applied reasons. Let G be a tree with leaf set $M = G_0$, which is called the "gene evolution tree", or gene tree, or an *intermediate* tree with respect to P and S. Given s_1 and s_2, their continuations s_1': $P \to G$ and s_2': $G \to S$ are defined. We regard s_1' as evolution of proteins relative to genes (in the gene tree), and s_2' as evolution of genes relative to species (in the

species tree). The problem is to minimize $c(s_1')+c(s_2')$ with respect to the gene tree G and, as a result, to find a tree G^* on which $c(s_1')+c(s_2')$ attains its minimum value.

2.2. Setting of the Intermediate Tree Problem and Formulation of Theorem 1

Let the degree of every node in the trees T_1 and T_2 except for the super-root and leaves be 3. All leaves are labeled with unique names, and M is the set of leaves together with their names in T_2.

The embedding s': $T_1 \to T_2$ for the trees T_1 and T_2 is defined above.

Definition 1. *A duplication in T_1 is a node of degree 3 which is mapped to a pipe. Informally, for example, the node $r \in T_1$ in Figure 4a has an image located in a pipe above the images of the children $v_7 \in T_1$ and $v_6 \in T_1$, which is interpreted as a furcation of the node r in this pipe into v_7 and v_6. A loss is a pair consisting of an edge $e \in T_1$ and a node $y \in T_2$ such that y belongs to the path of e in T_2 (the path does not contain the images $s'(e_+)$ and $s'(e_-)$, the extreme nodes of the path). Informally, an edge e passes through y and has no furcation in y. A loss is said to be implicit if a clade of one of the child pipes of y does not contain images of leaves from T_1, otherwise it is said to be explicit. We define the locus of a duplication to be its image and the locus of a loss to be the node y. Duplications and losses will be called evolutionary events. For example, all events for the T_1 tree shown in Figure 1 are presented in Figure 4. Let each event be assigned with its cost, a strictly positive rational number. The cost $c(s')$ of an embedding s' is the total cost of all its events. These definitions can be carried over to the case where S is a network (see Section 2.3).*

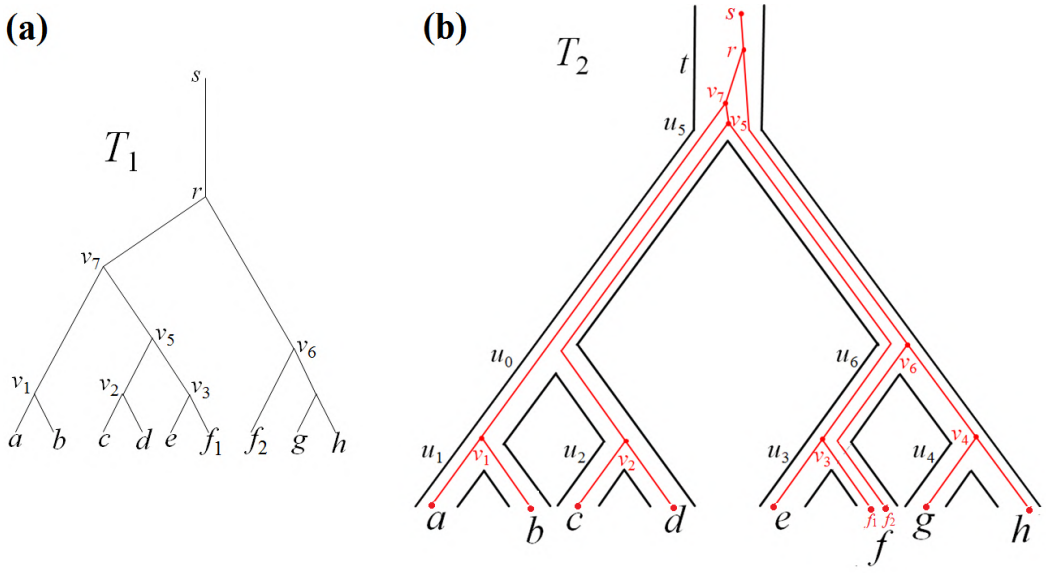

Figure 4. (**a**) Tree T_1 given in Figure 1a above. (**b**) Image tree isomorphic to T_1; marked in red inside T_2 (recall that its nodes are shown in bold and its edges are shown as polygonal curves). Here, two duplications, v_7 and r, are in the root pipe. Losses: in the edge (v_6,f_2) at node u_3; in the edge (v_7,v_1) at u_5 and u_0; in the edge (v_5,v_2) at u_0; in the edge (v_5,v_3) at u_6; and in the edge (r,v_6) at u_5. All losses are explicit. Interior nodes in T_2 are labeled by symbol u with an index. Note that the nodes r, v_3, and v_4 in Figure 2b are duplications.

The **intermediate tree problem** is as follows: given a tree P and a tree/network S together with leaf mappings s_1: $P_0 \to M$ and s_2: $M \to S$, find a tree G with leaf set $M = G_0$ for which the sum of the costs of the embeddings s_1': $P \to G$ and s_2': $G \to S$ is minimal. Under Definition 1, the problem consists of minimizing the total number of events that occur in these two evolutions taking into account their costs. This number $c(s_1')+c(s_2')$ is called the ***cost* of a tree *G***. The problem setting is not yet completed if S is a network; in this case we also need to define the continuation s_2': $G \to S$ of the mapping s_2. We define it in Section 2.3 below, where it is called the *minimum embedding*; in the case of a tree, there is

exactly one minimum embedding, i.e., the embedding s' defined in Section 2.1. The main feature of a minimum embedding in the case of a network is the fact that it is not defined uniquely; there exist many minimum embeddings.

The problem for the case of a tree S was posed in [8], where it was proved that the problem is NP-hard. The latter means that it cannot be solved by any polynomial algorithm if P $\neq$ NP. However, a polynomial algorithm or even a low-complexity algorithm can be found if we introduce some additional conditions into the problem. In [8], the problem for a tree was solved by a heuristic (quadratic-complexity) algorithm *given* that the mapping s_1 is injective; moreover, that algorithm did not assume the costs of events. Instead, we suggest *another condition*.

(*) The tree G contains only clades from a fixed-in-advance collection B of non-empty subsets of M such that B contains M itself and all its one-element subsets, and any non-one-element set A in B has a *partition* in B.

The latter means that A can be partitioned into two non-empty disjoint parts from B; in particular, B does not contain the empty set. In Section 2.5, we explain how one can construct a collection B satisfying condition (*).

Under this condition, we obtain an exact (cubic-complexity) algorithm solving this problem for both a tree and a network S. Even for the case of a tree, our algorithm is other than that in [8]. Additionally, for the case of a network we propose an algorithm finding one particular minimum embedding which is **denoted** by ${s_2}'\colon G \to S$.

Thus, below, we prove the following.

Theorem 1. *An exact cubic-time (in the input data size) algorithm is construct which, given a binary tree P, binary tree S, and a collection B of subsets of M, outputs a binary tree G minimizing the sum of costs of embeddings of P to G and of G to S provided that G contains only clades from B.*

Theorem 2. *An exact cubic-time (in the input data size) algorithm is constructed which, given a binary tree P, binary network S, and a collection B of subsets of M, outputs a binary tree G and a minimum embedding ${s_2}'$ minimizing the sum of costs of embeddings of P to G and of G to S provided that G contains only clades from B.*

To conclude this subsection, we note that the practical relevance of this problem is, in particular, explained in [8], where extensive references to the history of the intermediate tree problem in the context of bioinformatics are provided. The definition of a mapping from one tree to another, known in the literature as "alpha mapping," was proposed in the mid-1990s in [13,14]. For the case of trees T_1 and T_2, linear time algorithms for constructing a unique embedding $s'\colon T_1 \to T_2$ are known [20,21].

2.3. Definition of an Embedding of a Tree into a Network

Definition 2. *Let $s'\colon T_1 \to T_2$ be an embedding. Recall that edges in T_2 are referred to as pipes. An edge $e \in T_1$ enters a pipe $t \in T_2$ if s' maps e_+ to t_+ or above t_+ and maps e_- to t or below t. Informally, in the image tree, the edge e persists at t_+ and either becomes lost inside t or goes further through the whole t (see Figure 5a). Similarly, an edge e leaves a pipe $t \in T_2$ if s' maps e_+ to t_- or above and maps e_- strictly below t_-. Informally, in the image tree, the edge e arises at t_- or above and persists in one of the child pipes of t_-, maybe partially (see Figure 5b). Such a description is called an evolutionary scenario of T_1 in T_2.*

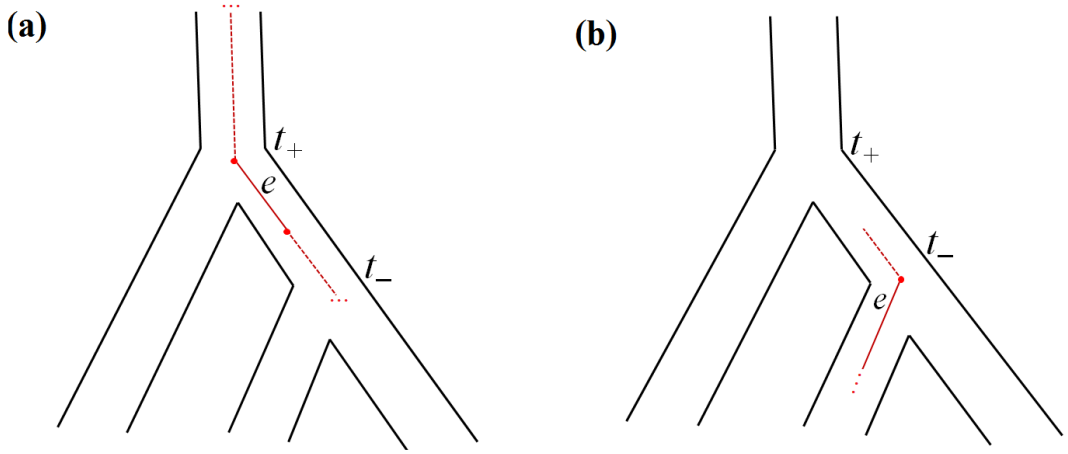

Figure 5. (**a**) Edge e enters pipe t: edge e persists at t_+ and either becomes lost in t (bold line) or leaves t (dashed line). (**b**) Edge e leaves the pipe t: edge e arises at t_- (bold line) or arises above and persists in one of the child pipes of t_-, passing through the whole t_1 or through its part (dashed line).

From ***binary network*** S, we understand a directed acyclic graph with nodes divided into four groups, one node of in-degree 0 and out-degree 1 (the *super-root*), nodes of in-degree 1 and out-degree 2 (*tree-type* nodes, including the *root*), nodes of in-degree 2 and out-degree 1 (*hybrid* nodes), and nodes of in-degree 1 and out-degree 0 (*leaves*). Thus, a tree is a particular case of a network with no hybrid nodes. A node may have two parents, so the notion of a nearest common ancestor is not well defined (see Figure 6). If there is a directed path from node x to node y, then we say that x is *above* y and write $x \geq y$ or $y \leq x$. A *hybrid pipe* t is a pipe whose endpoint t_- is a hybrid node. A *tree-type* pipe is defined similarly.

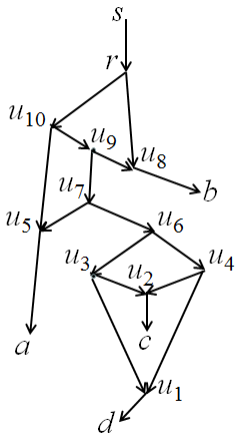

Figure 6. Example of a network: nodes u_3, u_4, u_6, u_7, u_9, u_{10}, and r are tree-type; nodes u_1, u_2, u_5, and u_8 are hybrid; and a, b, c, and d are leaves. There are four groups of nodes.

The continuation of this order onto all nodes and pipes, which is also written as x *is above* y ($x \geq y$) or y *is below* x ($y \leq x$), is defined similarly. For example, pipe e is below e_+ and above e_-. Sometimes, the term *ancestor* is used instead of *above*, and *descendant* is used instead of *below*; a node located immediately above some node is called its *parent*, and a node located immediately below some node is called its *child*; nodes located immediately below the same node are said to be *sibling* nodes; each of them is called a *sister* of any other one.

A *subnetwork* S_x is the network consisting of a node x and everything that lies below it, together with a root pipe (for S_x) entering x from above; if there are two such pipes, choose any of them. The leaf set in S_x is called a *clade* in S or the clade of the node x, or also

the clade of the root pipe entering x; we denote the clade of x by $x_<$. A subnetwork may contain nodes with two parents, such that one of them does not belong to S_x. Such nodes of a subnetwork will still be referred to as *hybrid* nodes. Throughout what follows, by a **network** we mean a *rooted binary network with a root pipe*, and similarly for a subnetwork.

As in the case of two trees, for a tree G and a network S we define a continuation s'' of a mapping s from leaves in G to leaves in S, though, on interior nodes, the value of s'' can also be a pipe in S. Namely, let a *continuation* s'' from nodes in G to nodes and pipes in S be any continuation of s preserving the natural ordering $\leq$ in G and having the following properties, the super-root in G is mapped to the root pipe in S; and the image $y = s''(x)$ of a node x is not a hybrid node. If $s''(x)$ is a tree-type node, then the paths $p_{x\text{-}x1}$ and $p_{x\text{-}x2}$ of the child edges of x enter different child pipes of $s''(x)$. Moreover, the mapping s'' for any edge $e \in G$ is *complemented* with pointing to a directed *path* p_e which consists of nodes in S leading from $s''(e_+)$ to $s''(e_-)$. An example in Figure 7b is the node e_-, which is mapped to t_-. For the case of a tree, such a path p_e is unique; above, it was called the path of the edge e. Such a non-unique s'' will be called an ***embedding*** of a tree G into a network S.

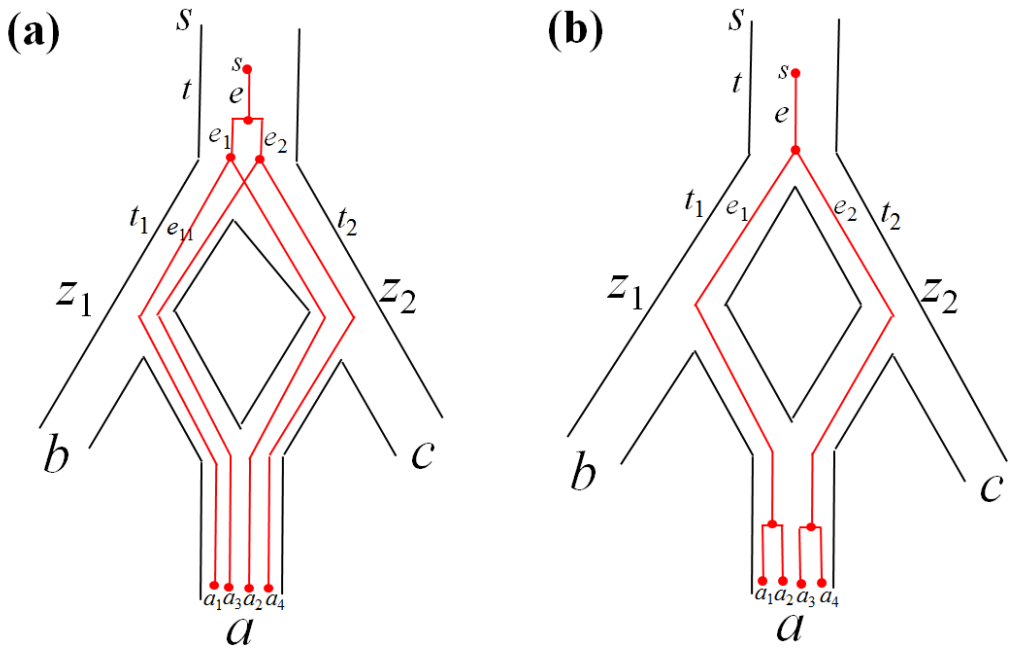

Figure 7. Example of two different minimum embeddings $G \to S$ with $G = ((a_1,a_2),(a_3,a_4))$ and a network shown in the figure. (**a**) The edges e_1 and e_2 neither enter nor leave t. The edge e_{11} leaves t_1. (**b**) The edge e_1 leaves the pipe t because s' maps e_{1+} to t_- and maps e_{1-} strictly below t_-, and p_{e1} passes through t_1. Similarly, e_2 leaves t through t_2.

Now we will repeat Definitions 1 and 2 for a network. An edge $e \in G$ ***enters*** a pipe $t \in S$ if s'' maps e_+ to t_+ or above t_+, maps e_- to t or below t, and p_e passes through t, maybe partially. Similarly, an edge e ***leaves*** a pipe t if s'' maps e_+ to t_- or above t_-, maps e_- strictly below t_-, and p_e passes through t_i, maybe partially, where t_i is a child pipe of t.

Let us be given an embedding s'': $G \to S$. A ***duplication*** in G is a node of degree 3 which is mapped to a pipe. A ***loss*** in G is a pair consisting of an edge $e \in G$ and a tree-type node $z \in S$, such that z lies in p_e (and does not coincide with $s'(e_+)$ or $s'(e_-)$). A loss is said to be *implicit* if one of the subnetworks with a root pipe starting at z does not contain images from G_0, otherwise, it is said to be *explicit*. The *locus* of a duplication is defined to be its image, and the locus of a loss is the node z. Duplications and losses will be called evolutionary *events*. We assume each of them to be assigned with its cost, a strictly positive rational number. The *cost* $c(s'')$ of an embedding s'' is the total cost of all *duplication* and *loss* events in it. The notions defined in this paragraph are also well applicable to an embedding s'': $P \to G$, which leads to costs of the embeddings s_1'': $P \to G$ and s_2'': $G \to S$. As a result,

we have two costs, $c_1(s'')$ and $c_2(s'')$, and their sum $c_1(s'') + c_2(s'')$. The duplication and loss events were introduced in [14] in connection with the "alpha" mapping.

Definition 3. *A minimum embedding s' of a tree G into a network S is an embedding s'' with the minimum cost.*

It is easy to construct a *minimum embedding* s': $G \to S$, where S is a network (minimum with respect to *duplication* and *loss* events). However, there can be many such minimum embeddings.

Such embeddings can be constructed by induction, where the induction step consists of embedding a subtree G_e into a subnetwork S_t in ascending order of their size (first over all t until we reach the root pipe, and then over all e until we reach the root edge), as is usually performed in dynamic programming.

At a step of this induction, costs of two embedding variants may happen to be the same; an example is shown in Figure 7, where e is a root edge and e_- is mapped to either t (then e is duplicated in t) or t_- (then e forks together with t). This generates two different minimum embeddings. Namely, for the tree $G = ((a_1,a_2),(a_3,a_4))$ and for the network with leaves a, b, and c, we show two different minimum embeddings under duplication cost 2 and implicit loss cost 1. Each of them has s cost of 6; the first embedding has one duplication and 4 implicit losses, whereas the second has two duplications and two implicit losses.

