Stockholms universitet

Exact & heuristic algorithms for correcting disturbed Later-Divergence-Time Graphs

Mohammed Habib

Handledare: Marc Hellmuth & David Schaller
Examinator: Lars Arvestad
Inlämningsdatum: August 2021

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

The occurrence of at least one horizontal gene transfer (HGT) event between two genes is indicated by the existence of an edge connecting the two corresponding vertices in a later-divergent-time (LDT) graph, i.e., a properly colored cograph with a set of triples (induced P_3s with pairwise distinct colors) that can be combined into a supertree. This means that phylogenetic trees can be reconstructed from LDT graphs; however, graph representations of real data tends to have a lot of noise which makes them not fulfill the properties of LDT graphs and as such it is desired to edit a given properly colored graph into an LDT graph such that the edit distance is minimized. Since this problem is NP-complete, we provide an integer linear program (ILP) formulation as well as an implementation of the ILP that edits a given properly colored graph into an LDT graph such that the edit distance is minimized. We also provide a heuristic that attempts to edit a given properly colored graph into an LDT graph and we look at different variations of this heuristic. Additionally, we present restrictions to this heuristic that ensures the resulting graph is an LDT graph.

To benchmark the different variations of the heuristic presented in this thesis, we generated properly colored non LDT graphs that we applied the heuristics to by perturbing LDT graphs using different probabilities for inserting and deleting edges. We then looked at how often these heuristics resulted in an LDT graph and how the edit distance of the heuristics compared to the exact solutions from the ILP. The results showed that the heuristics performed near optimal on smaller graphs such as those with 18 or less vertices. We also saw that the success rates decreased as the size of the graphs in-

creased and while we were only able to compare edit distances of graphs with up to 18 vertices due to time constraints, it did seem as though the edit distance relative to the exact solutions became worse as the size of the graph increased. Additionally, we saw that the perturbation probabilities also had an effect on the success rates as well as the edit distance.

Sammanfattning

Åtminstånde en horisontell genöverföring mellan två gener är indikerat av existensen av en kant mellan de två motsvarande noder i en later-divergent-time (LDT) graf som är karakteriserad som en korrekt färgad graf utan någon inducerad väg på fyra noder och en mängd tripletter (inducerad väg på tre noder med parvis disjunkta färger) som kan kombineras till ett superträd. Fylogenetiska träd kan alltså återuppbyggas från LDT grafer, men eftersom grafer som representerar data oftast kommer med massa störningar innebär detta att graferna måste korrigeras till LDT grafer sådant att redigerings avståndet är minimerat. Eftersom detta problem är NP-komplett, ger vi en ILP (heltal linjärt program) formulering samt en implementation av denna ILP som redigerar en given korrekt färgad graf till en LDT graf sådant att redigerings avtsåndet är minimalt. Vi ger även en heuristik som försöker redigera en given korrekt färgad graf till en LDT graf och vi kollar på olika varianter av denna heuristik. Vi lägger även fram restriktioner till denna heuristik för att garantera att resultatet är en LDT graf.

För att riktmärka de olika varianter av heuristiken vi presenterat, genererar vi korrekt färgade icke LDT grafer som vi använder dessa heuristiker på, genom att störa LDT grafer med olika sannolikheter för att ta bort samt lägga till kanter. Vi kollar sedan på hur ofta resultatet är en LDT graf samt hur bra redigerings avståndet är jämfört med de exakta lösningarna som genererades med hjälp av ILP. Resultaten av dessa riktmätningar visade att de olika varianterna av heuristiken var nästintill optimala på mindre grafer, alltså sådana med 18 eller färre noder. Vi såg dessutom att varianterna av vår heuristik presterade sämre ju större den inmatade grafen var och att sannolikheterna vid störningen av LDT grafen har

en effekt på prestandan.

Acknowledgements

I would like to thank both of my supervisors Marc Hellmuth and David Schaller for all the help I have received. Your feedback has been immensely helpful in getting me through this project.

Contents

1 Introduction

1.1 Background

Phylogenetic trees contain information pertaining to the evolutionary relationships between different species, organisms or genes. Within these trees, certain events are depicted in the form of branches, one of which is horizontal gene transfer (HGT), i.e., the transferal of genes through other means than parent to offspring. For example, it is believed that HGT greatly contributes to the evolution of bacteria, and adaptations such as antibiotic resistance [3].

Two methods of inferring horizontal gene transfer events are parametric and phylogenetic methods [8]. Parametric methods involve comparisons between parts of genomes and genomic signatures, i.e., characteristics of genome sequences, such as GC content, that are specific to certain species. If these widely differ, one can infer a potential HGT event [2, 9]. Phylogenetic methods involve reconstructing and comparing phylogenetic trees. While we can reconstruct such trees from graphs with certain properties [9], one of the challenges is retrieving a graph that adheres to those properties from real data and such data tends to come with a lot of noise. Thus by reducing the noise in such a way that a given graph satisfies the required properties, we are able to get a more accurate depiction of the relationships amongst species.