2.4. Algorithm for Constructing an Intermediate Tree

We present an algorithm for the general case where S is a network and where duplications and explicit and implicit losses have arbitrary costs. In the first reading, it might be convenient to assume that S is a tree. Recall that M is the set of leaves of the sought-for intermediate tree G^*, see Figure 3, where G is a variable tree, the argument of the functional over which the cost of an intermediate tree G is minimized. Let c_{dp} and c_{dm} denote, respectively, the costs of duplications under mappings are s_1 and s_2; e_{lp} and e_{lm} are the costs of explicit losses; and i_{lp} and i_{lm} are the costs of implicit losses.

For an arbitrary non-empty subset $A \subseteq M$, *denote* by $E(A)$ the set of edges $e \in P$ for which the following is valid: $s_1(e_<) \subseteq A$ and there is no $e_1 \in P$ with $e_1 > e$ for which $s_1(e_{1<}) \subseteq A$. For example, for the root edge r in G (or root pipe in S and mapping $s_2(s_1)$) we have $E(r_<) = \{r'\}$, where r' is the root edge in P. It is easily seen that a non-root edge t in the tree G is entered by precisely edges from $E(t_<)$. We will also consider the relation $s_2(t_<) \subseteq A \subseteq S_0$, where $t \in G$.

The algorithm presented below exhaustively examines all *partitions* of each set $A = (A_1 \cup A_2) \in B$, $A \subseteq M$. More precisely, we exhaustively examine all *pairs* consisting of a **set** $A \in B$ and a **pipe** $z \in S$, where $s_2(A) \subseteq z_<$. For that, we use the following order on pairs (A,z): ascending order of the cardinality $|A|$ of A; for equal cardinalities, we arbitrarily fix an order on the first coordinate; for a fixed A, we fix any order extending the relation $<$ over pipes $z \in S$ from leaves to the root. According to this order, the algorithm constructs a *final* tree $G = G(A,z)$ with leaf set A, partition $A = (A_1 * A_2)$, and root edge t in G which starts in z. The algorithm constructs a tree G so that $A_1 = t_{1<}$ and $A_2 = t_{2<}$, where t_1 and t_2 are children of the edge t. The algorithm also calculates $C(A,z)$, the sum of costs of embeddings into G over all subtrees in P with root edges $e \in E(A)$ plus the cost of the minimum embedding of G itself into the subnetwork S_z. This whole sum will be called the ***cost*** $C(A,z)$ ***of the tree*** $G(A,z)$. Clearly, the tree $G^* = G(M,r)$ obtained at the output of the algorithm is a solution to the intermediate tree problem, and its *cost* equals $C(M,r)$. The flow chart of the algorithm is shown in Figure 8.

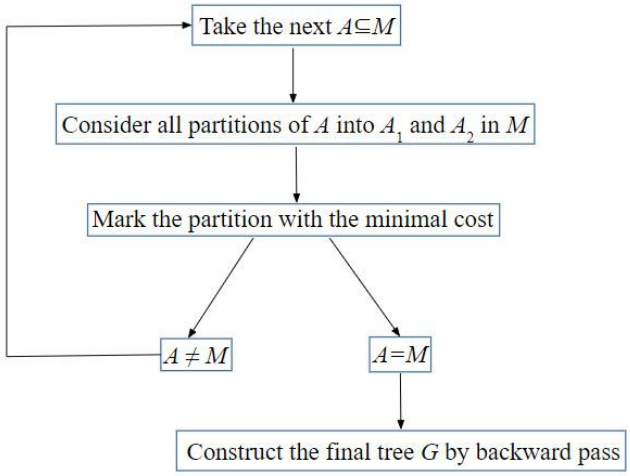

Figure 8. The flow chart of the algorithm for Theorem 1. The cost of a partition of A is computed based on the induction assumption that partitions of A_1 and A_2 and their costs are known.

The induction base for a one-element set $\{m\} = A$ and a leaf pipe $z \in S$, the tree $G(A,z)$ consists of m and the root edge in m, then $C(A,z) = c_{dp} \cdot \sum\limits_{x \in E(A)} (|x_<| - 1)$.

Consider the induction step. It includes four cases, in which T denotes a candidate for $G(A,z)$ and C denotes a candidate for the cost $C(A,z)$. In Cases 1 and 2 below, $z \in S$ forks in pipes z_1 and z_2, and in Case 2 it can also be a leaf in S.

If S is a tree, Case 4 is impossible, so only one of Cases 1–3 can be applied. Moreover, if $s_2(A) \subseteq t_{1<}$, then we may only proceed with Case 1; if $s_2(A_1) \subseteq t_{1<}$, $s_2(A_2) \subseteq t_{2<}$, we may only proceed with Case 3; if none of these conditions is satisfied, we proceed with Case 2. However, below we present the general algorithm assuming S to be a network.

(1) Let $z_- \in S$ be a tree-type node (a hybrid node is considered in Case 4), and assume that an edge $t \in G$ *leaves* z and enters the pipe z_1, i.e., $s_2'(t_-) \leq z_1$. This is the case of a *loss* of edge t at node z_- with respect to the embedding s_2'. Let $T = G(A,z_1)$ and $C = (A,z_1) + l$, where $l = e_{lm}$ if $z_{2<}$ contains an element from $s_2(M)$ and $l = i_{lm}$ otherwise. Note that $A = t_<$ and that for the tree S, the relation $s_2(A) \subseteq z_{1<}$ holds in this case only. Symmetrically for t_2.

(2) Let $A_1,A_2 \in B$ be a partition of A, and assume that there is *duplication* in a pipe $z \in S$ with respect to s_2', i.e., $s_2'(t_-) = z \in S$ and the edge t is forked in z into t_1 and t_2. Let $T = G(A_1,z)\hat{}G(A_2,z)$, where $\hat{}$ denotes the union of trees $G(A_1,z)$ and $G(A_2,z)$ under a common root, i.e., $t_{1<} = A_1$ and $t_{2<} = A_2$. *Denote* by k the number of nodes $v \in P$, such that $s_1'(v) = t_-$, i.e., one edge leaving v lies in $E(A_1)$ and the other is in $E(A_2)$. Let

$$C = C(A_1,z) + C(A_2,z) + \{c_{dm} + c_{dp} \cdot (|E(A_1)| + |E(A_2)| - |E(A)| - k) + e_{lp} \cdot (|E(A_1)| + |E(A_2)| - 2k)\}$$

If $s_1(P_0)$ intersects with both A_1 and A_2, or $C = C(A_1,z) + C(A_2,z) + \{c_{dm} + i_{lp} \cdot |E(A)|\}$ otherwise.

(3) Let $A_1,A_2 \in B$ be a partition of A and assume that $s_2'(t_-) = z_-$, i.e., the edge t is forked at a furcation $z_- \in S$, z_1 is entered by edge t_1 with clade A_1, and z_2 is entered by edge t_2 with clade A_2. This is the case of a coordinated furcation of t and z. As in Case 2, let $T = G(A_1,z_1)\hat{}G(A_2,z_2)$, and let k be the number of nodes $v \in P$ with $s_1'(v) = t_-$. Let

$$C = C(A_1,z_1) + C(A_2,z_2) + \{c_{dp} \cdot (|E(A_1)| + |E(A_2)| - |E(A)| - k) + e_{lp} \cdot (|E(A_1)| + |E(A_2)| - 2k)\}$$

If $s_1(P)$ intersects with A_1 and A_2, and $C = C(A_1,z_1) + C(A_2,z_2) + i_{lp} \cdot |E(A)|$ otherwise. Note that the condition $s_2(A_1) \subseteq t_{1<}$ and $s_2(A_2) \subseteq t_{2<}$ on the tree S holds in this case only.

(4) Edge t leaves a hybrid node z and goes into its unique child pipe z_0, i.e., $s_2'(t_-) \leq z_0$. Then, $T = G(A,z_0)$ and $C = C(A,z_0)$.

For a final $G(A,z)$ at step (A,z), the algorithm chooses the tree T with the minimum value of C over all cases and all partitions of A. This number C will be the cost $C(A,z)$ of this tree T; at each step (A,z), the algorithm obtains the cost $C(A,z)$ of a final tree $G(A,z)$. For

the final output tree, the algorithm chooses $G^* = G(M,r)$, where r is the root pipe in S. This G^* is a solution to the intermediate tree problem. All these facts are proved below. □

Proof of Theorem 1. The proof is constructed by induction, in the dynamic programming style typically used for the optimization of a loss function, the latter being in our case the cost of the tree G.

The algorithm exactness. Using induction on all pairs (A,z), let us check that for each of the above four cases the *cost* $c(T,A,z)$ of the tree T specified in this case equals the number C given there. In particular, the cost of the final tree $G^* = G(M,r)$ is $C(M,r)$. The induction base, as well as induction steps of Cases 1 and 4, are obvious. Consider Case 3 of the algorithm; Case 2 is considered similarly. Recall that $c(T,A,z) = c(s_1') + c(s_2')$, where $c(s_1')$ is the total cost of embeddings into T of subtrees in P with root edges from $E(A)$ (we call it the *first* part of the cost) and $c(s_2')$ is the cost of the embedding of T into S_z (the *second* part of the cost); denote $c(T,A,z) = 1(T,A,z) + 2(T,A,z)$. In Case 3, we obtain $1(T,A,z) = 1(T_1,A_1,z_1) + 1(T_2,A_2,z_2) + d$, where T_1 and T_2 are subtrees with root edges t_1 and t_2, and d is the total cost of duplications in t and losses in $t\_$ for the embedding of subtrees from P into T. Similarly, $2(T,A,z) = 2(T_1,A_1,z_1) + 2(T_2,A_2,z_2)$. Hence, $c(T,A,z) = c(T_1,A_1,z_1) + c(T_2,A_2,z_2) + d$. By the induction hypothesis, $C(A_1,z_1) = c(T_1,A_1,z_1)$ and $C(A_2,z_2) = c(T_2,A_2,z_2)$. It remains to show that $\{\dots\} = d$, where $\{\dots\}$ is the third term in the expression for C. Indeed, the edge t is entered by $|E(A)|$ edges from P, and the edges t_1 and t_2 are entered, respectively, by $|E(A_1)|$ and $|E(A_2)|$ edges that are children of the edges entering t. From the subtrees with root edges in $E(A)$, delete everything that is below the edges entering t_1 or t_2. We obtain $|E(A)|$ trees, in total with $|E(A_1)| + |E(A_2)|$ leaves. Let I be the set of inner nodes in them. Clearly, $|I| = |E(A_1)| + |E(A_2)| - |E(A)|$. Every node in I is either a duplication in t or a furcation in $t\_$, and vice versa, any such duplication or furcation corresponds to a node in I. By the definition, the number of furcations is k, and all other nodes in I are duplications. Among the $|E(A_1)| + |E(A_2)|$ edges entering t_1 or t_2, exactly $2k$ do not generate a loss in $t'\_$. Losses in $t'\_$ are either all explicit or all implicit, which depends on the sets $A_1 = t_{1<}$ and $A_2 = t_{2<}$ only. In the first case, we obtain the first expression for C. In the second case, we have $k = 0$, which implies that there are no duplications in t; i.e., $|E(A_1)| + |E(A_2)| = |E(A)|$. This follows from the facts that P and G are trees and from the definition of s'.

Now, following the same induction, we prove that the cost $C(A,z)$ of the tree $G(A,z)$ constructed at step (A,z) is the desired minimum in the intermediate tree problem. Let X be a minimum tree with leaf set A; by the condition, all its clades belong to some B which with respect to P, S_z, and some embedding s_2: $X_t \to S_z$ is minimal in the intermediate tree cost; in particular, s_2: $X_t \to S_z$ is a minimum embedding. For X, the conditions of one of Cases 1–4 of the algorithm are satisfied. Let, for example, the conditions of Case 3 be fulfilled; the other cases are considered similarly. Denote by $f(X) = s_1'(X) + s_2'(X)$ the minimized cost in the intermediate tree problem. Denote by X_1 and X_2 the subtrees with root edges t_1 and t_2 embedded in the subnetworks S_{z1} and S_{z2}. Their clades are some sets A_1 and A_2 in B. Repeating the arguments of the preceding paragraph, we obtain $f(X) = f(X_1) + f(X_2) + d$, where d is the sum of duplications in t in losses in $t\_$ when embedding the subtrees from P into X. Then, we have

$$C(A,z) \le C(A_1 \cup A_2, z) = C(A_1,z_1) + C(A_2,z_2) + d \le f(X_1) + f(X_2) + d = f(X);$$

here, $C(A_1 \cup A_2, z)$ is the value of C computed with respect to the partition A_1 and A_2. Hence, $f(G(A,z)) = f(X)$.

The algorithm runtime. We construct sets $E(A)$ for all $A \in B$ in a time of the order of $|B| \cdot |P|$. Inclusions and intersections for sets in B are computed in a time of the order of $|B|^2 \cdot |M|$. Inclusions of sets from B into clades of the network S are computed in a time of the order of $|B| \cdot |S| \cdot |M|$. All partitions of each set $A \in B$ into sets $A_1, A_2 \in B$ are computed in a time of the order of $|B|^3$; for every triple A, A_1, A_2 of sets one checks that A_1 and A_2 are disjoint and $|A_1| + |A_2| = |A|$. For every $A \in B$ and every partition into

A_1 and A_2, one computes the number k of nodes $v \in P$ in a time of the order of $|P|$. The number of sets A and of partitions of A is no greater than $|B|$, so the total computation time for all k and for all A is of the order of $|B|^2 \cdot |P|$.

At each induction step when constructing the trees $G(A,z)$, for each $A \in B$ and each pipe t we examine at most $|B|$ partitions of A. For each partition, the cost C is computed in a constant time. Therefore, the total construction time for all $G(A,z)$ is of the order of $|B|^2 \cdot |S|$. The total runtime of the algorithm is at most of the order of $|B| \cdot (|B| \cdot |M| + |S| \cdot |M| + |B|^2 + |B| \cdot |P| + |B| \cdot |S|)$. □

Remark 1. *If S is a tree, then the described algorithm can be generalized so that to take into account horizontal transfers when computing the embedding of G into S. Such an event assumes that pipes in S are divided into time layers, which often results in occurrence of nodes of degree 2 (see [22]). A horizontal transfer consists of transferring an edge of G from one pipe of S into another pipe lying in the same time layer. A transfer may have a furcation at the source or do not have it (in the first case, the furcation occurs in the pipe, with one of the child edges remaining in its pipe and the other being transferred). To obtain the corresponding algorithm, we need to add two more cases to the described-above algorithm. These are furcation of an edge e in z with transferring one of the child edges to another pipe in the same time layer, and transferring an edge e (without furcation) to another pipe of the same layer. In both cases, it is not necessarily required that* $s_2(A) \subseteq z_<$.

2.5. Constructing the Main Parameter of the Algorithm

Recall that M is a set of leaves in a tree G^* to be constructed by the algorithm from Section 2.4 (Theorem 1). The sought-for intermediate tree G^* depends on a fixed collection $\boldsymbol{B}$ of subsets in M; this $\boldsymbol{B}$ is called the *main parameter* of the algorithm. Since the intermediate tree problem is NP-hard, any polynomial-complexity algorithm does not necessarily find an intermediate tree G^{**} with an absolutely minimum cost which is a solution of the original problem in the unconditional setting (if $P \neq NP$).

Given an arbitrary collection B' of subsets of M, the main parameter is a collection $\boldsymbol{B}$ satisfying property (*), whose construction is described in Appendix A; this $\boldsymbol{B}$ contains, at most, $|M|$ times as many elements as the original B'.

Let us extend a given B' by adding images (under s_1) of all clades in P and inverse images (under s_2) of all clades in S; then we pass to B using the procedure described in Appendix A. Nevertheless, execution of the algorithm from Section 2.4 with this collection $\boldsymbol{B}$ may produce a result G^* whose cost can sometimes be essentially greater than the cost of the tree G^{**}.

Let us present a further extension for collection B', which could be of use in applied computations. We first describe this extension for the case where S is a tree. Consider the trees P and S with their embedding composition $s_2(s_1)'$. For each pipe $z \in S$, one considers the set O_z of edges in P that *leave* z. For each subset $O' \subseteq O_z \subseteq P$, one adds to B' all images $s_1(\cup\{e_< \mid e \in O'\})$ for all $O' \subseteq O_z$ and all $z \in S$ and then extends the obtained B' to the main parameter $\boldsymbol{B}$. Simulation has shown that for such $\boldsymbol{B}$ runtime of the algorithm from Section 2.4 remains to be cubic but the cost C^* of the tree G^* reduces significantly, so that C^* becomes close to the solution of the unconditional problem (simulation results are not presented here).

If, nevertheless, the main parameter $\boldsymbol{B}$ is not sufficiently large to provide a smaller C^*, we may use another mapping of P to S, which provides the minimum of duplication clusters (the definition and an efficient construction algorithm for this mapping are described in [23]). For example, the two bottom duplications in Figure 2b form a cluster in the sense of [23]. That embedding differs from our embedding s' in that duplications often occur higher than under s', which leads to increasing cardinalities of the sets O_t. If a main parameter $\boldsymbol{B}$ slows down the execution of the algorithm too much, we may, vice versa, consider the tree S with time layers, which reduces the sets O_t; in this case, the definition of the embedding and an efficient construction algorithm for it are described in [22].

If S is a network, then the extension of a standard collection B_0 is defined via the embedding s': $P \to S$ constructed in Theorem 3 disregarding *ILS* events. Let us briefly recall the construction. Exhaustively examine all pairs consisting of an edge $e \in P$ and a pipe $z \in S$ with condition $s_2(s_1)(e_<) \subseteq z_<$. For each such pair (e,z), one constructs a minimum embedding of the subtree P_e into a subnetwork S_z inductively; when passing to a larger pair (e,z), one chooses the case in Section 2.4 with the smallest cost. The resulting embedding is a mapping from $P = P_{r'}$ to the network $S = S_r$, where r' and r are the root edge and the root pipe in P and in S.

One can also adjust collection $\boldsymbol{B}$ using an embedding which, along with duplications and losses, takes account of Incomplete Linear Sorting events (see Section 2.7 below).

2.6. Example of the Algorithm Operation

Let us show how the algorithm works using an example from [6], where $S = (((a,b),c),(f,(d,e)))$ is a tree, $P = (((a,f),c),(e,(b,d)))$, $M = \{a,b,c,d,e,f\}$; and s_1 and s_2 map each leaf to itself. Let the cost of all explicit losses be 2, of all implicit losses be 1, and of all duplications be 3; and collection B' is as follows: $\{a,f\}$, $\{a,f,c\}$, $\{b,d\}$, $\{e,b,d\}$, $\{a,b\}$, $\{a,b,c\}$, $\{d,e\}$, $\{f,d,e\}$, the set M itself, and all one-element sets in M. The embedding s': $P \to S$ is shown as an image tree in Figure 9.

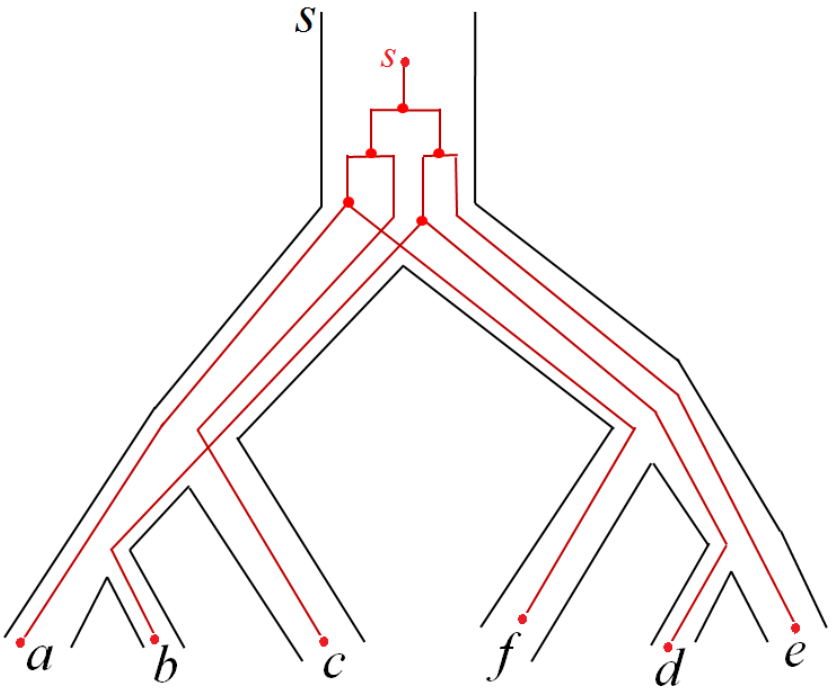

Figure 9. Embedding s' of the tree P into the tree S. This embedding is constructed according to the definition of an embedding in Section 2.1.

Construct the main parameter $\boldsymbol{B}$. The root pipe adds to B' the sets $\{a,b,f,d\}$, $\{a,f,e\}$, $\{b,c,d\}$, $\{c,e\}$, $\{a,b,c,f,d\}$, $\{a,c,f,e\}$, $\{a,b,d,e\}$, and $\{b,c,d,e\}$. The left-hand child pipe adds to it the sets $\{a,c\}$ and $\{b,c\}$, and the righ-hand one adds $\{f,d\}$ and $\{f,e\}$. Clearly, $\boldsymbol{B}$ satisfies condition (*).

Construct trees $G(A,z)$ for all sets $A \in \boldsymbol{B}$, $z \in S$, z being the ancestor edge for the set A of leaves. For one-element and two-element sets A, the trees are trivial. Consider three-element sets A. For the set $A = \{a,f,c\}$, the trees $((a,f),c)$ and $((a,c),f)$ have the same cost. For $\{e,b,d\}$, the tree $(b,(d,e))$ is the one of the minimum cost among the two trees. For $\{a,b,c\}$ and $\{f,d,e\}$, the trees $((a,b),c)$ and $(f,(d,e))$ are those of the minimum costs among the three trees. For $\{a,f,e\}$ and $\{b,c,d\}$, the trees $(a,(f,e))$ and $((b,c),d)$ are those of the minimum costs among the two trees.

For the set $\{a,b,f,d\}$ in $\boldsymbol{B}$, the minimum cost is given by the partition into $\{a,b\}$ and $\{f,d\}$; for $\{a,c,f,e\}$, by the partition into $\{a,c\}$ and $\{f,e\}$; for $\{a,b,d,e\}$ and $\{b,c,d,e\}$, by the partitions into $\{a,b\}$ and $\{d,e\}$ and into $\{b,c\}$ and $\{d,e\}$, respectively.

For the set $\{a,b,c,f,d\}$ in $\boldsymbol{B}$, the minimum cost is given by the partition into $\{a,b,f,d\}$ and $\{c\}$. Finally, for the whole M, the minimum cost is given by $\{a,b,f,d\}$ and $\{c,e\}$. Thus, the resulting tree G^* is $(((a,b),(f,d)),(c,e))$, the resulting cost $C^* = 26$, and the obtained embeddings s_1': $P \to G$ and s_2': $G \to S$ are shown in Figure 10. In total, they contain 2 duplications and 11 losses. This is the global minimum G^{**} of the cost of a tree G in the unconditional problem.

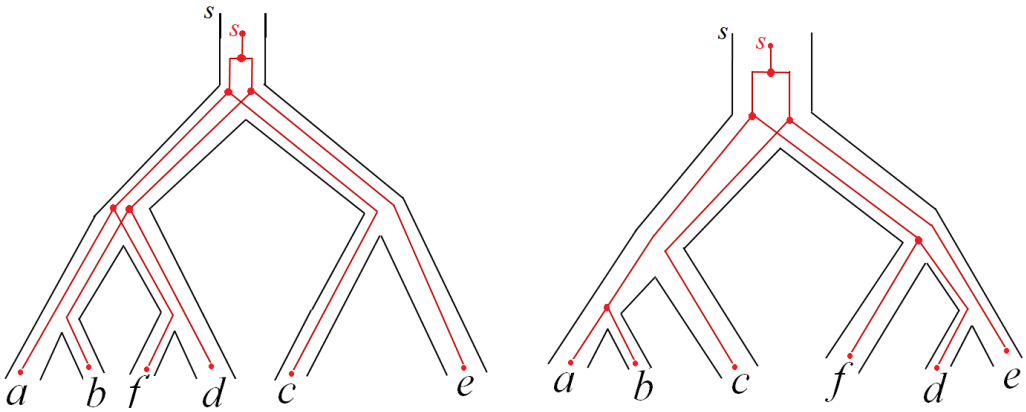

Figure 10. Embeddings $P \to G^* \to S$ with the tree G^* constructed by our algorithm for the main parameter $\boldsymbol{B}$ specified above. These embeddings are constructed according to the definition of an embedding in Section 2.1. (**Left**): the tree G^* with the tree P embedded in it; (**right**): the tree S with the tree G^* embedded in it.

For all costs equal to 1, our algorithm outputs the same tree G^*. For the same input data, the algorithm from [8] has constructed the tree $G = (((c,a),f),(b,(d,e)))$; the corresponding embeddings are shown in Figure 11. Here, in total, there are 3 duplications and 11 losses.

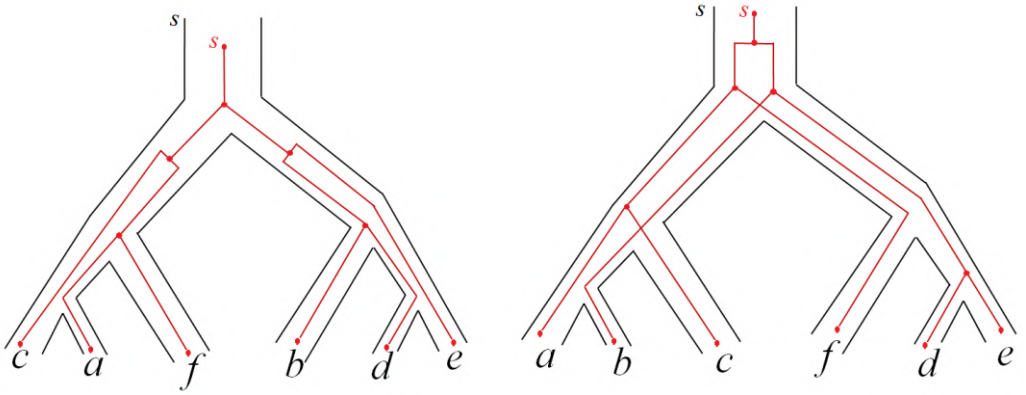

Figure 11. Embeddings $P \to G \to S$ with the tree G constructed by the algorithm from [8]. These embeddings are constructed according to the definition of an embedding in Section 2.1. (**Left**): tree G with tree P embedded in it; (**right**): tree S with tree G embedded in it.

If we confine ourselves with the main parameter B', our algorithm constructs a tree with a cost strictly greater than in the unconditional problem; namely, 3 duplications and 11 losses (see Figure 12). In [8], there are also 3 duplications and 11 losses, but our embeddings are quite different (see Figures 11 and 12).

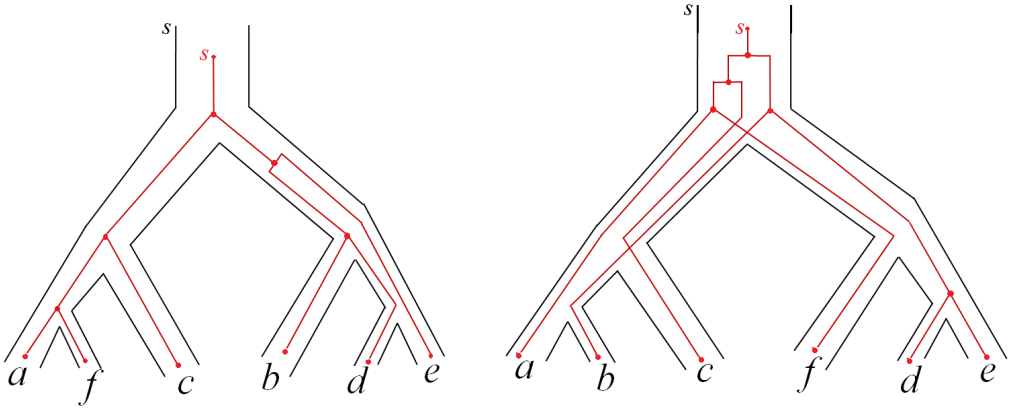

Figure 12. Embeddings $P \to G^* \to S$ with tree G^* constructed by our algorithm for the other main parameter $\boldsymbol{B}$. These embeddings are constructed according to the definition of an embedding in Section 2.1. (**Left**): tree G^* with tree P embedded in it; (**Right**): tree S with tree G^* embedded in it.

2.7. ILS-Minimum Embedding of a Tree into a Network and Theorem 3

Let G and S be any given binary tree and binary network, respectively, and let s'': $G \to S$ be an embedding. The *Incomplete Linear Sorting* (*ILS*) event is an edge $e \in G$ entering a pipe $t \in S$ provided that at least two edges enter t (see [9]). The *locus* of an *ILS* event is the pipe t. The *ILS-cost* over a pipe $t \in S$ entered by $k_t \geq 1$ edges from G is defined as $c_t = c \cdot (k_t - 1)$, where c is the cost of a single *ILS* plus the costs of all duplications and all losses in t. *The ILS-cost* over S is defined as $\sum\limits_{t \in S\downarrow} c_t$ plus the costs of all duplications and all losses in S, where t runs over pipes $t \in S$ entered by at least one edge from G; the latter is denoted as $t \in S\downarrow$. Such pipes will be called *non-empty*, while all others are said to be *empty*. Clearly, $\sum\limits_{t \in B\downarrow} c_t = c \cdot (\sum\limits_{t \in B\downarrow} k_t - n)$, where n is the number of non-empty pipes in S.

An ***ILS-minimum embedding*** of s': $G \to S$ is defined to be an embedding with the *total minimum cost* of $c \cdot (\sum\limits_{t \in B\downarrow} k_t - n)$, where n is the number of non-empty pipes in s', plus the costs of all duplications and losses. This cost is called the *ILS-minimum cost* of s'; it reflects how much s' differs from an isomorphism.

Similarly, we define an *ILS event* for a given set $\{s_e$: $G_e \to S\}$ of embeddings of different trees G_e into a given network S. Edges occurring in S from different s_e are considered equally. The *ILS-cost* of a set $\{s_e\}$ is defined by the same expression $c \cdot (\sum\limits_{t \in B\downarrow} k_t - n)$, where n is the number of pipes that are non-empty for at least one of the embeddings s_e, plus the costs of all duplications and all losses in S. A set $\{s_e\}$ is called an ***ILS-minimum set of embeddings*** if the *ILS-cost* of $\{s_e\}$ is minimal as compared to any set $\{r_e$: $G_e \to S\}$ of embeddings of these G_e into S with some conditions on all s_e and r_e. This cost is called the *ILS-minimum cost* of $\{s_e\}$.

A *bridge* in S is a pipe t for which any path from the super-root to the clade $t_<$ contains t. In our case, this is equivalent to the following condition: t is a pipe such that its removal makes the undirected graph corresponding to S disconnected. A leaf pipe is always a bridge. A *block* in S is an inclusion-maximal set of nodes in S for which the induced subgraph does not contain a bridge. Two blocks either coincide or are disjoint. For example, in Figure 6, bridges are the pipe u_7-u_6 and also the root and leaf pipes; blocks are the super-root, leaves, and two complements to the edge u_7-u_6 without the root or leaf pipes. Note that blocks of a network S form a rooted, though now non-binary, tree. For a network S of *level* k any block contains at most k hybrid nodes. This number k is a measure of distantness between S and the tree.

We say that pipe t *enters* block B if B contains its head t_- but does not contain its tail t_+; such a t is unique for B and is a bridge in S. Pipe t *leaves* B if B contains its tail t_+ but does not contain its head t_-; there can be many pipes leaving B, and all of them are bridges

in S. In what follows, for any pipe $t \in S$ we *assume* a ***unique block*** B_t in S, such that $t_- \in B$; however, we consider the embedding $G_e \to S_t$. The mapping $t \mapsto B_t$ between incoming pipes t and blocks B in S (except for the super-root) is a one-to-one mapping.

For an arbitrary non-empty set A of leaves in an arbitrary network S, $E(A)$ is *defined* exactly as in the above case where S is a tree. Namely, for an edge $e \in G$ the *relation* $e \in E(A)$ means the following: $s(e_<) \subseteq A$ and there is no $e_1 \in G$ for which $e_1 > e$ and $s(e_{1<}) \subseteq A$. It is easily seen that for any edge e entering a pipe $t \in S$ we have $e \in E(t_<)$ or $\exists e' > e$ ($e' \in E(t_<)$). As is noted above, if S is a tree, then finally there remains only one disjunction term $e \in E(t_<)$.

Let s'': $G \to S$ be an embedding, and introduce the following condition:

(*) *each non-root pipe t which is a bridge in S is entered by precisely edges from* $E(t_<)$.

We have proved the following result.

Theorem 3. *Let s:* $G_0 \to S_0$ *be an arbitrary mapping from leaves of a binary tree G to leaves of a binary network S. The algorithm has been constructed which outputs an ILS-minimum continuation* s': $G \to S$ *of s with the property (*). Its runtime is of the order of* $|G| \cdot |S| \cdot k \cdot 2^{4k}$, *where* $|\cdot|$ *is the number of nodes in the tree or network and k is the level of the network S. It is exact if the cost of a single ILS is not less than the sum of the duplication cost and the cost of an implicit loss.*

In [9], a construction algorithm was suggested for a minimum (with respect to *ILS* events only) embedding of a tree into a network of level 1. Additionally, in [9] (section "Conclusion"), a method was outlined to generalize that algorithm to the case of a network of any level k. Our algorithm uses the idea from [9] combined with ideas of the algorithm of Theorem 1.

Denote by c_d, e_l, i_l, and c_i, respectively, the costs of duplication, explicit loss, implicit loss, and *ILS* event.

Claim 1. *If* $c_i \le c_d + e_l$, *then there exists an ILS-minimum embedding s*:* $G \to S$ *such that property (*) is valid.*

Proof. Let s' be any *ILS*-minimum embedding. We successively rearrange s' to a desired embedding s^*; to this end, we exhaustively examine all non-root bridges $t \in S$ in an arbitrary order. The algorithm that we will construct now is not used in what follows; i.e., we need Claim 1 as an existence result only. Namely, in the Proof of Theorem 3 we construct an embedding s': $G \to S$, which possesses property (*) and is minimal among the embeddings with property (*); considering Claim 1, it will be absolutely minimal.

We perform the proof by contradiction. Assume that $t \in S$ and we have the following:

(**) edge $e_1 \in G$ enters t, $e_1 \notin E(t_<)$, and e_1 is an edge the most distant from the root among edges satisfying these two conditions.

Let us show that a sibling edge e_2 of e_1 also satisfies all the three conditions, and rearrange s' so that again to obtain a minimum embedding s'' such that both e_1 and e_2 do not enter t, there arises one new edge entering t, and the number of edges entering all other pipes does not increase. We repeat this rearrangement until we find an edge e_1 satisfying condition (**).

Thus, take $e' > e_1$ ($e' \in E(t_<)$; such an e' is unique. Consider a path φ from the root through the edges e' and e_2 to some leaf and a path ψ of pipes that contain φ so that ψ ends at $t_<$. Then, ψ passes through t. Assume that some edge e_3 in φ enters t. Then, we must have $e_3 = e_2$. Indeed, by the choice of e_1, the edge e_3 on φ cannot lie below e_2, and since e_1 enters t, it cannot lie above e_2. Thus, e_1 and e_2 enter t and are forking from some edge e, i.e., they fork at $e_- \in G$. Let $z = s'(e_-)$.

If z is a pipe, then e_1 and e_2 appeared in z as a result of a duplication and entered child pipes of z, either both entered the same pipe z_1 or each one entered its own. In the first case, the duplication can be moved to z_1, child of z, thus reducing the numbers of losses and *ILS* events when computing the embedding cost of s'; in the second case, the duplication can be eliminated by moving it to z_-. In both cases, the cost of the resulting new embedding

becomes strictly less, which contradicts the minimality of s'. The case that z is a hybrid node is impossible.

If z is a tree-type node, then the edges e_1 and e_2 arise in z and are continued to t. Consider a new embedding s'', which differs from s' only by the fact that $s''(e_-) = t$, and draw a path p_e along any of the paths p_{e1} or p_{e2} (see Figure 13).

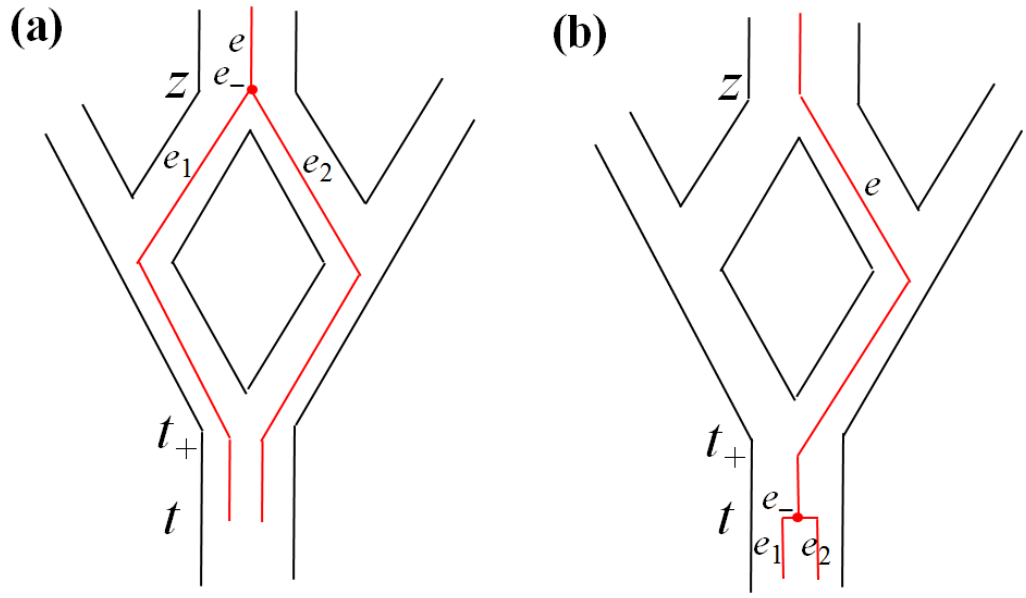

Figure 13. (**a**) Fragment of a network with a given embedding s' when there is *ILS* event at t_+. (**b**) The same fragment with the new embedding s'' when there is a loss at z and a duplication at t.