In this thesis our goal is to find ways to reduce noise in given data, specifically properly colored, undirected and loop-free graphs, such that phylogenetic trees can be reconstructed and compared in order to infer HGT events.

Graphs [9, 6]. In this thesis all graphs are finite, undirected and loop-free, denoted by $G = (V, E)$, where $V(G) := V$ is the set of vertices and $E(G) := E$ is the set of edges connecting vertices $x, y \in V$, which we also refer to as x and y being adjacent. We denote the set of all vertices adjacent to a vertex x by $N(x)$ and we call this set the *neighborhood* of x. The *degree* of a vertex is the number of vertices it is adjacent to. An edge connecting vertices $u, v \in V$ is denoted by (u, v). The *complement* of a graph $G = (V, E)$ is denoted by $\overline{G} = (V, \overline{E})$, $\overline{E}$ being the complement of E, i.e., $\overline{E} = \{(x, y) \notin E \mid \forall x, y \in V, x \neq y\}$. A *complete graph* is a graph in which every pair of vertices is connected by an edge, i.e., for all $x, y \in V, x \neq y$, there is an edge (x, y). We denote a complete graph on n vertices by K_n. For a graph H, such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, we say that H is a *subgraph* of G, which is denoted by $H \subseteq G$. An *induced* subgraph H of G has vertex set $S \subset V$ and edge set $E' = \{(x, y) \in E(G) \mid x, y \in S\}$. A *walk* is a sequence of vertices $\langle v_1, v_2, \ldots, v_k \rangle$ such that $(v_i, v_{i+1}) \in E(G)$ and $v_i = v_j$ is possible, i.e., vertices can be repeated. A *path* is a walk in which all vertices are distinct and a path on n vertices is denoted by P_n. A *cycle* is a walk in which only the first and last vertices are repeated and are the same, i.e., $v_1 = v_k$ in the cycle $\langle v_1, v_2, \ldots, v_k \rangle$ where $v_1, v_2, \ldots, v_{k-1}$ are pairwise distinct. Given a graph $G = (V, E)$ such that $x, y, z \in V$ and $(x, y), (y, z) \in E$, we denote by the sequence $\langle x, y, z \rangle$, a path on three vertices. A graph G is connected if there is a path between any two distinct vertices of G. An *independent set* of a graph $G = (V, E)$ is a set $I \subseteq V$ such that for any two distinct vertices $x, y \in I$, $(x, y) \notin E$. A *complete multipartite* graph is a graph $G = (V, E)$ with independent sets $I_1, ..., I_n$ where $(x, y) \in E$ if and only if $x \in I_i$ and $y \in I_j$ such that $i \neq j$. A *colored graph* is a graph whose vertices are colored by some colors from a set C. By (G, σ), we denote a colored graph G with a vertex coloring $\sigma: V \to C$ and we say G is *properly colored* if for all $(x, y) \in E(G)$, $\sigma(x) \neq \sigma(y)$.

Definition 1. *Let $G = (V, E)$ be a graph and C be a set of colors. The map $\sigma: V \to C$ is a color map for the graph G.*

Trees [9, 6] are defined as connected graphs with no cycles and we denote the leaves of a tree T by $L(T)$. A *rooted* tree is a tree T such that it is rooted in one of its vertices and we denote the root by ρ_T. If the root of a tree has degree 1, we say the tree is *planted* and we denote the planted root by 0_T. An *inner vertex* has at least degree 2 and the set of inner vertices of a tree T is given by $V^0(T) := V(T) \setminus (L(T) \cup 0_T)$. By $x \preceq_T y$ we denote that y lies on the unique path from the root of the tree T to x, i.e., y is an *ancestor* of x and consequently, x is a *descendant* of y. For an edge (x, y) in a tree such that $x \preceq_T y$ we say that x is the *child* of y and y is the *parent* of x. A tree is *phylogenetic* if all inner vertices have at least two children. By $x \prec y$ we denote that y is a *strict ancestor* of x, i.e., $x \preceq y$ where $x \neq y$. In terms of ancestry, the leaves of a tree T are $\preceq_T$-minimal and we say that T is a tree on $L(T)$. The *last common ancestor* of two vertices $x, y \in V(T)$, denoted by $lca_T(x, y)$, is the $\preceq_T$-minimal vertex u, such that $x \preceq_T u$ and $y \preceq_T u$. *Gene* and *species* trees are trees whose leaves represent genes and species, respectively. These trees are generally shown in conjunction with one another, i.e., gene trees residing in species trees, as shown in figure 1.