For s'', one explicit loss appears at z. In the node t_+, one *ILS* event is replaced with a duplication at t and loss at z. The embedding cost does not increase, but now the number of edges e entering t is less by 1, and the number of edges entering any other pipe have not increased. By repeating this modification, we find the desired *ILS*-minimum embedding s^*. □

Let a set D of hybrid pipes in a network S be fixed. For an embedding s'': $G_e \to S_t$ of a subtree G with root edge e into a subnetwork of S with root pipe t, a block $B_t \subset S$ is said to be *D-coordinated* if D is the set of non-empty hybrid pipes in B. For the same embedding s'', *denote* by $N(t,D)$ the set of all non-empty pipes in B_t. The cardinality of this set will play an important role in what follows.

For a set $M = \{G_e \to S_t\colon e \in X\}$ of embeddings of subtrees into a fixed subnetwork, where X is a fixed set of edges of G, a block $B_t \subset S$ is said to be *D-coordinated* if D is the set of hybrid pipes in B_t that are non-empty for at least one embedding in M.

Claim 2. *Let $e \in G$, $t \in S$ and B_t be a block in S with $t_- \in B_t$. We construct an algorithm which, given any leaf mapping $s\colon G_e \to S_t$ and any set D of hybrid pipes in a block $B_t \subset S$, outputs a set Y, such that $Y = N(t,D)$ if there exists a continuation $s''\colon G_e \to S_t$ of s that D-coordinates B_t. The runtime of the algorithm is if the order of $|B_t|$.*

Proof. Let such an s'' exist. The algorithm exhaustively examines bridges that leave B_t and records which of them are empty. For that, we use the fact that a bridge x leaving B_t is empty with respect to s'' if, and only if, the clade of x is disjoint with $s''(e_<)$. Then, the algorithm exhaustively examines tree-type pipes $x \in B_t$ from leaves to the root and labels x as empty, if and only if, either $x = t$ or both child pipes of x are empty; labels of hybrid pipes are known by the condition. □

For any embedding of leaves $s_0\colon G_0 \to S_0$, we delete from S the subnetwork of every pipe, such that its clade is disjointed with $s_0(G_0)$ and its sibling clade is not. In the resulting

graph S', we combine maximum-length sequences $t_1 > \ldots > t_n$ of pipes (t_{i+1} is a child pipe to t_i) in S for each of which (for $1 \leq i \leq n-1$) one child subnetwork has been deleted into a «compound» pipe t, which will be referred to as *new*; we again denote the resulting *binary network* by S'. For every new pipe in S', denote $l(t) = n \geq 2$; for all other pipes in S', denote $l(t) = 1$. The leaf mappings s_0: $G_0 \to S_0$ and s_0: $G_0 \to S_0{}'$ coincide.

For a binary network S', we formulate a **new minimum embedding problem**. We replace the search for a minimum embedding s': $G \to S$ with the search for a minimum embedding s': $G \to S'$. The latter is more convenient due to the fact that for every tree pipe $t \in S'$ there are edges e_1 and e_2 that enter, respectively, the child pipes t_1 and t_2. Furthermore, the cardinality of the network S' is not greater (and typically less) than the cardinality of S.

Thus, we define an *implicit loss* in S' to be a pair (e,t) where the edge e enters a new pipe $t \in S'$; if a new pipe is the root pipe $r \in S'$, then e is the root edge in G. Therefore, to several implicit losses in the former sense there corresponds a single implicit loss in the new sense. The *cost* of a new implicit loss is set to be $i_l \cdot (l(t) - 1)$, where i_l is the cost of a single implicit loss equal to its former value. The *locus* of an implicit loss is assumed to be the pipe t. An explicit loss and a duplication are defined as above and have the original costs e_l and e_d. An *ILS event* is defined as above; it is a pipe entered by strictly greater than one edge. However, the *cost* of a new *ILS event* on a pipe t is $c_i \cdot (k_t - 1) \cdot l(t)$, where k_t is the number of edges entering t. We split this cost into a sum of two terms, the *first ILS term* and the *second ILS term*, $c_i \cdot (k_t - 1)$ and $c_i \cdot (k_t - 1) \cdot (l(t) - 1)$. The c*ost* of a new *ILS* event on a network S is $c_i \cdot \left(\sum\limits_t k_t - n \right) + c_i \cdot \left(\sum\limits_t (k_t - 1)(l(t) - 1) \right)$, where $k_t \geq 1$ and where n is the number of non-empty pipes in S'. Here, we will also distinguish between the *first ILS term* and the *second ILS term*. The *ILS-minimum continuation* and the *ILS-minimum cost* are defined as above, taking into account all the above-mentioned events. Thus, the new problem is to find an *ILS*-minimum continuation s': $G \to S'$ of s_0 with property (*). This problem is of interest in connection with the following fact.

Claim 3. *Given a solution of the new problem by a linear-time algorithm, one can construct a solution of the original problem with property (*).*

Proof. Let s': $G \to S'$ be a solution of the new problem for a new network S', and let its *ILS*-minimum cost be c. Define an embedding s: $G \to S$ for the original problem by keeping the values $s(g) = s'(g)$ for all $s'(g)$ except for $s'(g) = t$, where t is the new pipe in S'. In the latter case, we have $s(g) = t_n$, where $t_n \in S$ is the lowermost pipe in $t \in S'$. The *ILS* cost of s is also c. Assume that there exists a solution s^*: $G \to S$ of the original problem with an *ILS*-minimum cost $c' < c$. Then, define an embedding s^{**}: $G \to S'$ by keeping the values $s^{**}(g) = s^*(g)$ for all $s^*(g)$ except for $s^*(g) = t_i$, where t_i is a part of the new pipe t. In the latter case, we have $s^{**}(g) = t$. However, the *ILS* costs of s^* and s^{**} are the same. Let us check property (*) for s. Consider edges entering pipe t with respect to s'; with respect to s, they enter either t (if t is not new) or $t_1, t_2, \ldots, t_n$ (otherwise). Since $E(t_<) = E(t_{1<}) = \ldots = E(t_{n<})$, we conclude that any bridge $t \in S$ is entered by exactly the edges from $E(t_<)$. □

We *again denote* the resulting network by S; its size is of the order of $k \cdot |G|$, where k is the level of the original (or equivalently, of the resulting) network.

For a given embedding s'': $G_e \to S_t$, define the ***B-cost*** of a block B to be the total cost of all events in B plus the cost of duplications and implicit losses in the pipe t. For a given set $\{s_e$: $G_e \to S_t\}$ of embeddings, let the ***B-cost*** of a block B be the total cost of all events in B plus the cost of all duplications and implicit losses in t that arise from all these embeddings s_e.

For an *arbitrary set* D of hybrid pipes in B_t, by a *D-minimum embedding* we call an *embedding* s'': $G_e \to S_t$ satisfying property (*), with D coordinating the block B_t, and having the smallest B_t-cost among all embeddings $G_e \to S_t$ that satisfy these two conditions. We *denote* such an s'' by $s(e,t,D)$ and *denote* its B_t-cost by $C(e,t,D)$. If there is no any such s'', we say that a D-minimum embedding is not defined.

For a set $\{G_e \mid e \in E(t_<)\}$, by a *D-minimum* ***set*** *of embeddings* we call a set $\{s_e\colon G_e \to S_t\}$ of embeddings satisfying property (*), with D coordinating the block B_t, and having the smallest B_t-cost among all sets $\{r_e\}$ that satisfy these two conditions. We *denote* this B_t-cost by $c(t,D)$ and *denote* this D-minimum set by $s(t,D)$.

Denote by $c(t)$ the minimum of $c(t,D)$ over all D specified above, which is attained at some D_0, and *define* $s(t) = s(t,D_0)$. If $\mid B_t \mid = 1$, then $D = \emptyset$. If t is a bridge, then $s(t)$ is called a *bridge set*, and its elements are referred to as *bridge embeddings*.

The result of Theorem 3 consists of constructing a set $s(r)$ with r a root pipe in S such that $s(r)$ consists of a single embedding, which is the desired *ILS-minimum continuation* s': $G \to S$.

Algorithm for *ILS*-minimum embedding of a tree *G* into a network *S*. By induction, during a forward pass of the algorithm we construct $C(e,t,D)$, and during its backward pass we *could* construct $s(e,t,D)$, though the latter is not needed. The usual linear order on these pairs is fixed: pipes t from the leaves to the root, and for each fixed t we look through edges e from the leaves to the root.

For each pair (e,t), with properties $s(e_<) \subseteq t_<$ (if t is not a bridge) or $e \in E(t_<)$ (otherwise), and **for each set *D* of hybrid pipes in B_t**, we find a D-minimum embedding $s(e,t,D)$ of $G_e \to S_t$ and its B_t-cost, or notice that they are undefined. We denoted this B_t-cost by $C(e,t,D)$, and for this embedding, *denote* the **set of non-empty pipes in B_t** by $N(e,t,D)$. Additionally, for each pair (e,t) we compose a list $D(e,t)$ of admissible sets D.

For an arbitrary set M of pipes, define $f(M) = \sum\limits_{t \in M} l(t)$. By Claim 2, the set $N(t,D)$ can be constructed an algorithm which is linear in $\mid S \mid$ given the set D; this allows, for any $t \in S$ and for any sets D_1 and D_2, to compute in time of the order of $\mid S \mid^2 \cdot 2^{4k}$ the values of $f(N(t,D_1) \cap N(t,D_2))$ and $f(N(t_1,D_1) \cap N(t_2,D_2))$, where t_1 and t_2 are children of t. It is specifically these intersections that are used in what follows. The flow chart of the algorithm is shown in Figure 14.

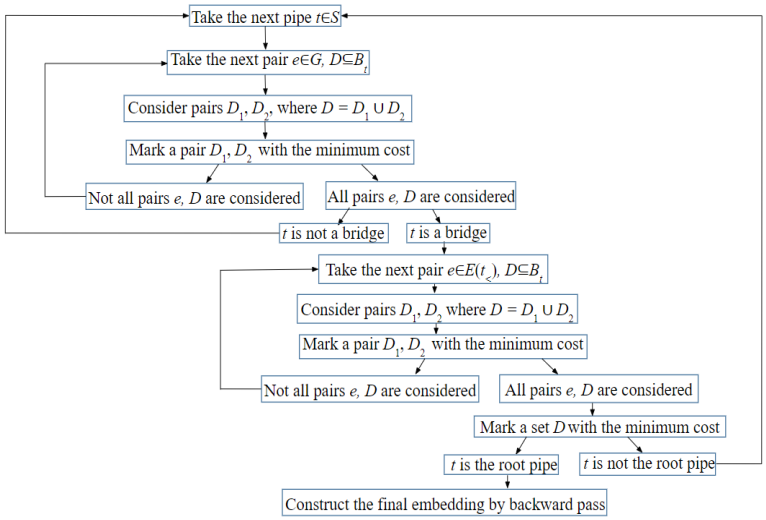

Figure 14. The flow chart of the algorithm for Theorem 3.

The induction base for constructing D-minimum embedding and bridge embedding of $G_e \to S_t$: for a leaf pipe $t_- \in B_t$ and a leaf edge $e \in G$, $D(e,t) = \{\emptyset\}$ and the embeddings are trivial, its B-cost being $C(e,t,D) = i_l \cdot (l(t)–1)$.

The induction step for D-minimum embeddings. Consider the following four cases, analogous to those considered in Section 2.4, where $U(e,t,D)$, by the proof given below, will be found to be equal to the B-costs of the embeddings defined below. In Cases 1–4 below, computations are performed if all the involved quantities are well defined and all the required conditions are fulfilled. Recall that D is a set of hybrid pipes in B.

(1) Assume that edge e leaves t and "swings" to the child pipe t_1 at a furcation in S, and $t_1 \in D$ (if t_1 is a hybrid pipe) or $e \in E(t_{1<})$, $D = \emptyset$ (if t_1 is a bridge) or D is arbitrary (if t_1 is a tree-type pipe and is not a bridge). The embedding continues (since $e_+ \mapsto t$) an already known $(D\backslash t_1)$-minimum (if t_1 is not a bridge) or bridge (otherwise) embeddings of $G_e \to S_{t1}$. Its B-cost is, respectively, $U(e,t,D) = U(e,t_1,D\backslash t_1) + e_l + i_l \cdot (l(t)–1)$ if $D\backslash t_1 \in D(e,t_1)$, or $U(e,t,D) = e_l + i_l \cdot (l(t)–1)$, symmetrically for t_2. For each e, t, and D, if, in this case or in one of the cases below, we have $U(e,t,D) < \infty$, then we add D to the list $D(e,t)$.

(2) Assume edge e forks in t into e_1 and e_2. Exhaustively examine all pairs (D_1,D_2) where D_1 and D_2 are any sets of hybrid pipes in B, such that $D_1 \in D(e_1,t)$ and $D_2 \in D(e_2,t)$. For each $D' = D_1 \cup D_2 \subset B_t$ we set the initial value of $U(e,t,D') = \infty$. For every (D_1,D_2) compute

$$V = U(e_1,t,D_1) + U(e_2,t,D_2) + c_i \cdot f(N(e_1,t,D_1) \cap N(e_2,t,D_2)) + c_d + i_l \cdot (l(t) - 1).$$

If V is strictly less than the current value of $U(e,t,D')$ for $D' = D_1 \cup D_2$, set $U(e,t,D') = V$. In particular, we compute $U(e,t,D)$.

(3) Assume that edge e forks at a furcation in S, t_1 is entered by edge e_1, and t_2 is entered by e_2. For each D' we set the initial value of $U(e,t,D') = \infty$. If t_1 and t_2 are not bridges, exhaustively examine all pairs (D_1,D_2), where D_1 and D_2 are sets of hybrid pipes in B, such that $D_1 \in D(e_1,t_1)$, $D_2 \in D(e_2,t_2)$. For every (D_1,D_2) compute

$$V = U(e_1,t_1,D_1) + U(e_2,t_2,D_2) + c_i \cdot f(N(e_1,t_1,D_1) \cap N(e_2,t_2,D_2)) + i_l \cdot (l(t) - 1).$$

If V is strictly less than the current value of $U(e,t,D')$ for $D' = D_1 \cup D_2 \cup \{t_1,t_2\}$, set $U(e,t,D') = V$. Here, by the definition, the set $\{t_1,t_2\}$ contains only hybrid pipes.

If exactly one of the pipes t_1 and t_2 is a bridge (assume that this is t_1), exhaustively examine each sets D_2 of hybrid pipes in B such that $D_2 \in D(e_2,t_2)$. For D_2, if $e_1 \in E(t_{1<})$ compute $V = U(e_2,t_2,D_2) + i_l \cdot (l(t) - 1)$, and if V is strictly less than the current value of $U(e,t,D_2 \cup \{t_2\})$, set $U(e,t,D_2 \cup \{t_2\}) = V$. Here, similarly, the set $\{t_2\}$ contains only hybrid pipes.

If both pipes t_1 and t_2 are bridges, $e_1 \in E(t_{1<})$, and $e_2 \in E(t_{2<})$, set $U(e,t,\emptyset) = i_l \cdot (l(t) - 1)$. In particular, we compute $U(e,t,D)$.

(4) Assume that edge t' leaves t, enters its unique child pipe t_1, and $t_1 \in D$ (if t_1 is a hybrid pipe); $e \in E(t_{1<})$ and $D = \emptyset$ (if t_1 is a bridge). Then, the embedding continues (since $e_+ \mapsto t$) the already known $(D\backslash t_1)$-minimum (if t_1 is not a bridge) or bridge (otherwise) embedding $G_e \to S_{t1}$, and $U(e,t,D) = U(e,t_1,D\backslash t_1) + i_l \cdot (l(t)–1)$ if $D\backslash t_1 \in D(e,t_1)$, or $U(e,t,D) = i_l \cdot (l(t)–1)$.

Choose an appropriate case that has the minimum B-cost $U(e,t,D)$ and memorize it, as well as the sets D_1 and D_2 in Cases 2 or 3. These will be used in the further backward pass of the algorithm, which will result in the construction of a $s(e,t,D)$ embeddings. There is no need to compute $U(e,t,D)$ for e, t, and D; these values are needed only for arguments satisfying the given conditions, which are necessary to proceed with the next cases of the algorithm. As is proved below, $C(e,t,D) = U(e,t,D)$; accordingly, the backward pass of the algorithm finds the embeddings $s(e,t,D)$: $G_e \to S_t$.

(5) The induction step for bridge embeddings after the corresponding construction step for D-minimum embeddings. Let t be a bridge. By property (*) (see the Proof of Theorem 3 below), t is entered by precisely the edges from $E(t_<)$. Arbitrarily order edges e_1, e_2, e_3, … ,$e_m \in E(t_<)$. Similarly to what was performed in Case 2 (but without the c_d term), by exhaustively examining pairs D_1, D_2 of sets of hybrid pipes in B, where $D_1 \in D(e_1,t)$ and $D_2 \in D(e_2,t)$, for every $D = D_1 \cup D_2$ we find $D_1{}'$ and $D_2{}'$ that correspond to a D-minimum two-element **set** $M_2 = \{G_{e1} \to S_t, G_{e2} \to S_t\}$ of embeddings. More precisely, for each D we minimize over D_1 and D_2, where $D = D_1 \cup D_2$, the quantity V in the expression from Case 2 without the c_d term, i.e.,

$$V = U(e_1,t,D_1) + U(e_2,t,D_2) + c_i \cdot f(N(e_1,t,D_1) \cap N(e_2,t,D_2)) + i_l \cdot (l(t) - 1.$$

Assume that the minimum of V is finite and is attained at some D_1' and D_2'. *Denote* it by $U(e_1,e_2,t,D)$ and add D to the list $D(e_1,e_2,t)$ of admissible sets. Note that the set $M_2 = \{s(e_1,t,D_1'), s(e_2,t,D_2')\}$ itself is not used here. Let $D_1' = f_{21}(D)$ and $D_2' = f_{22}(D)$. Again examining pairs D_1, D_2, where $D_1 \in D(e_3,t)$ and $D_2 \in D(e_1,e_2,t)$, for every $D = D_1 \cup D_2$ we find D_1'' and D_2'' that correspond to a D-minimum 3-element **set** $M_3 = \{G_{e1} \to S_t, G_{e2} \to S_t, G_{e3} \to S_t\}$ of embeddings. More precisely, for each D we minimize over D_1 and D_2, where $D = D_1 \cup D_2$, the quantity

$$V = U(e_3,t,D_1) + U(e_1,e_2,t,D_2) + c_i{\cdot}f(N(e_3,t,D_1) \cap N(e_1,e_2,t,D_2)) + i_l \cdot(l(t) - 1.$$

Assume that the minimum is finite and is attained at some D_1'' and D_2''. *Denote* it by $U(e_1,e_2,e_3,t,D)$ and add D to the list $D(e_1,e_2,e_3,t)$ of admissible sets. Note that the set $M_3 = \{s(e_1,t,f_{2,1}(D_1'')), s(e_2,t,f_{2,2}(D_1'')), s(e_3,t,D_2'')\}$ itself is note used here. Proceeding further in this way with computing $U(e_1,e_2, \dots\ e_i,t,D)$, for each D we construct a D-minimum set $M_m = \{G_{e1} \to S_t, G_{e2} \to S_t, \dots, G_{em} \to S_t\}$ of embeddings, which we denote by $s(t,D)$. The set $s(t,D)$ contains exactly $m = |E(t_<)|$ embeddings, where m is the number of edges entering t.

Choose a bridge set $s(t) = s(t,D_0)$ with the minimum $U(e_1,e_2, \dots\ e_m,t,D)$ over all D. For $t = r$, the root pipe, this set $s(r)$ consists of a single element, which is the desired embedding s': $G \to S$ with the minimum B-cost. The *ILS*-minimum cost s' equals the sum over all bridges $t \in S$ of B_t-costs of sets $s(t)$ plus the costs of *ILS* events in t, where B_t corresponds to t. □

Proof of Theorem 3. To prove the exactness of the algorithm, we first show that the expressions for computing $U(e,t,D)$ in the algorithm output the B_t-cost of the corresponding embeddings, which, clearly, D-coordinate the block B_t. For Cases 1 and 4, note the following, the embedding $G_e \to S_t$, as compared to the embedding $G_e \to S_{t1}$, has one non-empty pipe (t_1) more and one entry (into t_1) more. Therefore, the first *ILS* term is the same for both embeddings. This is also true for the second *ILS* term as well, since it equals 0 at t. For Case 3, the embedding $G_e \to S_t$, as compared to the union of embeddings $G_{e1} \to S_{t1}$ and $G_{e2} \to S_{t2}$ has two non-empty pipes (t_1 and t_2) more and two entries (into t_1 and t_2) more. Therefore, the first *ILS* term of the embedding $G_e \to S_t$ and of the union $G_{e1} \to S_{t1}$ and $G_{e2} \to S_{t2}$ are the same if pipes that are non-empty for both embeddings of the union are counted once each time. Similarly, the second *ILS* term of the embedding $G_e \to S_t$ and of the union are the same if pipes that are non-empty for the embeddings are counted $l(t)-1$ times. This explains the presence of the term $c_i{\cdot}f(N(e_1,t_1,D_1) \cap N(e_2,t_2,D_2))$ or, in Case 2, $c_i{\cdot}f(N(e_1,t,D_1) \cap N(e_2,t,D_2))$ in the expressions for C. Additionally, for Case 2 let us check that in the expression for V we may use the values $U(e_1,t,D_1) = C(e_1,t,D_1)$ and $U(e_2,t,D_2) = C(e_2,t,D_2)$, i.e., regard the embeddings $G_{e1} \to S_t$ and $G_{e2} \to S_t$, respectively, to be D_1-minimum and D_2-minimum. Indeed, the minimum B_t-cost is attained at some embedding s: $G_e \to S_t$, to which there correspond some embeddings s_1: $G_{e1} \to S_t$ and s_2: $G_{e2} \to S_t$ with some D_1 and D_2. If, for example, s_1 is not D_1-minimum, then by replacing it with a D_1-minimum one, we strictly decrease the B_t-cost of the embedding s, a contradiction. Note the following, here we use the fact that $N(t,D_1)$ is independent of the embedding $G_{e1} \to S_t$ that D_1-coordinates the block B_t. Correctness of using the values $U(e_1,t_1,D_1) = C(e_1,t_1,D_1)$ and $U(e_2,t_2,D_2) = C(e_2,t_2,D_2)$ in Case 3 is prove similarly. All arguments justifying Case 2 can be literally repeated to justify the fact that in the induction step for bridge embeddings, every M_i is D-minimal and $U(e_1,e_2, \dots\ e_i,t,D)$ is its B_t-cost. In particular, this is true for a bridge set M_m.

Using induction from leaves to the root, let us prove that for any bridge t the constructed bridge set $s(t)$ is *ILS* minimal; then, in particular, so is s'. For leaf pipes, this is obvious; let t be a non-leaf pipe. The total cost of events for $s(t)$ can be represented as a sum of three terms, the B_t-cost, the total cost of events in subnetworks $S_{t'}$ for bridges t' leaving B_t, and the cost of the *ILS* event in t. The construction of $s(t)$ implies the minimality of the first of these terms. The induction hypothesis on the *ILS* minimality of bridge embeddings

for t' satisfying property (*) implies the minimality of the second term. Property (*) for t implies that the third term is constant and equals $c_i \cdot (|E(t_<)| - 1) \cdot l(t)$.

Let us check property (*) for the embedding s'. Embedding $s(e,t,D)$ satisfy property (*)′: if e' enters a bridge t', then $e' \in E(t'_<)$. Indeed, the conditions in Steps 1–4 ensure this property for bridges leaving B. After that, it can be proved by induction on the construction of $s(e,t,D)$. Since any element of a bridge set is one of the $s(e,t,D)$, property (*)′ is satisfied for bridge sets too, in particular, for s'. Now (*) follows from the fact that for any bridge t and any $e \in E(t_<)$ a path in G from the root to a leaf passing through e contains an edge entering t.

The algorithm runtime. For each pair (e,t) in the construction of a D-minimum embedding we look through at most 2^k sets D, or at most 4^k pairs (D_1,D_2). If t is a bridge pipe, it additionally requires at most $|E(t_<)| \cdot 2^{4k}$ pairs (D_1,D_2), which yields the overall estimate of the order of $|G| \cdot |S| \cdot k \cdot 2^{4k}$. □

Remark 1. *The minimum embeddings of a tree into a network, taking ILS events into account or disregarding them, can be different. For example, let a tree be $G = (l_1,l_2)$, and let a network S be the network shown in Figure 13 with the following modifications: at the bottom there is a single leaf l, and side branches in S are continued to the root (two rhombuses are added). Then, a minimum embedding $G \to S$ without taking into account ILS is shown in Figure 13a (no duplications and losses, one ILS), and a minimum embedding taking into account ILS (if the duplication cost is less that the ILS cost) is shown in Figure 13b (no ILS and no losses, one duplication).*

Remark 2. *The problem of constructing a minimum embedding s': $G \to S$ can also be posed without the assumption that the embedding is a continuation of some leaf mapping s. If we disregard ILS events in this problem, then such an embedding can be constructed in a standard way: by exhaustively examining pairs (edge e, pipe t) and choosing for each pair the best of the four cases given in Section 2.4; here, the condition $s'(e_<) \subset t_<$ cannot be used. The authors are unaware of any polynomial algorithm for constructing a minimum embedding s' in this setting with taking into account the ILS (even if S is a tree).*

2.8. Example of Executing the Algorithm

Let us demonstrate the operation of the algorithm by an example of a network S with root r and a tree $G = (c,(a,d)),b)$ with root edge r shown in Figures 6 and 15. Let the duplication cost be 3, loss cost be 2, and the *ILS* event cost be 5. Denote the blocks $B_1 = \{u_1,u_2,u_3,u_4,u_6\}$ and $B_2 = \{u_5,u_7,u_8,u_9,u_{10},r\}$. The corresponding sets of hybrid pipes are $H_1 = \{u_3\text{-}u_1, u_4\text{-}u_1, u_3\text{-}u_2, u_4\text{-}u_2\}$ and $H_2 = \{u_7\text{-}u_5, u_{10}\text{-}u_5, u_9\text{-}u_8, r\text{-}u_8\}$. In Figure 15, an *ILS* minimum embedding s': $G \to S$ is shown. Namely, edges in G are shown in red along the pipe in S. In particular, the edge e_2 terminates with a duplication. The *ILS* cost of s' is the sum of the B_1-cost, B_2-cost, and the *ILS* cost in the pipe $u_7\text{-}u_6$, i.e., it is 8 + 9 + 1 = 18.

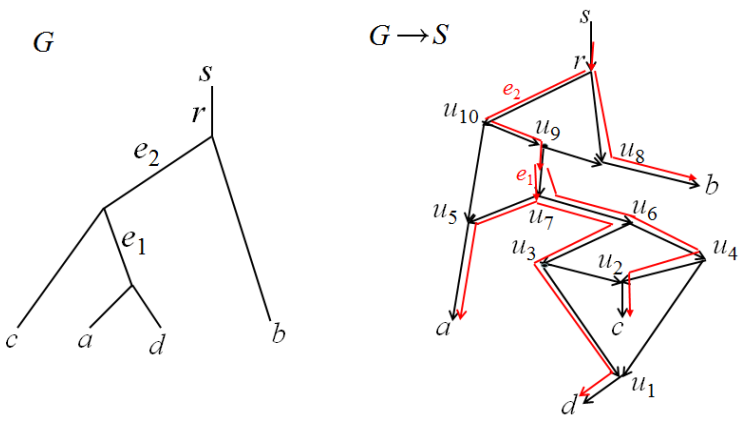

Figure 15. Tree G and its embedding s' (shown in red) in the network S constructed by our algorithm. In the pipe u_7 there is a duplication. The network S was shown in Figure 6.

The algorithm exhaustively examines pipes $t \in B_1$. Among them, only the leaf edges c and d in G can satisfy the condition $s'(e_<) \subset t_<$. For the pairs (c,t) and (d,t) there may exist at most one set D, depending on whether this condition is satisfied. If D is well defined, there exists an embedding $s(c,t,D)$ or $s(d,t,D)$. For instance, $D = \{u_3\text{-}u_1\}$ for the pair $(d,u_6\text{-}u_3)$. Then, the algorithm constructs D-minimum embeddings for the bridge $t = u_7\text{-}u_6$. For it, only the edges c and d satisfy condition (*). For each of the pairs, (c,t) and (d,t), there are two sets D for which the embeddings $s(c,t,D)$ and $s(d,t,D)$ are well defined. For instance, these are the sets $\{u_3\text{-}u_1\}$ and $\{u_4\text{-}u_1\}$ for the pair (d,t). For them, the embeddings are trivial. The same for the pair (c,t).

Then, the algorithm constructs a bridge set for the bridge $t = u_7\text{-}u_6$. To this end, the algorithm orders the edges in $E(t_<) = \{c,d\}$. As was already noted, for each of the pairs (c,t) and (d,t) there exist two sets D with the corresponding embeddings $s(c,t,D)$ and $s(d,t,D)$. The algorithm examines four pairs (D_1,D_2): $(\{u_3\text{-}u_2\},\{u_3\text{-}u_1\})$, $(\{u_4\text{-}u_2\},\{u_4\text{-}u_1\})$, $(\{u_3\text{-}u_2\},\{u_4\text{-}u_1\})$, and $(\{u_4\text{-}u_2\},\{u_3\text{-}u_1\})$. The first pair yields the pipe $u_6\text{-}u_3$, which is non-empty for both embeddings, $s(c,t,D_1)$ and $s(d,t,D_2)$. The second pair yields the pipe $u_6\text{-}u_4$, which is non-empty for both embeddings $s(c,t,D_1)$ and $s(d,t,D_2)$. The third and fourth pairs do not yield pipes that are non-empty for both embeddings. Thus, the first two pairs define a bridge set with an *ILS* event, and the last two, without it. Respectively, the B_1-cost of the first two bridge sets is 13 (four losses and an *ILS*), and that of the last two is 8 (four losses). Of the two last pairs, the algorithm chooses any one; in Figure 15, the pair $(\{u_4\text{-}u_2\},\{u_3\text{-}u_1\})$ is chosen.

The algorithm exhaustively examines pipes in B_2. For pipe t in the set $\{u_7\text{-}u_5$, $u_{10}\text{-}u_5$, $u_9\text{-}u_8$, $r\text{-}u_8\}$ and any e, the embeddings $s(e,t,D)$ are trivial and well defined only for $D = \emptyset$ or are not well defined. Consider the pipe $t = u_9\text{-}u_7$. For it, the edges a, c, d, e_1, and e_2 in G satisfy condition $s'(e_<) \subset t_<$. For the first three of them, the embeddings are trivial. For e_1, the embedding is well defined only for $D = \{u_7\text{-}u_5\}$, and the algorithm chooses Case 3 (forking at a furcation) with the minimum B_2-cost 0. For e_2, the embedding is well defined only for $D = \{u_7\text{-}u_5\}$, and the algorithm chooses Case 2 (duplication in t), referring to the pairs (c,t) and (e_1,t) already considered. The B_2-cost of the embedding $s(e_2,t,D)$ is 2 + 0 + 3 = 5. Consider the pipe $t = u_{10}\text{-}u_9$. For it, all $e \in G$ satisfy the condition $s'(e_<) \subset t_<$. For leaf edges, the embeddings are trivial. For e_1 the embedding is well defined only for $D = \{u_7\text{-}u_5\}$; the algorithm chooses Case 1 referring to the pair $(e_1,u_9\text{-}u_7)$ already considered; and the B_2-cost is 2. For e_2, the set D and the chosen case are the same; the B_2-cost is 7. For r, the only set is $D = \{u_7\text{-}u_5,u_9\text{-}u_8\}$; the algorithm chooses Case 3 referring to the pairs $(e_2,u_9\text{-}u_7)$ and $(b,u_9\text{-}u_8)$ already considered; and the B_2-cost is 7 + 0 = 7. Consider the pipe $t = r\text{-}u_{10}$. For this, all $e \in G$ satisfy the condition $s'(e_<) \subset t_<$. For the leaf edges b, c, and d, the embeddings are trivial. For the edge a, we have two sets, $D = \{u_{10}\text{-}u_5\}$ and $D = \{u_7\text{-}u_5\}$; for each of them, the embedding is trivial. For e_1, we have the same two sets D; for the first of them, the algorithm chooses Case 3, and for the second, Case 1; both B_2-costs are 4. For e_2 we have $D = \{u_7\text{-}u_5\}$ and Case 1; the B_2-cost is 9. For r we have $D = \{u_7\text{-}u_5$, $u_9\text{-}u_8\}$ and Case 1; the B_2-cost is 9. Consider the root pipe $t = s\text{-}r$. For it, we consider the root edge r only. We have two sets D: $D_1 = \{r\text{-}u_8,u_7\text{-}u_5\}$ and $D_2 = \{u_9\text{-}u_8,u_7\text{-}u_5\}$. For D_1, the algorithm chooses Case 3 with B_2-cost 9; for D_2, Case 1 with B_2-cost 11. Since the B_2-cost for D_1 is smaller, as a resulting s' the algorithm chooses the embedding $s(r,s\text{-}r,D_1)$; Figure 15.

3. Reconstruction of Structures along a Tree

3.1. Problem Setting and Main Results

A *structure* is a directed graph consisting of paths (excluding isolated nodes) and cycles, including loops. Its edges are assigned with *unique names*; one may assume the names to be positive integers. An *extremity* is a tail or a head of an edge. A *join* of two nodes consists of identifying them, and a *cut* is the operation inverse to the join (see Figure 16). Sometimes, instead of *join* (two nodes) they say *merge*, *identify*, or *glue*.

Figure 16. Join (from left to right) and cut (from right to left) of extremities x and y of edges i and j in a structure. The join of the extremities x and y (as two separate nodes in the structure) consists of identifying them as a node z, and the cut means that a node z in a structure is replaced with separate extremities x and y of edges i and j.

There are four operations over a structure, which are called *SCJ* (Single-Cut-or-Join) *operations*: *cut* of its node of degree 2, *join* of two its nodes of degree 1, *deletion* of an isolated edge, and *insertion* of an isolated edge. Each operation is assigned a strictly positive rational number called the *cost* of the operation. The *SCJ distance* from a structure a to a structure b is the minimum total cost of operations required to transform a into b. It is easily seen that the *SCJ* distance equals the weighted sum of the number of pairs of extremities joined in a but not in b, joined in b but not in a, the number of edges that are present in a but not in b, and the number of edges that are present in b but not in a. The *SCJ* distance is a measure of the evolutionary distance between two structures and is used to construct evolutionary trees and infer evolutionary relationships. The lower the *SCJ* distance between two structures, the more closely related they are thought to be in terms of their evolutionary history. The same is true for the *DCJ* distance. The *SCJ*, as well as the *DCJ* (Double-Cut-and-Join), operations are well known, and many dozens of papers are devoted to their study, see, e.g., the survey [6] (Section 3).

Additionally, there is another set of operations over a structure, now consisting of six operations, which, in the context of the present paper, will be called *DCJ operations*, though, usually, this term refers to the first four of them. These are the following operations: *cut* of a node of degree 2 of the structure, *join* of two its nodes of degree 1, double and sesquialteral *intermerging*, *deletion,* and *insertion* of a connected fragment of edges. The first two operations are the same as above. The last two are the following, deleting a whole component of a structure (path or cycle) or adding a fragment as a separate component; another option is as follows, such a fragment is cut at its extremities, then it is deleted, and the remaining extremities (if they are two) are joined again; respectively, a node of degree 2 in a structure is cut (or it is a node of degree 1) and the fragment is joined by its extremities (or its extremity) to the obtained extremities (or to the existing extremity). These two operations are generalizations of the corresponding *SCJ* operations. The third and fourth operations are compositions of the first two. *Double intermerging* (denoted by *DM*) consists of cutting a pair of nodes of degree 2 and joining the extremities thus formed with each other. *Sesquialteral intermerging* (denoted by *SM*) consists of cutting a node of degree 2 and joining one of the extremities thus formed with a node of degree 1. Quite similarly to the above, each of these operations is assigned with a cost, and the *DCJ distance* between structures a and b is defined. These operations have long been studied; their detailed description with corresponding figures and historical references can be found in a large number of papers and surveys, e.g., in [6,24,25]. Usually, only four operations are called *DCJ* operations, Double Intermerging, Sesquialteral Intermerging, and two Ordinary Intermergings (Cut and Join). To a cyclic structure, only the Double Intermerging operation can be applied, because only this operation, being applied to such a structure, preserves its cyclicity and equal content property.

A structure is called *cyclic* if it consists of cycles only. A set of structures is said to be *of equal content* of names if all structures in it have the same set of names.