Definition 2 ([9])**.** *Let T be a rooted tree. The map $\mathscr{T}_T: V(T) \to \mathbb{R}$ is a time map for the tree T if $x \prec_T y$ implies $\mathscr{T}_T(x) < \mathscr{T}_T(y)$ for all $x, y \in V(T)$.*

Cographs [9, 5] have many equivalent definitions, one of which is graphs that contain no path on four vertices as an induced subgraph, hence why they are sometimes called P_4-free-graphs. If G is a cograph, then so is any induced subgraph of G, as well as the complement $\overline{G}$. Starting with K_1s, we can recursively construct a cograph by means of joins or disjoint unions of those cographs. This recursive construction of a cograph $G=(V,E)$ defines a *cotree*, (T,t), where t is a labeling of the inner vertices, such that $L(T)=V$ and the inner vertices are labeled with the numbers 0 and 1. These numbers represent a join (0) or disjoint union operation (1) between the cographs formed by the subtrees of an inner vertex. A cograph G formed by a cotree T has an edge $(x,y)\in E \iff t(lca_T(x,y))=1$.

Rooted Triples [9, 7]. A rooted triple is a binary tree T on three leaves. We write $ab|c$ whenever the path from a to b does not intersect the path from c to the root, i.e., $lca_T(a,b) \prec lca_T(a,c) = lca_T(b,c)$, and we say the tree T *displays* the rooted triple $ab|c$. We denote a set of triples by R and say that R is consistent if there is a tree T with $L_R \subseteq L(T)$, where $L_R := \bigcup_{r\in R} L(r)$, that displays every triple $r \in R$. By $R' \subseteq R$, we denote a maximum consistent subset of R, which is a set of triples such that $|R \setminus R'|$ is minimal, i.e., R' excludes the least amount of triples possible in order for R' to be consistent. Since all trees in this thesis are rooted, we simply refer to a rooted triple as a triple. A *species triple* is a triple on the leaves of a species tree or on the color set of a colored graph. Throughout, we will generally denote the vertices of a colored graph by non-capitalized roman letters and their colors by the same, capitalized, roman letter. A colored graph (G,σ) exhibits a species triple $XY|Z$ if there is a P_3, $\langle x,z,y\rangle$, as an induced subgraph of G, such that $\sigma(x)=X$, $\sigma(y)=Y$, $\sigma(z)=Z$ are pairwise distinct. We denote the set of species triples of a colored graph G by R_G.

Given a set of species triples R_G of a colored graph G, we denote the set of P_3s forming a species triple $r \in R_G$ by Q_r, and we define the set of all P_3s forming the species triples in R_G by $Q := \bigcup_{r \in R_G} Q_r$.

2 Later-Divergence-Time Graphs

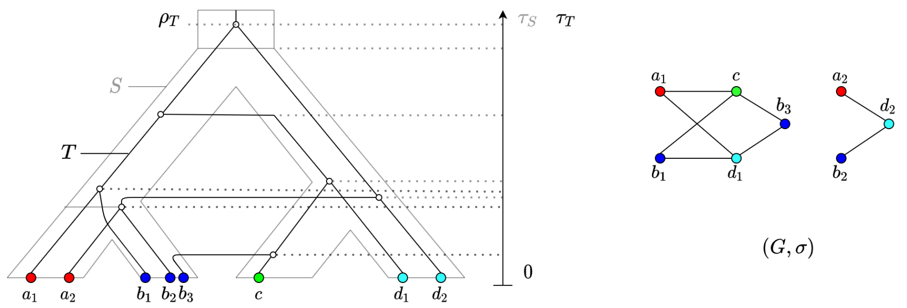

Figure 1: It shows a gene tree, T residing in a species tree, S (to the left) and the corresponding LDT graph (to the right). ρ_T denotes the root of T. This figure is taken from [9].

We begin this chapter by defining a certain type of graphs, namely Later-Divergence-Time (LDT) graphs.

To a gene tree (T, σ), species tree S with $\sigma(L(T)) \subseteq L(S)$ and corresponding time maps $\mathcal{T}_T$ and $\mathcal{T}_S$, respectively, the LDT graph [9] has vertex set

$$V := L(T)$$

and edge set

$$E := \{(x,\ y) \mid x,\ y \in L(T),\ \tau_T(lca_T(x,\ y)) < \tau_S(lca_S(\sigma(x),\ \sigma(y))\}.$$

Theorem 1 (LDT graph [9]). *A graph is an LDT graph if and only if it is a properly colored cograph (G, σ) such that R_G is consistent.*

In figure 1, we see how we can easily extract an LDT graph from a given species tree S and a gene tree T. In order to reconstruct a species and gene tree, we need to make sure that the data we are given, i.e., a properly colored graph, is an LDT graph. In the case where a given properly colored graph (G,σ) is not an LDT graph, we need to edit it such that the resulting graph (G^*,σ) becomes an LDT graph, i.e., it is a properly colored cograph such that R_{G^*} is consistent.

Theorem 2 ([9])**.** *LDT graph-modification is NP-complete.*

By theorem 2 we know that editing a properly colored graph G into an LDT graph such that the edit distance between G and G^* is minimal is NP-complete. Thus we will attempt to develop heuristics that edits a given properly colored graph into an LDT graph. Furthermore, we will formulate an ILP that will give us exact solutions in terms of the edit distance between the input graph and the edited graph. We do this for the purpose of benchmarking future heuristics. We now proceed by presenting additional theory that will help us with formulating this ILP.

For a consistent set of triples R we write $R \vdash (xy|z)$ if every tree that displays R also displays $xy|z$, and we say that R *infers* the triple $xy|z$. We will use the inference rules from [7] to infer additional triples in a consistent set of triples.

$$\{(ab|c),(ad|c)\} \vdash (bd|c) \qquad (i)$$

$$\{(ab|c),(ad|b)\} \vdash (bd|c),(ad|c) \qquad (ii)$$

$$\{(ab|c),(cd|b)\} \vdash (bd|c),(cd|a), \qquad (iii)$$

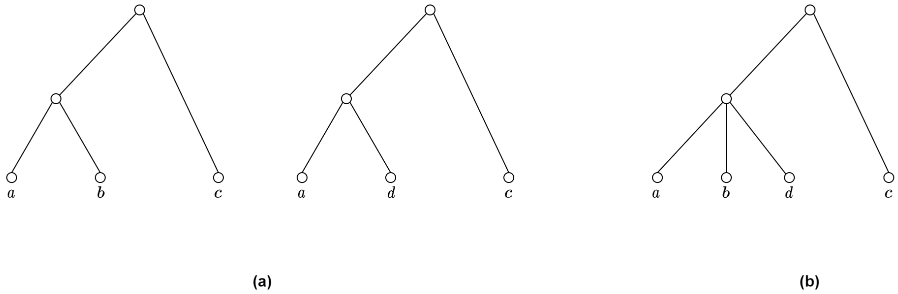

Figure 2: (a) shows two trees displaying triples $ab|c$ and $ad|c$. (b) shows a tree displaying both triples in (a).

In figure 2 (a) we have two triples, $ab|c$ and $ad|c$. In (b) we see the tree displaying both of those triples, and we can see that the path from a to d does not intersect the path from c to the root. Thus this tree also displays the triple $bd|c$, which is what inference rule (i) states.

We say that R is strictly dense if for all distinct leaves $x, y, z \in L$ there is exactly one triple $r \in R$ with $L_r = \{x, y, z\}$ [7]. Finally we give the definition of the closure of a consistent set of triples R followed by additional theory.

$\langle R \rangle$ is the set of all trees on L_R that display all the triples of R and by $\mathfrak{R}(T)$, we denote the set of all triples that are displayed by a tree T. We define the closure of a consistent set of rooted triples the same way as in [7],

$$\mathrm{cl}(R) = \bigcap_{T \in \langle R \rangle} \mathfrak{R}(T).$$

As a side note, we have by definition of the closure, $R \vdash (xy|z) \iff xy|z \in cl(R)$.

Theorem 3 ([7]). *Let R be a strictly dense triple set on L with $|L| \geq 3$. The set R is consistent if and only if cl$(R') \subseteq R$ hold for all $R' \subseteq R$ with $|R'| = 2$.*

Lemma 1 ([7]). *Let R be a strictly dense set of rooted triples. For all $L' = \{a, b, c, d\} \subseteq L_R$ we have the following* *statements: All triples inferred by rule* $[ii]$ *applied on triples* $r \in R$ *with* $L_r \subset L'$ *are contained in* R *if and only if all triples inferred by rule* $[iii]$ *applied on triples* $r \in R$ *with* $L_r \subset L'$ *are contained in* R*. Moreover, if all triples inferred by rule* $[ii]$ *applied on triples* $r \in R$ *with* $L_r \subset L'$ *are contained in* R *then all triples inferred by rule* $[i]$ *applied on triples* $r \in R$ *with* $L_r \subset L'$ *are contained in* R*.*

Lemma 2 ([7])**.** *Let* R *be a consistent set of triples on* L*. Then there is a strictly dense consistent triples set* R' *on* L *that contains* R*.*

2.1 ILP-formulation

We now present all of the necessities for this ILP, which includes the constants, variables, constraints, as well as the objective function. The input graph is $G = (V, E)$ and the final edited graph will be $G^* = (V, E^*)$.