For any tree, an *arrangement* over it is an assignment of a structure to each interior node of the tree. Arrangements are considered to be specified at leaves; therefore, after specifying them at interior nodes, an arrangement will be set throughout the tree.

The **problem of reconstruction** is to find structures at interior nodes of the tree so that the **resulting arrangement is a minimum** of a given functional defined on all arrangements over the given tree. Here, as such a functional, we consider the sum of lengths of all edges of a tree, where the length of an edge is the distance between the structures at its endpoints.

As a distance, we consider the *SCJ* or *DCJ* distance; where each edge is directed from the root to the leaves of the tree. The value of this functional is called the *cost of the arrangement*. An arrangement with the lowest cost is said to be *minimum*, and its cost is referred to as the *minimum cost*, which is also sometimes called the *global minimum*. An arrangement is said to be *locally minimum* if replacing a structure at any interior node of the tree with any other does not reduce the cost of the arrangement.

A star-like tree (or simply a *star*) is a tree consisting of a root and leaves to which the root is connected. A *two-star tree* is a tree with two interior nodes (denote them by r and u), such that the root r is connected with arbitrarily many leaves and u is connected with only two leaves (see Figure 17).

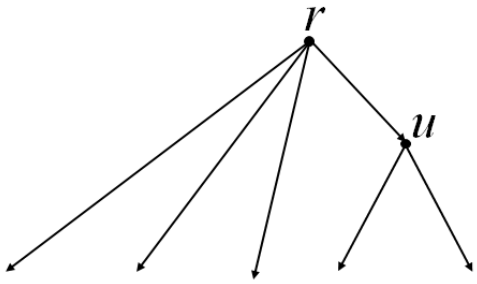

Figure 17. Two-star tree. Every edge of the tree is directed from the root to leaves.

Setting of the *SCJ* and *DCJ* reconstruction problems. Given: a rooted tree, not necessarily binary, to each leaf of which there is assigned a *loopless* structure; and arbitrary strictly positive costs of all *SCJ* or all *DCJ* operations. Find: an arrangement (at interior nodes of the tree) such that the sum over all edges of the tree of *SCJ* or *DCJ* distances between structures at the endpoints of an edge is minimum. These essentially different reconstruction problems are referred to as, respectively, *SCJ* or *DCJ reconstructions*.

We distinguish a ***cyclic*** reconstruction problem, where all structures specified at the leaves are cyclic, and only cyclic structures can be attributed to internal nodes (i.e., only cyclic arrangements are considered). Such reconstruction problems will be called *cyclic*. In a cyclic *DCJ* reconstruction, the *SM*, join, and cut operations cannot be applied, and, accordingly, in the set of *DCJ* operations there remain the *DM* operation and insertion/deletion of a component of a structure.

Another particular case of the reconstruction problem is as follows; structures at the leaves are of equal content, and only arrangements with the same set of names as for the structures at the leaves are considered. In this case, we speak about the ***equal-content*** reconstruction problem. In the cyclic equal-content *DCJ* reconstruction, in addition, the insertion/deletion operations cannot be applied, and accordingly, in the set of *DCJ* operations there remains the *DM* operation only. In this case, the cost of this operation can be omitted without loss of generality.

In the reconstruction problems, we prove the following theorems, where n is the number of names in the structures assigned to the leaves and l is the number of leaves.

Theorem 4. *An exact algorithm is constructed which solves the SCJ reconstruction problem on any two-star tree for arbitrary costs of SCJ operations. The runtime of the algorithm is of the order of* $n^2 \cdot (l + log(n))$.

Recall that the *SAT problem* for a given conjunctive normal form (CNF) is the problem of determining whether there exist values 0 and 1 of the variables in this CNF for which the CNF equals 1, and if they do exist, giving an example of those values.

Theorem 5. *An exact algorithm is constructed which reduces the cyclic equal-content DCJ reconstruction problem on any star to a sequence of length* $log(n \cdot l)$ *of SAT problems in which every CNF contains of the order of* $n^2 \cdot l^2$ *variables and of the order of* $n^3 \cdot l^2$ *disjunction terms.*

Numerous references concerning this problem can be found, e.g., in [24]. Let us present results directly related to Theorems 4 and 5. In [10], for ***equal costs*** of *SCJ* operations and equal-content structures, there was described an exact quadratic-time algorithm for structure reconstruction on an arbitrary tree, loops at leaves being admissible. In [26], for ***equal costs*** of *SCJ* operations and arbitrary content of structures, there was described an exact quadratic-time algorithm for structure reconstruction on an arbitrary tree (loops being admissible). In [11], for ***arbitrary costs*** of *SCJ* operations and arbitrary content of structures (but with the condition that structures do not contain loops), there was described an exact algorithm for structure reconstruction ***on a star***. Thus, in Theorem 4 we pass from a star to a two-star tree.

In [24], there was considered a generalization of the *SCJ* reconstruction problem where each pair of extremities is assigned with a weight, and when this pair is joined, its weight is added to the objective function. Additionally, it was proved there that this problem is NP-hard, and an FTP-algorithm for solving it was described. Namely, there was an algorithm described with a runtime being exponential in one variable only; this variable reflects the number of conflicts at interior nodes of the tree when reconstructing by each pair of extremities separately (for exact formulations; see [24]). The algorithm is based on reduction to an integer linear programming (ILP) problem.

In Theorem 5, the cyclic equal-content *DCJ* reconstruction problem on a star is reduced to a sequence of SAT problems. Note that the reduction to a SAT solver instead of reduction to an ILP-packet is much more efficient (see [27]).

3.2. SCJ Reconstruction Algorithm

Recall that we consider structures with *no loops at the leaves*. We will identify a node of a tree and a structure assigned to it. Additionally, we consider only edges contained in at least one leaf structure, such edges will be called *admissible*. Both constraints do not lose the generality of solution.

If an edge of a leaf structure is not present at some leaf (which is also a structure), we add it there as a loop; then the *SCJ* reconstruction problem reduces to the case of equal-content structures at leaves and at all interior nodes. Let the *cost of joining an isolated edge* into a loop be equal to the cost of edge deletion, and the *cost of cutting a loop* be equal to the cost of adding an isolated edge. Thus, there remain two operations only; join and cut of extremities of an admissible edge, but four costs for them are defined.

Denote the names of the extremities of the ith edge by i_1 (tail) and i_2 (head). There arises a one-to-one correspondence between structures and *matchings* on the *set* (later: a *graph*) M of names of extremities of all admissible edges. Namely, ***structure*** a ***corresponds*** to a matching P on M such that the extremities i_k and j_l are connected by an edge in P if, and only if, i_k and j_l are joined in a.

Let us be given a two-star tree (Figure 17). Recall that an edge of a tree is considered together with the direction from the root to the leaves. An *event* for an unordered pair $p = (i_k, j_l)$, $i_k \neq j_l$, on an edge of the tree is passing from being joined at one end of the edge to being cut at another end of it, or vice versa. A *p-scenario* is a specification of being joined/cut for a fixed pair p at all interior nodes of the tree; at the leaves, this specification is already given. The *cost* of a p-scenario is the total cost of events with the pair p according to this specification. Note that the set $\mathcal{S}$ of p-scenarios formed for all pairs p does not necessarily define an arrangement (at interior nodes) on the tree; even if $\mathcal{S}$ defines an arrangement at the tail i_1 of an edge, it need not define a structure in i_2, since at i_2 one extremity can be joined with two other extremities. Our efforts are aimed to rule out such a situation.

Let us describe the *SCJ* reconstruction algorithm. We say that an unordered pair p is *admissible* if its extremities are joined in at least one leaf of the tree. For each admissible pair p, independently of other pairs of edge extremities, we inductively compute (*forward pass* of the proposed algorithm from the leaves to the root) two quantities, p_{yes} and p_{no}. In each leaf v we set $p_{yes}(v) = 0$ if p is joined, and $p_{yes}(v) = \infty$ if it is not joined. Similarly, $p_{no}(v) = 0$ if p is not joined, and ∞ if it is joined.

In the induction step, we compute:

$\boldsymbol{p_{yes}}(v)$, the minimum cost of all p-scenarios starting at v *provided that the pair p is joined* at v, and

$\boldsymbol{p_{no}}(v)$, a similar quantity *provided that p is not joined at v.*

For that, during the forward pass of the algorithm, for every next node v of the tree we consider all its child nodes. For each child v_1 of v, we know $p1_{yes}$ and $p1_{no}$; we compute $p_{yes} = \min(p1_{yes}, p1_{no} + c_1)$, where c_1 is the cost of the corresponding cut, and take the sum over all children. For instance, at node r (see Figure 17), we have $p_{yes}(r) = \sum_{v_i} \min(p_{yes}(v_i), p_{no}(v_i) + \boldsymbol{c_1})$, where v_i runs over all children of r. If the minimum is attained on the first argument, we record the pointer $p_{yes} \to yes$, and otherwise, $p_{yes} \to no$, where the first pointer means "join" of the extremities in p at v_1, and the second, "no join" of these extremities. Similarly, we compute $p_{no} = \min(p1_{yes} + c_2, p1_{no})$, where c_2 is the cost of the corresponding join, and take the sum over all children. For instance, at node r (the same figures), we have $p_{no}(r) = \sum_{v_i} \min(p_{yes}(v_i) + \boldsymbol{c_2}, p_{no}(v_i))$, where v_i runs over all children of r. If the minimum is attained at the first argument, we record the pointer $p_{no} \to yes$, and otherwise $p_{no} \to no$. Thus, a pair p at v is labeled by p_{yes}, p_{no}, and two pointers, one of them beginning with p_{yes} and the other with p_{no}. These pointers are used in the backward pass of the algorithm when passing from v to v_1.

We consider the set M of all extremities at one leaf of the tree (or, equivalently, at all leaves of the tree) as a loopless graph with all *admissible* unordered *edges p*. If the edges of the graph M are assigned with weights, which are rational or even integer numbers, then we *denote* by $w(x)$ the weight of any *matching* x on M; here, every x is a subgraph in M with nodes of degrees 0 or 1. In other words, a matching is a subset of edges in a given graph where no two edges share a common vertex. To each edge p in M, we can assign the weight $p_{yes} - p_{no}$ and construct a ***minimum weighted matching*** P, which is obtained by minimizing $w(x)$ over all matchings x in M. Thus, at the root r, the structure P_r is *defined* to be the minimum matching with *weights* $p_{yes} - p_{no}$ obtained at the *root node r* of the tree as a result of the forward pass of the algorithm.

Using the minimum matching P_r, for each edge p in M, by backward pass of the algorithm we uniquely determine whether the pair p is joined at u. Namely, if $p \in P_r$, then we use the pointer that begins with p_{yes}; if $p \notin P_r$, we use the pointer that begins with p_{no}. Namely, for $p_{yes} \to yes$ the pair p will be joined at u, and for $p_{yes} \to no$ it will not be joined at u. However, unlike the P_r, it is not clear whether in this way there will be defined a structure at u. In Lemma 1, it is shown that any extremity at u is joined with at most one extremity; thus, the structure P_r at r defines some structure P_u at u. Therefore, there arises an arrangement $\{P_r, P_u\}$ on the tree, which is said to be *final*; this is the output of the algorithm. Of course, this backward pass can be defined for any tree, but in this case Lemma 1 need not necessarily hold. The flow chart of the algorithm is shown in Figure 18.

Denote by $s(P)$ the set $\mathcal{S}$ of p-scenarios for all unordered admissible pairs p, according to which a pair p is joined at r if, and only if, $p \in P$, and a pair p is joined or not joined at u according to the above-mentioned backward pass. Here, we used P at r instead of P_r. *Denote* by $c(P)$ the total (over all such p) cost of p-scenarios in $s(P)$.

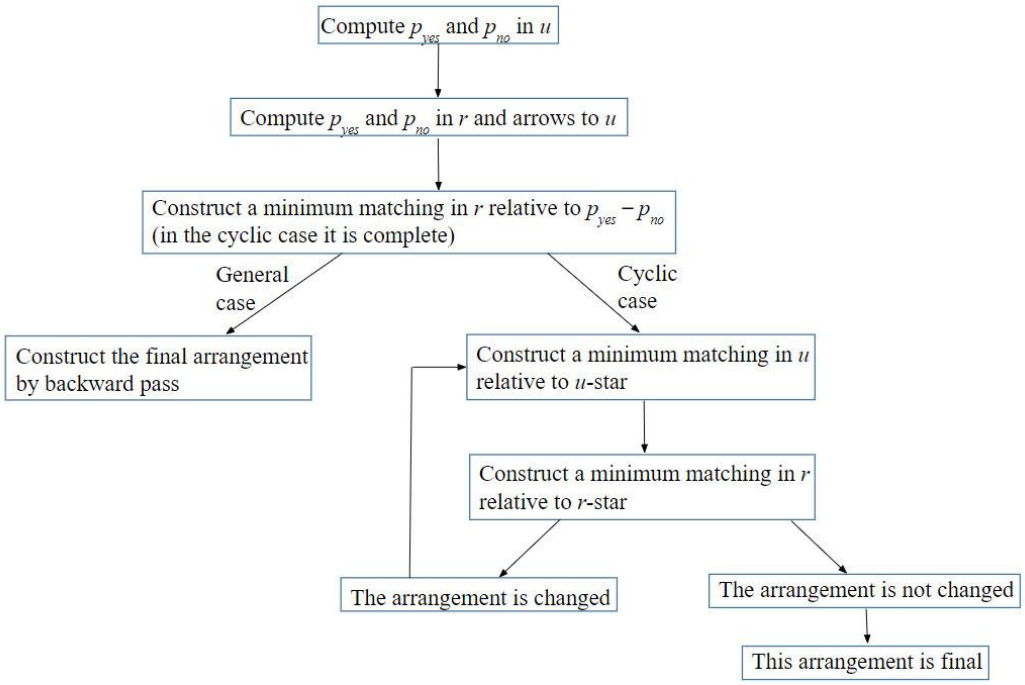

Figure 18. The flow chart of the algorithm for Theorem 4.

3.3. Exactness and Runtime of the SCJ Reconstruction Algorithm

Denote by P_0 the empty matching.

Lemma 1. *For any structure P at the root node r we have c(P) = c(P$_0$) + w(P). The minimum of c(P) is attained at P$_r$ which defines the minimum final arrangement on the tree.*

Here, *final* means that the backward pass of the algorithm defines exactly structures, i.e., any extremity at u is joined with at most one extremity. Thus, the algorithm defines a structure at u.

Proof. Any minimum arrangement does not have every unordered pair p joined if p is not joined in at least one leaf of the tree. Therefore, when finding the minimum arrangement, it is possible—as we have done—to confine ourselves only with pairs p that are joined in at least one leaf of the tree, i.e., to confine ourselves with admissible pairs p only.

Recall that $p \in P$ means that an edge p belongs to a matching $P \subseteq M$, and $p \notin P$ means that an edge p in M does not belong to P. The definition of p_{yes} and p_{no} at r implies that for any matching P in M we have

$$c(P_0) = \sum_{p \notin P_0} p_{no} = \sum_{p \in P} p_{no} + \sum_{p \notin P} p_{no},\ c(P) = \sum_{p \in P} p_{yes} + \sum_{p \notin P} p_{no}.$$

Then, $c(P) - c(P_0) = \sum_{p \in P} (p_{yes} - p_{no}) = w(P)$. Therefore, the minimum of $c(P)$ is attained at P_r, since P_r is a minimum matching.

To prove the second claim, assume that at u there are two joined pairs p_1 and p_2 incident to each other. At each of the three neighbors of u, at least one of the pairs p_1 and p_2 is not joined by the condition. Since there are exactly three neighbors, at least one of the pairs p_1 and p_2 (say p_1) is not joined at least at two neighbors of u. Then, by making p_1 not joined at u, one strictly reduces the cost of the p-scenario, which contradicts the minimality of $c(P_r)$. Thus, the backward pass defines an arrangement with cost $c(P_r)$ over a two-star tree. Assume that there is an arrangement with a cost strictly less than this. It defines some matching P at r. We have $c(P) < c(P_r)$, which contradicts the minimality of $c(P_r)$. □

Proof of Theorem 4. The proposed algorithm consists of the three above-described steps: forward pass, constructing a matching P_r, and backward pass, which defines P_u. The exactness of the algorithm follows from Lemma 1.

The runtime of the algorithm is the sum of the computation time for the three above-mentioned steps. Since the number of admissible p is not greater than nl, the runtime of the first and third steps is of the order of nl. A minimum matching is constructed in a time of the order of $n' \cdot m + (n')^2 \cdot \log(n')$, where n' is the number of nodes in the graph M and m is the number of edges in it, ([28] Chapter 11). Since $n' = 2n$ and $m \leq n \cdot l$, this time is of the order of $n^2 \cdot (l + \log(n))$. □

Remark 3. *The described algorithm obviously can be generalized to the case when there are several nodes u (see Figure 17), i.e., any child of the root of the tree is a leaf or a node with exactly two children, which are leaves. For this case, the Proof of Theorem 4 can be literally repeated.*

3.4. Example of Executing the SCJ Reconstruction Algorithm

A two-star tree and structures at its leaves are shown in Figure 19. Let the cost of all operations be 1.

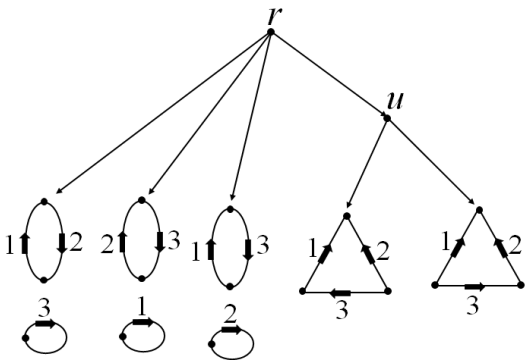

Figure 19. Example of executing the *SCJ* reconstruction algorithm: input data at the leaves (two components in each of the three left-hand leaves); all costs are uniform.

At node u we have $p_{yes}(u) = 0$, $p_{no}(u) = 2$ for $p = (1_2,2_2)$; and similarly $p_{yes} = 1$, $p_{no} = 1$ for $p = (3_i,1_1)$ and $p = (3_i,2_1)$, where i is arbitrary; $p_{yes} = 2$, $p_{no} = 0$ for other p. At node r we have $p_{yes}(r) = 3$, $p_{no}(r) = 1$ for $p = (1_2,2_2)$; $p_{yes} = 3$, $p_{no} = 2$ for $p = (3_2,1_1)$ and $p = (3_2,2_1)$; $p_{yes} = 4$, $p_{no} = 1$ for $p = (3_1,1_1)$ and $p = (3_1,2_1)$; $p_{yes} = 3$, $p_{no} = 1$ for the seven p that are joined at an r-leaf but not at a u-leaf; and $p_{yes} = 4$, $p_{no} = 0$ for other p.

Weights of all pairs are positive; therefore, the minimum matching is empty. Hence, there are no joins at r; following the arrows, we obtain a structure at u. The resulting arrangement is shown in Figure 20; its cost is 14.