Binary constants $E_{xy} \in \{0, 1\}$ such that

$$E_{xy} = 1 \iff (x, y) \in E.$$

Binary variables $\varepsilon_{xy}, T'_{\alpha\beta|\gamma} \in \{0, 1\}$ such that

$$\varepsilon_{xy} = 1 \iff (x, y) \in E^*,$$
$$T'_{\alpha\beta|\gamma} = 1 \iff \alpha\beta|\gamma \in R_{G^*}.$$

Objective function

$$min \sum_{x,y \in V} E_{xy}(1 - \varepsilon_{xy}) + \sum_{x,y \in V} \varepsilon_{xy}(1 - E_{xy}).$$

The purpose of the objective function is to minimize the symmetric difference between G and G^*.

The following numbered constraints are presented in [7] and as such we will not provide any proofs for those here.

Constraints

$$\varepsilon_{xy} = \varepsilon_{yx} \tag{ILP 0}$$

Constraint (ILP 0) is applied for all unordered pairs $x, y \in V, x \neq y$. This ensures G^* is undirected.

$$\varepsilon_{xy} = 0, \text{ for all } x, y \in V,\ \sigma(x) = \sigma(y). \tag{ILP 1}$$

Constraint (ILP 1) ensures G^* is properly colored.

$$\varepsilon_{wx} + \varepsilon_{xy} + \varepsilon_{yz} - \varepsilon_{xz} - \varepsilon_{wy} - \varepsilon_{wz} \leq 2 \tag{ILP 2}$$

Constraint (ILP 2) is applied for all ordered 4-tuples (w, x, y, z), which makes sure G^* is a cograph. This is illustrated in figure 3. The first three terms in (ILP 2) represent the three solid edges, whereas the remaining three terms represent the three dotted edges. For the purpose of turning any ordered 4-tuple of pairwise distinct vertices into a cograph, either at least one of the dotted edges has to be present or at least one of the solid edges has to be removed, given that no different order of the same four vertices result in a P_4.

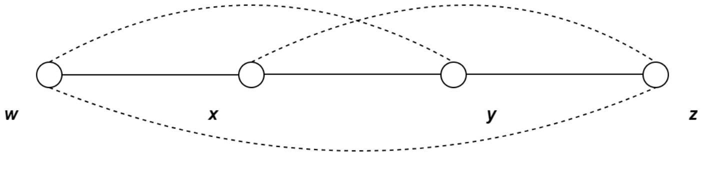

Figure 3: A P_4. To break this P_4 by insertion, at least one of the dotted edges need to be inserted. By deletion, at least one of the existing (solid) edges has to be deleted.

$$2T'_{\alpha\beta|\gamma} + 2T'_{\alpha\delta|\beta} - T'_{\beta\delta|\gamma} - T'_{\alpha\delta|\gamma} \leq 2 \tag{ILP 3}$$

By theorem 3 and lemma 1, we know that we can use the inference rule (ii) to verify consistency. As stated in [7], constraint (ILP 3) is a direct translation of rule (ii). This constraint is applied for all ordered 4-tuples $(\alpha,\beta,\gamma,\delta)$.

$$T'_{\alpha\beta|\gamma} + T'_{\alpha\gamma|\beta} + T'_{\beta\gamma|\alpha} = 1 \tag{ILP 4}$$

To get a maximal consistent set of species triples R_{G^*} we use lemma 2, i.e., we construct a strictly dense consistent set of triples. We do this by applying the constraint (ILP 4) for all unordered (α,β,γ).

Lastly, we set the following constraint for all $x,y,z \in V$ such that $\sigma(y),\sigma(z),\sigma(y)$ are pairwise distinct.

$$\varepsilon_{xy} + \varepsilon_{yz} + (1-\varepsilon_{xz}) - T'_{\sigma(x)\sigma(z)|\sigma(y)} \leq 2 \tag{ILP *}$$

This enforces the value of the variable to be $T'_{\sigma(x)\sigma(z)|\sigma(y)} = 1$ whenever there is a species triple, i.e., a P_3 with three distinct colors. If we dont have this constraint, then we could have a P_3 $\langle x,y,z\rangle$ such that $\sigma(x),\sigma(z),\sigma(y)$ are pairwise distinct, but $T'_{\sigma(x)\sigma(z)|\sigma(y)} = 0$, in which case the triple $\sigma(x)\sigma(z)|\sigma(y) \notin R_{G^*}$ and as such is not accounted for when checking for consistency. We provide a short proof for (ILP *).

Proof. A P_3 $\langle x,y,z\rangle$ has two edge, (x,y) and (y,z), thus $\varepsilon_{xy}+\varepsilon_{yz}+(1-\varepsilon_{xz}) = 3 \not\leq 2$ when x,y,z forms a P_3. Therefore $T'_{\sigma(x)\sigma(z)|\sigma(y)} = 1$ whenever there is a P_3 with three distinct colors. □

We give two additional optional constraints if one would like to limit the editing to only removing or only inserting edges. These are applied for all $x,y \in V$.

$$\varepsilon_{xy} \geq E_{xy} \tag{ILP +}$$

$$\varepsilon_{xy} \leq E_{xy} \tag{ILP -}$$

To restrict the editing to only inserting edges, we use constraint (ILP +). Similarly, if we only want to remove edges, we use constraint (ILP -). If both inserting and removing edges is allowed, then neither of these two constraints are used.

3 Heuristics

Given a properly colored graph, G, we want to edit it such that the resulting graph G^* is an LDT graph, i.e., it is a properly colored cograph with a set R_G that is consistent. To this end, we will use a library called Asymmetree [11], which includes a few methods we will need. Amongst these methods, we have a heuristic by Crespelle that runs in $\mathcal{O}(n^2)$ where n is the amount of vertices of a given graph G [5]. This heuristic edits a given graph G into a cograph, which we will denote by $cographEditing(G)$. Additionally we have a method that checks if a given graph G is a cograph by attempting to construct a cotree. If a cotree can be constructed then G is a cograph. To check for consistency we use the BUILD algorithm which is also included in this library. BUILD is a top-down recursive algorithm that makes use of an auxiliary graph called Aho graph, which we now define.

Definition 3 ([9]). *Let R be a set of rooted triples on the vertex set L. The Aho graph [R, L] has vertex set L and edge set $\{(x,y) \mid \exists z \in L : xy|z \in R\}$.*

Proposition 1 ([9]). *A set of triples R is compatible if and only if for each subset $L \subseteq L_R$ with $|L_R| > 1$ the graph [R, L] is disconnected.*

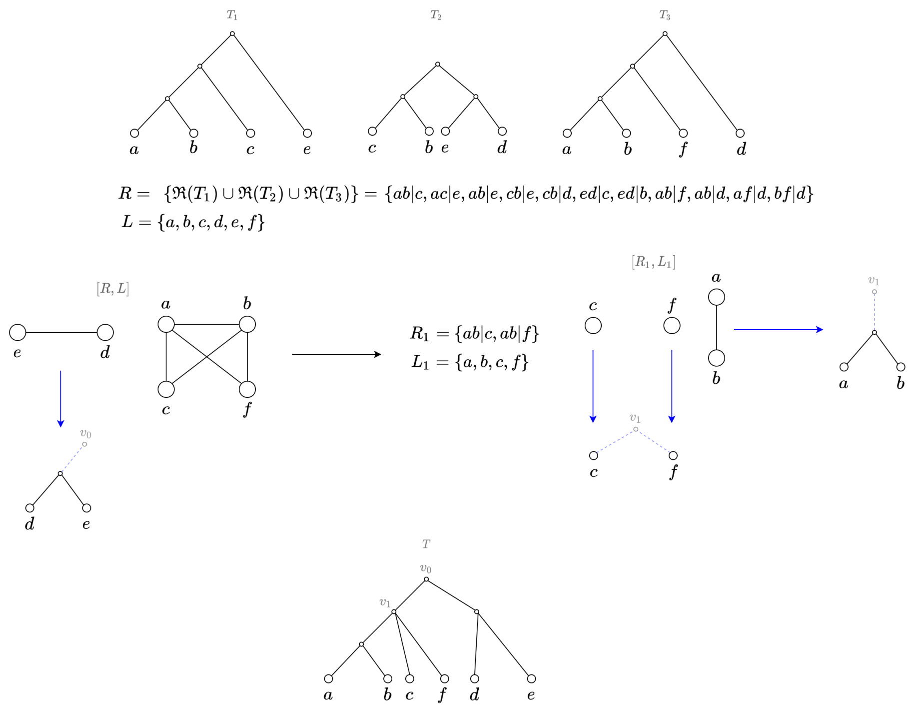

Figure 4: An example of BUILD applied to $[R, L]$, the result of which is the tree T. This example was taken from [10].

In figure 4 we see how the Aho graph is used in the BUILD algorithm to verify consistency. The trivial cases of BUILD that result in a tree, are when the vertex set of the Aho graph $|L| \in \{1, 2\}$. This is illustrated by the blue arrows in this figure. BUILD is applied to each component $[R_i, L_i]$ of $[R, L]$, and as such R_i becomes restricted to the vertex set L_i, i.e., $R_i := \{ab|c \in R \mid a, b, c \in L_i\}$. This is shown by the black arrow originating at the right component of $[R, L]$ with vertex set L_1 and triples set R_1, resulting in the Aho graph $[R_1, L_1]$, which has three components. No Aho graph is constructed for these components as they all have vertex count 1 or 2 and as such BUILD outputs the trees shown by the blue arrows. If BUILD outputs a tree for each component it is applied to, the resulting tree T displays all triples in R, as shown in the figure, and as such R is consistent. BUILD has a runtime of