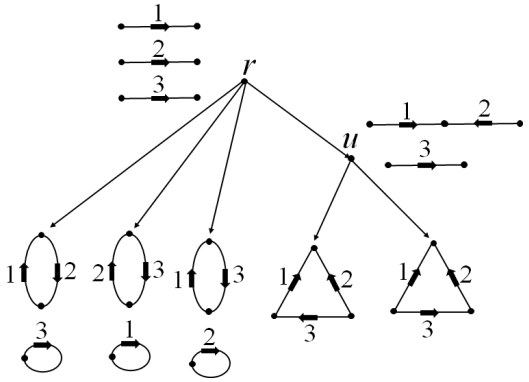

Figure 20. Result of executing the *SCJ* reconstruction algorithm (output arrangement): input data at the leaves are shown in Figure 19; all costs are uniform.

Now assume that the cost of a cut of any pair is 1, and the cost of a join is 4. At node u we have $p_{yes}(u) = 0$, $p_{no}(u) = 2$ for $p = (1_2,2_2)$; and similarly $p_{yes} = 1$, $p_{no} = 4$ for $p = (3_i,1_1)$ and $p = (3_i,2_1)$, where i is any; $p_{yes} = 8$, $p_{no} = 0$ for other p. At node r we have $p_{yes}(r) = 3$, $p_{no}(r) = 4$ for $p = (1_2,2_2)$; $p_{yes} = 3$, $p_{no} = 8$ for $p = (3_2,1_1)$ and $p = (3_2,2_1)$; $p_{yes} = 4$, $p_{no} = 4$ for $p = (3_1,1_1)$ and $p = (3_1,2_1)$; $p_{yes} = 3$, $p_{no} = 4$ for the seven p that are joined at an r-leaf but not at a u-leaf; and $p_{yes} = 4$, $p_{no} = 0$ for other p; a non-zero term arises only on the edge (r,u) for the pair $(1_2,2_2)$ when computing p_{no}®.

For weights of the edges in M we obtain the following: –1 for $p = (1_2,2_2)$; –5 for $p = (3_2,1_1)$ and $p = (3_2,2_1)$; 0 for $p = (3_1,1_1)$ and $p = (3_1,2_1)$; –1 for the seven p that are joined at an r-leaf but not at a u-leaf; and 4 for other p. A minimum matching has weight –7 and consists of the pairs $(3_2,1_1)$, $(1_2,2_1)$, and $(2_2,3_1)$. The resulting arrangement is shown in Figure 21, its cost being 49.

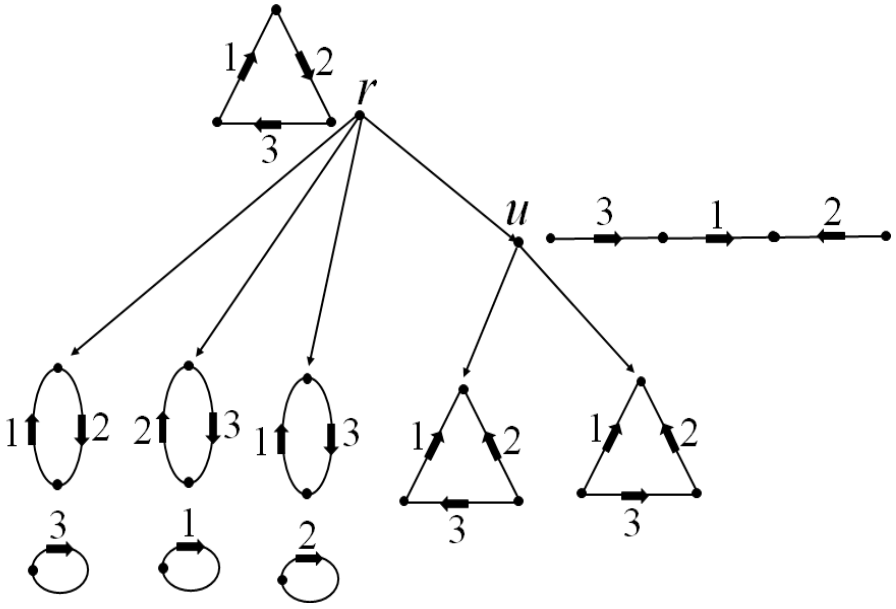

Figure 21. Result (output arrangement) of executing the *SCJ* reconstruction algorithm: input data at the leaves are shown in Figure 19, the cost of the cut is 1, and the cost of a join is 4.

3.5. Heuristic Algorithm for the Cyclic Case of the SCJ Reconstruction Problem on an Arbitrary Tree

The authors are unaware of whether an exact polynomial-time *SCJ* reconstruction algorithm for a structure over a two-star tree in the cyclic case is possible, even without regard to the costs. A heuristic algorithm for this case can easily be obtained by modifying the algorithm described in Section 3.2. Namely, for P_r we take a minimum complete weighted matching. After that, as described in [11], we could descend to a locally minimum arrangement by alternating replacements of complete matchings at u and at r, beginning from u. At a current node v, a matching is replaced with a minimum complete one with respect to the weights $p_c = p_{yes} - p_{no}$ of unordered pairs p (edges in M), where p_{yes} is the total cost of events with the pair p on edges incident to v provided that it is joined at v, and p_{no} is the similar cost provided that p is not joined at v.

Let all costs be uniform in the example presented in Figure 19; a minimum complete matching P_r is shown in Figure 20. One can easily check that the descent to a local minimum can be made in one step; the resulting arrangement is shown in Figure 22, its cost being 20. In this case, the local minimum is also global.

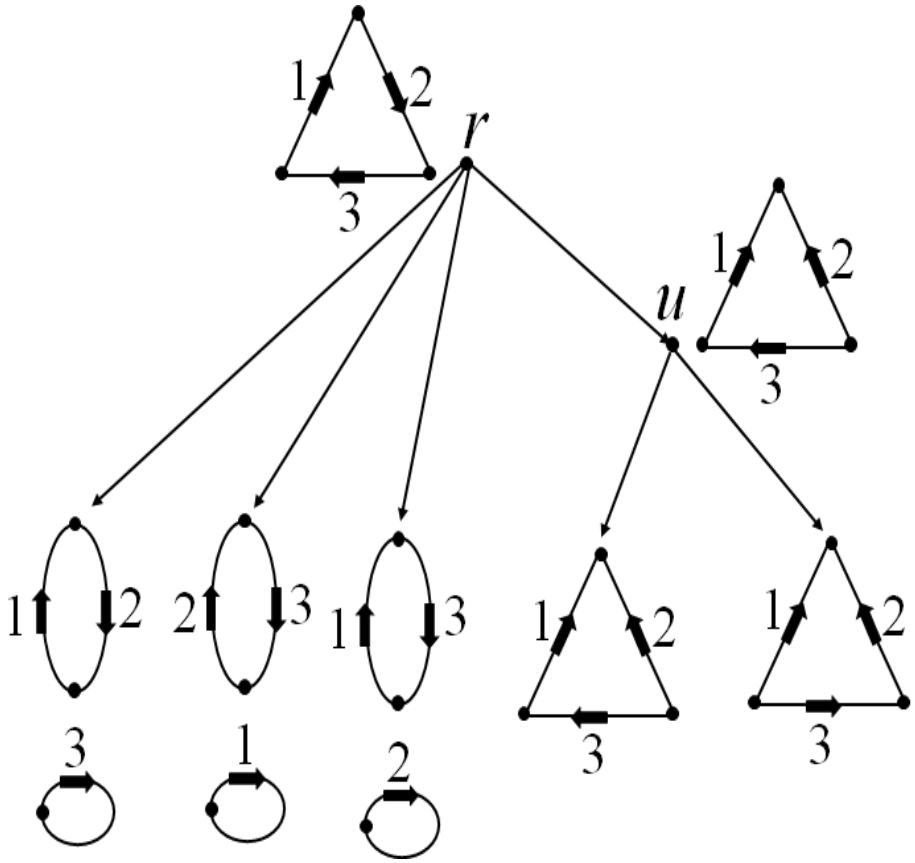

Figure 22. Result (output arrangement) of executing the heuristic cyclic *SCJ* reconstruction algorithm; input data at the leaves are shown in Figure 19; all costs are uniform.

3.6. DCJ Reconstruction Algorithm

In the cyclic equal-content setting (with only the *DM* operation), the *DCJ* reconstruction problem becomes NP-hard even for a tree consisting of a root and three leaves [29]. In [12], a much more general problem of *DCJ* reconstruction has been exactly reduced to an integer linear programming (ILP) problem.

The cyclic equal-content setting *DCJ* reconstruction problem considered here is exactly reduced to a sequence of SAT problems. The latter way is preferable, since presently there exist programs, called SAT solvers, which in several minutes verify the satisfiability of a formula with tens of thousands of variables and hundreds of thousands of clauses, and if it is satisfiable, output a satisfying assignment of variables; see, e.g., [27] and the Internet site http://fmv.jku.at/lingeling (accessed on 24 February 2023), where references to the corresponding programs and numerous publications can be found. In the cyclic equal-content setting costs of any arrangements are positive integers. By an *admissible* edge we call any edge in given structures assigned to leaves of a given tree in the *DCJ* reconstruction problem. One name at the leaves of the tree exactly corresponds to one admissible edge.

To describe the reduction in the cyclic equal-content *DCJ* reconstruction problem on a star with tree root r to a sequence of SAT problems, we consider an ***auxiliary problem***:

(*) Does there exist an arrangement (in fact, a structure at r) with a cost no greater than a given positive integer $\boldsymbol{k}$?

We will decide the (*) problem by finding a SAT_k problem such that any of its affirmative solution describes a structure R at r in the (*) problem, and the thus-obtained arrangement on a star will be a solution to the auxiliary (*) problem.

It is clear that solving this SAT_k problem is equivalent to solving problem (*). Because of this, the smallest k^* for which the corresponding SAT_{k^*} problem has an affirmative solution is the lowest arrangement cost in the *DCJ* reconstruction problem; the corresponding

arrangement is minimum. Below a reduction in the (*) problem to a SAT_k problem is described for every k.

Being able to solve the (*) problem for various k, we can trivially solve the original *DCJ* reconstruction problem. Let k_1 and k_2 be such that $k_1 \leq k^* \leq k_2$. For example, we may set $k_1 = 0$ (in Section 3.8 we present a way to obtain a nontrivial lower estimate for k^*) and $k_2 = nl$, where n is the number of names in leaves and l is the number of leaves in a given tree. Given these bounds, k_1 and k_2, for k we take the integral part $[(k_1 + k_2)/2]$, and then solve the SAT_k problem. If the solution is affirmative, we pass to the left-hand part of the segment $[k_1, k_2]$; otherwise, to the right-hand one. We again denote this next segment by $[k_1, k_2]$, and so on. In $[\log_2(k_2 - k_1)] + 1$ steps, we arrive at a segment $k_1 = k_2$ and the desired $k^* = k_1$.

Now we describe the conjunctive normal form CNF_k, form for any given k. However, first, we describe an important general construction. For two structures, a and b, there was defined a graph $a + b$, called the *breakpoint graph*. For equal-content structures, a and b, the breakpoint graph can equivalently be defined simpler, as follows. Consider the set M of extremities of all edges in a (or, equivalently, in b), and in the *two matchings on M* corresponding to the structures a and b. Then, $a + b$ is defined to be the set of edges in M consisting of the edges belonging to one of these matchings. The minimum number of *DM* operations required to transform one cyclic equal-content structure a into another b is $n - x$, where n is the number of names in a (or, the same thing, in b), and x is the number of cycles in $a + b$. This can easily be proved directly. Every edge in $a + b$ corresponds to a pair of joined extremities in a or in b, and, at the same time, corresponds to an edge of one of the two matchings in M corresponding to a and b. In the graph $a + b$, edges are labeled by successively alternating symbols a or b according to whether this edge belongs, as a joined pair, to a or to b. If a and b are cyclic structures, then $a + b$ is a cyclic graph.

Because of this, the cost y of an arrangement along a star is equal to $y = nl - x$, where x is the *total number of cycles* (over all leaves s) in all breakpoint graphs $R + s$, where R is a structure at the root r, and s is the given structure at a leaf s, as well as n is the number of names in leaves and l is the number of leaves in the given tree. Therefore, the existence of an arrangement with cost $y \leq k$, ref. to (*), is equivalent to the existence of a structure R in r with the total number x of cycles satisfying $nl - k \leq x$. Thus, to reduce the auxiliary problem (*) to a SAT_k problem, we have to construct a conjunctive normal form CNF_k which is satisfiable if, and only, if there exists a matching R on M at r, such that $nl - k \leq x$. Now we construct this CNF_k that describes such R on M and estimates the total number of cycles in all $R + s$ from below.

For every unordered pair $p = (i_k,j_l)$, $i_k \neq j_l$, of different extremities in M, we introduce a ***variable***, also denoted by $p = (i_k,j_l)$, such that at r we have the following, $p = 1$ if p is joined and $p = 0$ otherwise. A set of these variables describes the subgraph in M; it can be a matching on M and the desired structure R at r. For each leaf s, for each p joined at s, and for each positive integer m, $1 \leq m \leq nl - k$, we introduce a ***variable*** p_{sm} such that $p_{sm} = 1$ if p belongs to the mth cycle in the set of all cycles in all $R + s$, where to each m there corresponds a least one cycle different from those corresponding to other $m_1 \neq m$; and $p_{sm} = 0$ otherwise. In the first case, m will be called the *cycle number* of the pair p. If $p_{sm} = 1$, the extremities in p are neighboring in the corresponding cycle.

Let us express the fact that each extremity $i_j \in M$ is joined with exactly one extremity, i.e., values of the variables p on which CNF_k takes the value 1 define a partition of M into cycles. For instance, in M there are 4 nodes, denoted by 1, 2, 3, and 4, and for node 1 we write

$$((1,2) \wedge \neg(1,3) \wedge \neg(1,4)) \vee ((1,3) \wedge \neg(1,2) \wedge \neg(1,4)) \vee ((1,4) \wedge \neg(1,2) \wedge \neg(1,3)). \quad (1)$$

In general, we obtain a part of a CNF of the form $\vee_i ((1,i) \wedge \wedge_{i'} \neg(1,i'))$, where i runs over all nodes in M except for 1, and i' runs over all nodes in M except for 1 and i. We conjunctively combine these formulae for all other nodes in M except for 1.

The next three parts of CNF_k depend on the parameter k. We express the fact that at every leaf s, every joined pair p is assigned with a unique number m. For instance, at leaf s, node 1 is joined with 2 and $nl - k = 3$; i.e., $1 \le m \le 3$. Then, we write a part of the CNF:

$$((1,2)_{s1} \wedge \neg(1,2)_{s2} \wedge \neg(1,2)_{s3}) \vee ((1,2)_{s2} \wedge \neg(1,2)_{s1} \vee \neg(1,2)_{s3}) \vee ((1,2)_{s3} \wedge \neg(1,2)_{s1} \wedge \neg(1,2)_{s2}). \quad (2)$$

In general, we have $\vee_m ((i,j)_{sm} \wedge \wedge_{m'} \neg(i,j)_{sm'})$, where i and j are joined extremities in s, m running over all cycle numbers and m' running over all numbers except m. By conjunctively combining such formulae for all other joined $p = (i,j)$ and all leaves s, we obtain $\wedge_{ps} \vee_m (i,j)_{sm} \wedge \wedge_{m'} \neg(i,j)_{sm'}$. Here, and in what follows, we use structures defined at the leaves. Note that some other q_s in the same leaf s (or even in another leaf; see below) may be given the same number m.

Now we express the fact that for every cycle number m ($1 \le m \le nl - k$), at some leaf there exists a joined pair p having this number. For example, there are two leaves; the pairs p joined at leaf 1 are (1,2) and (3,4), and those joined at leaf 2 are (1,3) and (2,4). Then, for every cycle number m we write

$$(1,2)_{1m} \vee (3,4)_{1m} \vee (1,3)_{2m} \vee (2,4)_{2m}.$$

In general, this will be the disjunction of the variables p_{sm} over all joined p and all s, for each m separately, $\vee_{ps} p_{sm}$. By conjunctively combining such formulae for all cycle numbers m, we obtain $\wedge_m \vee_{ps} p_{sm}$.

Now let us ensure equality of cycle numbers of all pairs p in each cycle in $R + s$, for any given leaf s. It suffices to ensure the equality of numbers for neighboring pairs, i.e., for those separated by some joined pair x in R. In other words, we have to ensure the implication that if two joined pair p and q at one leaf s are connected by a joined pair from R, then p and q have the same cycle number m. This is written separately for every cycle number m. For instance, for any joined pairs $p = (1,2)$ and $q = (3,4)$ at the same leaf s, we write (in the parentheses, there are all possible joined pairs in R)

$$((1,3) \vee (1,4) \vee (2,3) \vee (2,4)) \to (p_{sm} \leftrightarrow q_{sm}). \quad (3)$$

Here, p_{sm} can be equal to 0; recall that a joined pair $x \in R$ is an edge in $R + s$. By conjunctively adding such formulae for all cycle numbers m, all leaves s, and joined pairs $p = (p_1,p_2)$, $q = (q_1,q_2)$ in s, we obtain

$$\wedge_{mspq}((p_1,q_1) \vee (p_1,q_2) \vee (p_2,q_1) \vee (p_2,q_2)) \to (p_{sm} \leftrightarrow q_{sm}).$$

It may seem that we also need to ensure that joined pairs in different cycles have different cycle numbers. However, this is not necessary: if different cycles have the same numbers, the number of cycles will be greater than $nl - k$, which also satisfies the conditions of problem (*). The flow chart of the algorithm is shown in Figure 23.