$\mathcal{O}(|R||L|)$ [10] and the correctness of BUILD is a consequence of proposition 1 [9]. Furthermore we have a few heuristics for triples consistency editing. The heuristic we will use for triples consistency editing is Aho's BUILD with weighted mincut [1, 4], which makes use of BUILD to determine consistency, as well as edits the Aho graph such that consistency is upheld. This algorithm is used to find a maximum consistent triples set R' given an inconsistent triples set R such that $R' \subseteq R$. The way it works is by removing edges from a weighted aho graph $[R, L]$, R being an inconsistent set of triples, such that the resulting aho graph $[R', L]$ is of a consistent set of triples R', whilst minimizing the total weight of the edges removed. The weight of an edge $(a, b) \in [R, L]$ is based on the number of occurrences of triples $ab|x \in R, x \in L$. This is illustrated in figure 5 (c). The runtime for BUILD with weighted mincut is mainly dependent on the *Stoer-Wagner* [12] algorithm which, for a given graph $G = (V, E)$, has a runtime of $\mathcal{O}(|V|^3)$. For BUILD with weighted mincut on $[R, L]$, the runtime is therefore $\mathcal{O}(|L| * |L|^3) = \mathcal{O}(|L|^4)$ since the stoer-wagner algorithm is applied at most $|L|$ times. Finally we will need to develop a method that edits a given properly colored graph G into G^* such that the set of triples of G^* becomes consistent. We will denote this procedure as *triples editing*. In figure 5 (c) and (d) we see an example of triples editing on an inconsistent triples set R, and how cuts are made to make it consistent.

3.1 Challenges

For triples editing, the main challenge in editing G into G^* such that R_{G^*} becomes consistent, lies in destroying species triples without introducing new ones. When destroying a species triple, we need to destroy all P_3s forming that triple. To destroy a P_3 $\langle x, y, z \rangle$, we have three options.

(i) Remove the edge (x, y)

(ii) Remove the edge (y, z)

(iii) Insert the edge (x, z)

Each of these choices risk introducing one or more (possibly new) species triples. When removing edges, we make the following observation.

Observation 1. *If e is an edge in an induced K_3 of an undirected, loop-free graph G, then removing e from G will create a new P_3 in G.*

Proof. A P_3 is a path on three vertices, thus it has two edges. Let $G' = (V', E')$, be an induced subgraph of G, on three vertices. Removing an edge from G' will reduce its edge count to $|E'|-1$. For G' to be a P_3, we must have $|E'|-1 = 2$, thus $|E'| = 3$, which makes G' a K_3 since it is an undirected, loop-free graph with $|E'| = 3$ and $|V'| = 3$. □

For choices (i) and (ii), we know, by observation 1, that these will only introduce as many P_3s, as induced subgraphs of a given graph G, as there are K_3s as induced subgraphs, sharing the edge being removed. For case (iii) where we insert an edge, we take a look at figure 5 (f) to illustrate when we might introduce new species triples. Inserting any of the edges e_1, e_2, e_3 into the graph would create new P_3s $\langle b_2, a_2, c_3 \rangle$, $\langle d, a_2, c_3 \rangle$ and $\langle b_1, a_2, c_3 \rangle$, which would give us the triples, $BC|A = r_1$ and $DC|A = r_2$. If r_1, $r_2 \in R \setminus R' = F$, then we know that including those triples in R_{G^*} would make it inconsistent and we denote such triples as *forbidden* triples. In this case none of those triples are in R and as such we do not know whether they would make the set R_{G^*} inconsistent or not and we denote such triples as *new* triples. In figure 5 we show an example of triples editing. In the first step, we extract the species triples along with the P_3s forming them,