Proof of Theorem 5. The description of CNF_k expresses the condition $nl - k \le x$, and the trivial bounds for k^* (from 0 to nl) imply the exactness of the reduction. Let us check the estimate for the size of CNF_k; in respect of the number of variables, it is obvious. To obtain an estimate for the number of disjunctions, note that Equation (1) is equivalent to a conjunction of four disjunctions:

$$((1,2) \vee (1,3) \vee (1,4)) \wedge (\neg(1,2) \vee \neg(1,3)) \wedge (\neg(1,2) \vee \neg(1,4)) \wedge (\neg(1,3) \vee \neg(1,4)).$$

Equation (2) is equivalent to an analogous conjunction. Equation (3) is equivalent to a conjunction of four disjunctions:

$$(\neg(1,3) \vee \neg p_{s1} \vee q_{s1}) \vee (\neg(1,3) \vee p_{s1} \vee \neg q_{s1})$$

$$\wedge(\neg(1,4) \vee \neg p_{s1} \vee q_{s1}) \vee (\neg(1,4) \vee p_{s1} \vee \neg q_{s1})$$

$$\wedge(\neg(2,3) \vee \neg p_{s1} \vee q_{s1}) \vee (\neg(2,3) \vee p_{s1} \vee \neg q_{s1})$$

$$\wedge(\neg(2,4) \vee \neg p_{s1} \vee q_{s1}) \vee (\neg(2,4) \vee p_{s1} \vee \neg q_{s1}).$$

Here, at each leaf there are of the order of n^2 pairs (p,q), which in total, over all leaves, gives ln^2; this is multiplied by the number nl of cycle numbers. □

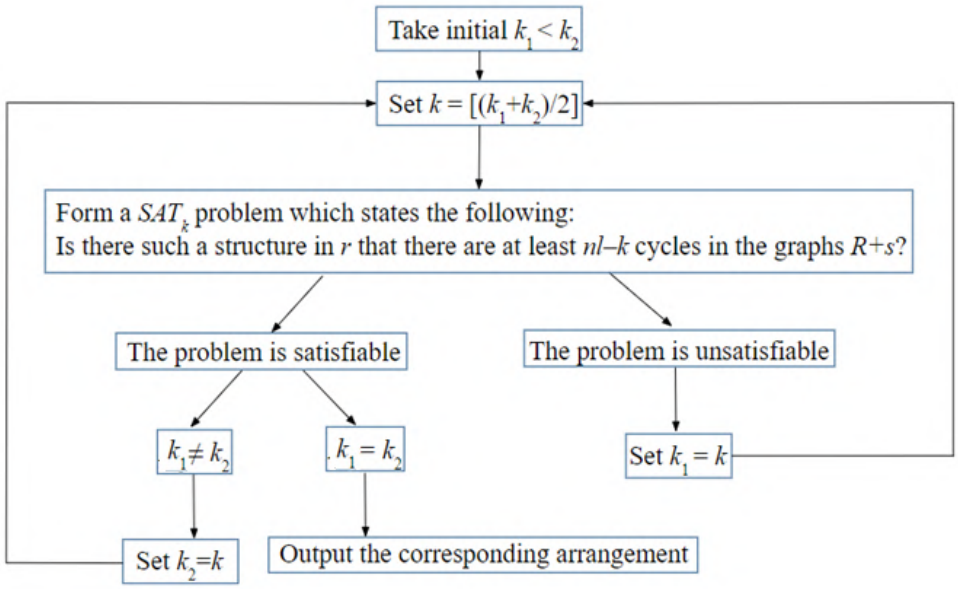

Figure 23. The flow chart of the algorithm for Theorem 5.

3.7. Example of the DCJ Reconstruction

It is often reasonable to confine ourselves with a *reduced version* of the problem: a root structure is sought for among the structures in which any joined pair p of extremities occurs at least at one of the leaves. Our simulation (results are not presented here) shows that a root structure appearing in the DCJ reconstruction usually possesses this property. Then, the number of variables and disjunctions in the corresponding CNF_k considerably reduces.

Let us perform the DCJ reconstruction with a *reduced version* for the input data shown in Figure 24.

For the upper and lower bounds for the parameter k, we take the trivial values $k_1 = 0$ and $k_2 = nl = 9$, where $n = 3$ is the number of names in leaves and $l = 3$ is the number of leaves. According to the algorithm in Section 3.6, we construct CNF_2, which has $nl - k = 7$, and obtain that it has no affirmative solution; therefore, a structure R is not well defined. Then, we construct CNF_3, which has $nl - k = 6$, and obtain that it does have an affirmative solution. The satisfying collection of variables is as follows (we present those equal to 1 only): $(1_2,2_1)$, $(2_2,3_1)$, $(3_2,1_1)$; $(1_2,2_1)_{l1}$, $(2_2,1_1)_{l2}$, $(3_1,3_2)_{l2}$, $(2_2,3_1)_{m3}$, $(3_2,2_1)_{m4}$, $(1_1,1_2)_{m4}$, $(1_2,3_1)_{r5}$, $(3_2,1_1)_{r6}$, $(2_1,2_2)_{r6}$, where the indices l, m, and r indicate, respectively, the left, middle, and right leaves. The first three variables determine a corresponding structure R with six cycles (in total) in all graphs $R + s$. The resulting arrangement is shown in Figure 25a, and these breakpoint graphs $R + s$, each of them consisting of two cycles, are shown in Figure 25b.

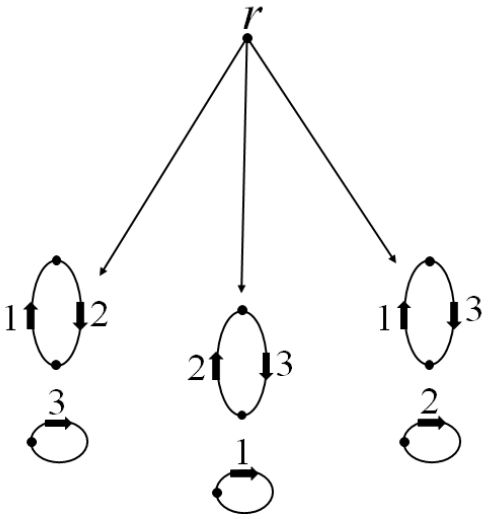

Figure 24. Input data at leaves (two cycles at each leaf) for the *DCJ* reconstruction with a *reduced version*: reduction to SAT problems.

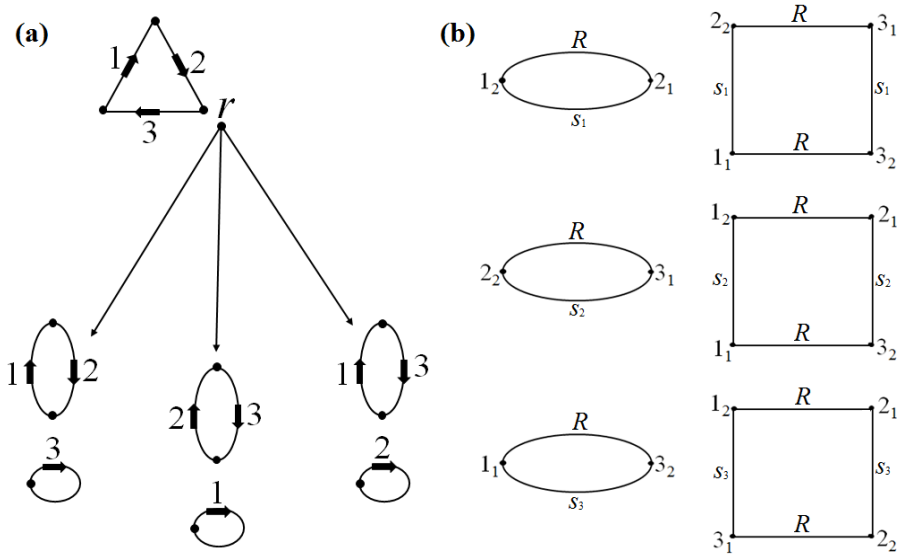

Figure 25. (**a**) Resulting arrangement in the *DCJ* reconstruction for the data given in Figure 20, obtained by a reduction to SAT problems. (**b**) Three corresponding breakpoint graphs *R*+*s*, each of them containing two cycles of lengths 2 and 4.

3.8. Lower Bound on the Parameter k in the Conjunctive Normal Form CNF_k and Final Form of a Common Graph

One can improve the lower bound on k^* by constructing a *common graph G* of all leaf structures. This is a graph with node set *M*, and an edge connects two extremities if they are joined in some leaf structure; the edge is labeled with the name of the corresponding leaf. The common graph of structures *a* and *b* is their breakpoint graph. Thus, two nodes can be connected by several (up to *l*) edges, and the degree of any node of the graph is *l*, since all the structures are cyclic. A common graph *G* is said to be of a *final form* if every one of its connected components is two nodes connected by *l* edges. This graph *G* is of a final form if, and only if, it is obtained from identical leaf structures. In a common graph *G* of a final form there are *n* connected components where *n* is the number of names in leaves of the tree. A *DM operation* over *G* consists of replacing two edges having the same name with other two edges having the same name and the same four extremities.

A *DM* operation is invertible, and its inverse is also a *DM* operation. Therefore, the total number of *DM* operations in transformations of a root structure r to leaf structures s equals the total number of DM operations in transformations of leaf structures to a root one. Additionally, the latter equals the number of *DM* operations in the shortest transformation (commonly called *reduction*) of G to a final form. A cycle in G is said to be *alternating* if each edge in it is labeled by one of two names and these labels alternate. In a graph of a final form there are $nl\cdot(l-1)/2$ alternating cycles (all of length 2). Note the following, any alternating cycle in G is contained in precisely one $s_i + s_j$ where s_i and s_j are any two leaves of the tree. Therefore, the number of alternating cycles in an arbitrary G is not greater than in a final one, and a DM can increase this number by at most l–1. Hence, we obtain a lower estimate of $\frac{nl}{2} - \frac{d}{l-1}$ for the SAT bound, where d is the *number* of alternating cycles in G. This is a lower *bound* for k*.

An upper *bound* for k* can be obtained by exhaustively examining the arrangements in which r is a structure coinciding with a structure at some leaf and by computing the minimum cost among all these arrangements.

4. Discussion

We have described an algorithm for constructing an intermediate tree, which, in particular, makes it possible to determine coordinated evolution of proteins with respect to the genes that encode them, and of these genes themselves with respect to the species that contain them. This has also been performed for an important case where the species tree is represented by a network. We have described algorithms for reconstructing the evolution of structures along the tree, which, in particular, allow the reconstruction of genomic structures along a given tree.

Let us outline possible directions for further research. In the problem of constructing an intermediate tree G, we have proposed a way to construct a collection B of clades for which our algorithm outputs the best reconciliation of G with a preassigned tree P and network S. Further research can be aimed at finding other ways to construct collection B. Additionally, of interest is the case where P is a non-binary tree, along with the problem of optimal binarization of P. A binarization method is described and can easily be adapted for our algorithm, but the problem of choosing an optimal binarization still remains. The question of whether it is possible to remove the constraints on costs of the events in the statement of Theorem 3 so that the described algorithm is still exact also remains open.

The questions of an exact efficient algorithm for *SCJ* reconstruction on an arbitrary tree for arbitrary costs of *SCJ* operations and of for *DCJ* reconstruction even on a two-star tree in the cyclic case remain open.

Author Contributions: Conceptualization, V.L. and K.G.; proof, K.G. and V.L.; writing—original draft preparation, K.G.; writing—review and editing, V.L.; supervision, V.L.; project administration, V.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by RFBR grant 20-01-00670.

Data Availability Statement: Not Applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. The Procedure to Construct the Main Parameter of the Algorithm

Given collection B', exhaustively examining sets from B' in ascending order of their cardinality (in the case of equal cardinalities, in an arbitrary order). For each successive set A, we perform the first feasible step out of the following five (Steps 1–5).

(1) If there exists a partition of A into two sets A_1 and A_2 in B', then procced with the next set A.

(2) If there exist two sets, A_1 and A_2, in the current B' such that the sets $A_1 \cap A$ and $A_2 \cap A$ form a partition of A, consider both of the cases $A = A_i \cap A$ with $i = 1$ and $i = 2$,

and so on over all steps until we obtain one-element sets A. At each step, we add to B' the sets forming the partition of A; the same in all the following steps.

(3) If there exists a set A_1 in the current B' such that $A_1 \subset A$ (take A_1 for which $|A_1|$ is the largest), consider both of the sets $A = A_1$ and $A = A \backslash A_1$, and so on over all steps until one-element sets for A are obtained.

(4) If there exists a set A_1 in the current B', such that $A_1 \cap A$ and $A \backslash (A_1 \cap A)$ is a partition (take A_1 for which $|A_1 \cap A|$ is the largest), then consider both sets, and so on over all steps until one-element sets are obtained.

(5) Take any partition of A into two sets A_1 and A_2 of the same cardinality (with accuracy to 1) and so on over all steps until one-element sets are obtained.

Let us estimate the procedure time for the main parameter $\boldsymbol{B}$. We consider at most $|B'|$ sets A from the original B', and for each of them we construct a binary tree with at most $|M|$ nodes. At each node, we consider five cases; the time required to check the conditions of the cases is of the order of at most $(|B'| \cdot |M|)^2 \cdot |M|$. Thus, the total time required to construct the collection $\boldsymbol{B}$ is of the order of $|B'|^3 \cdot |M|^4$. Additionally, a fast version of constructing collection $\boldsymbol{B}$ is possible where, in the conditions of Steps 1–4, we use not the current B' but rather the original B'. Then, the runtime of the procedure is of the order of $|B'|^3 \cdot |M|^2$. □

References

1. Li, T.; Yang, B.; Liu, H.; Ju, J.; Tang, J.; Subramanian, S.; Zhang, Z. GMDL: Toward precise head pose estimation via Gaussian mixed distribution learning for students' attention understanding. *Infrared Phys. Technol.* **2022**, *122*, 104099. [CrossRef]
2. Liu, H.; Fang, S.; Zhang, Z.; Li, D.; Lin, K.; Wang, J. MFDNet: Collaborative Poses Perception and Matrix Fisher Distribution for Head Pose Estimation. *IEEE Trans. Multimed.* **2021**, *24*, 2449–2460. [CrossRef]
3. Quinlan, J.R. Induction of decision trees. *Mach. Learn.* **1986**, *1*, 81–106. [CrossRef]
4. Witten, I.H.; Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations; Morgan Kaufmann: San Francisco, CA, USA. Available online: http://www.cs.waikato.ac.nz/~ml/weka/book.html (accessed on 24 February 2023).
5. Moayedi, H.; Bui, D.T.; Kalantar, B.; Foong, L.K. Machine-Learning-Based Classification Approaches toward Recognizing Slope Stability Failure. *Appl. Sci.* **2019**, *9*, 4638. [CrossRef]
6. Bulteau, L.; Weller, M. Parameterized Algorithms in Bioinformatics: An Overview. *Algorithms* **2019**, *12*, 256. [CrossRef]
7. Huson, D.H.; Rupp, R.; Scornavacca, C. *Phylogenetic Networks—Concepts, Algorithms and Applications*; Cambridge University Press: Cambridge, UK, 2010.
8. Kuitche, E.; Lafond, M.; Ouangraoua, A. Reconstructing protein and gene phylogenies using reconciliation and soft-clustering. *J. Bioinform. Comput. Biol.* **2017**, *15*, 1740007. [CrossRef]
9. LeMay, M.; Libeskind-Hadas, R.; Wu, Y.-C. A Polynomial-Time Algorithm for Minimizing the Deep Coalescence Cost for Level-1 Species Nets. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2022**, *19*, 2642–2653. [CrossRef]
10. Feijao, P.; Meidanis, J. SCJ: A Breakpoint-Like Distance that Simplifies Several Rearrangement Problems. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2011**, *8*, 1318–1429. [CrossRef]
11. Gorbunov, K.Y.; Lyubetsky, V.A. Multiplicatively exact algorithms for transformation and reconstruction of directed path-cycle graphs with repeated edges. *Mathematics* **2021**, *9*, 2576. [CrossRef]
12. Lyubetsky, V.A.; Gershgorin, R.A.; Gorbunov, K.Y. Chromosome structures: Reduction of certain problems with unequal gene content and gene paralogs to integer linear programming. *BMC Bioinform.* **2017**, *18*, 537. [CrossRef]
13. Page, R.D.M. Maps between trees and cladistic analysis of historical associations among genes, organisms and areas. *Syst. Biol.* **1994**, *43*, 58–77.
14. Guigo, R.; Muchnik, I.; Smith, T. Reconstruction of ancient molecular phylogeny. *Mol. Phylogenet. Evol.* **1996**, *6*, 189–213. [CrossRef]
15. Li, D.; Liu, H.; Zhang, Z.; Lin, K.; Xiong, N. CARM: Confidence-aware recommender model via review representation learning and historical rating behavior. *Neurocomputing* **2021**, *455*, 283–296. [CrossRef]
16. Liu, H.; Nie, H.; Zhang, Z.; Li, Y.-F. Anisotropic angle distribution learning for head pose estimation. *Neurocomputing* **2020**, *433*, 310–322. [CrossRef]
17. Liu, T.; Liu, H.; Li, Y.-F.; Zengzhao, C.; Zhang, Z.; Liu, S. Flexible FTIR Spectral Imaging Enhancement for Industrial Robot Infrared Vision Sensing. *IEEE Trans. Ind. Inform.* **2019**, *16*, 544–554. [CrossRef]
18. Liu, H.; Zheng, C.; Li, D.; Shen, X.; Lin, K.; Wang, J.; Zhang, Z.; Zhang, Z.; Xiong, N. EDMF: Efficient Deep Matrix Factorization with Review Feature Learning for Industrial Recommender System. *IEEE Trans. Ind. Inform.* **2021**, *18*, 4361–4371. [CrossRef]
19. Van Iersel, L.; Jones, M.; Weller, M. Embedding Phylogenetic Trees in Networks of Low Treewidth. In Proceedings of the 30th Annual European Symposium on Algorithms (ESA 2022), Berlin/Potsdam, Germany, 5–9 September 2022; pp. 69:1–69:14.

20. Zhang, L. On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies. *J. Comput. Biol.* **1997**, *4*, 177–187. [CrossRef] [PubMed]
21. Ma, B.; Li, L.; Zhang, L. From gene trees to species trees. *SIAM J. Comput.* **2000**, *30*, 729–752. [CrossRef]
22. Rusin, L.Y.; Lyubetskaya, E.V.; Gorbunov, K.Y.; Lyubetsky, V.A. Reconciliation of Gene and Species Trees. *BioMed Res. Int.* **2014**, *2014*, 642089. [CrossRef]
23. Van Iersel, L.; Janssen, R.; Jones, M.; Murakami, Y.; Zeh, N. Polynomial-Time Algorithms for Phylogenetic Inference Problems Involving Duplication and Reticulation. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2020**, *17*, 14–26. [CrossRef]
24. Luhmann, N.; Lafond, M.; Thevenin, A.; Ouangraoua, A.; Wittler, R.; Chauve, C. The SCJ Small Parsimony Problem for Weighted Gene Adjacencies. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **2019**, *16*, 1364–1373. [CrossRef] [PubMed]
25. Gorbunov, K.Y.; Lyubetsky, V.A. Linear time additively exact algorithm for transformation of chain-cycle graphs for arbitrary costs of deletions and insertions. *Mathematics* **2020**, *8*, 2001. [CrossRef]
26. Gorbunov, K.Y.; Gershgorin, R.A.; Lyubetsky, V.A. Rearrangement and inference of chromosome structures. *Mol. Biol.* **2015**, *49*, 327–338. [CrossRef]
27. Sohanghpurwala, A.A.; Hassan, M.W.; Athanas, P. Hardware accelerated SAT solvers—A survey. *J. Parallel Distrib. Comput.* **2017**, *106*, 170–184. [CrossRef]
28. Korte, B.; Vigen, J. *Combinatorial Optimization. Theory and Algorithms*, 6th ed.; Springer: Bonn, Germany, 2018.
29. Tannier, E.; Zheng, C.; Sankoff, D. Multichromosomal median and halving problems under different genomic distances. *BMC Bioinform.* **2009**, *10*, 120. [CrossRef] [PubMed]