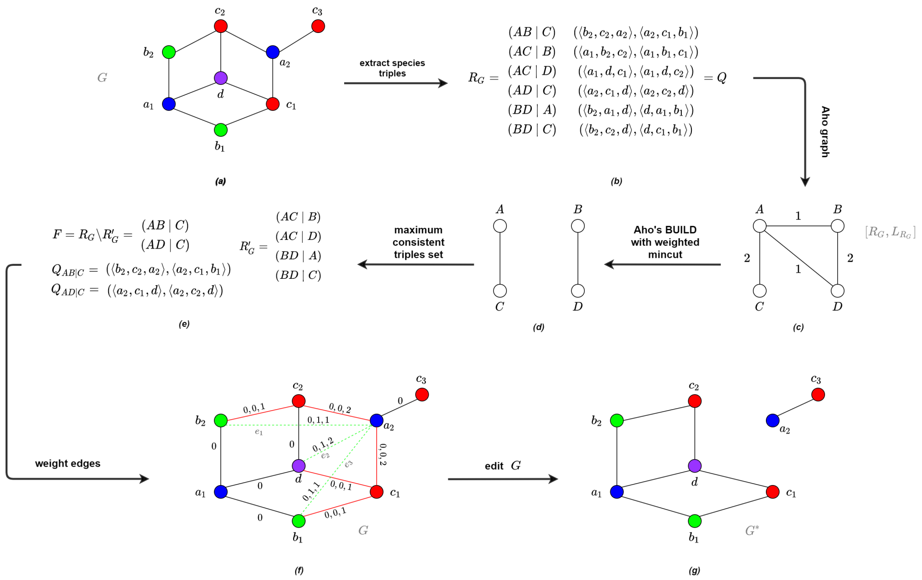

Figure 5: Example of a properly colored graph G with an inconsistent set of triples being edited into G^* such that R_{G^*} becomes consistent.

which gives us R_G and Q as shown in (b). This is done in $\mathcal{O}(|E||V|)$. Next, we apply BUILD with weighted mincut to the aho graph $[R, L_R]$ of which the result is the graph shown in (d). At this point we have a maximum consistent triples set $R_{G^*} \subseteq R_G$ and a set of forbidden triples $F = R_G \setminus R_{G^*}$, as shown in (e). We now weight the edges of G, specifically, the edges that are part of any P_3 $q \in Q$, which is shown in (f). For example, removing (red) the edge (c_2, a_2) introduces no forbidden or new triples, and is part of two P_3s in Q, hence the weight $0,0,2$. Similarly, by inserting (green) the edge (d, a_2) we would introduce no forbidden triples, one new triple, and would destroy two forbidden P_3s, hence the weight $0,1,2$. Finally, we edit the graph G, which is done by destroying all P_3s $q \in Q_r$ where $r \in F$, i.e., all of the forbidden P_3s forming the forbidden species triples in F. At this point, we edit G such that the resulting graph is the one shown in (g). We destroy all forbidden species triples by destroying all P_3s forming them, i.e., all forbidden P_3s. We have three options (i-iii) to destroy a P_3 and when choosing an edge to edit, we mainly want to minimize the introduction of forbidden species triples. Secondly we want the introduction of new species triples to be minimized, and thirdly, we want the amount of forbidden P_3s being destroyed to be maximized. Thus the edges that are edited in this step are (c_2, a_2) and (a_2, c_1).

3.2 Triples editing

We begin by introducing a procedure that checks if inserting an edge introduces new species triples, in which case those triples are returned.

Algorithm 1 `potentialTriples` finds potential species triples as a result of inserting a given edge to a properly colored graph G.

Input: Two non adjacent vertices a, b of a properly colored graph (G, σ).
Output: A set of species triples $R^\bullet$, created by inserting the edge (a, b).

```
procedure POTENTIALTRIPLES(a, b)
    R• ← ∅
    for c ∈ N(a) do
        if σ(b), σ(c), σ(a) are pairwise distinct and c ∉ N(b) then
            R• ← R• ∪ {σ(b)σ(c)|σ(a)}
    for c ∈ N(b) do
        if σ(a), σ(c), σ(b) are pairwise distinct and c ∉ N(a) then
            R• ← R• ∪ {σ(a)σ(c)|σ(b)}
    return R•
```

Lemma 3. *Algorithm 1 terminates and correctly outputs a set of potential species triples introduced by inserting a given edge into a given graph $G = (V, E)$. The runtime is $\mathcal{O}(|V|)$.*

Proof. The existence of a species triple $AB|C$ in a colored graph means there is a P_3 $\langle a, c, b\rangle$ such that $\sigma(a) = A$, $\sigma(b) = B$, $\sigma(c) = C$ are pairwise distinct, thus by checking that c does not form a K3 with a and b and that their colors are pairwise distinct, we ensure $R^\bullet$ consists of species triples that would be introduced as a result of inserting the edge (a, b) to G. Since G is a finite graph, $N(a)$ and $N(b)$ are finite and as such algorithm 1 terminates after at most $|N(a)-2|+|N(b)-2|$ iterations. If $N(a)$ and $N(b)$ are of maximum size then $|N(a)|=|N(b)|=|V|-2$ since a and b are adjacent to all other vertices but themselves. This results in the runtime $\mathcal{O}(2*(|V|-2))=\mathcal{O}(|V|)$. $\square$

In the following procedure, we make use of `potentialTriples` in order to weight all edges of G based on their occurrences in any P_3, $q \in Q_r$ where $r \in F$, F being a given set of forbidden species triples. Furthermore, edges are also weighted based on the amount of forbidden and new species triples that are introduced as a result of editing an edge. We will denote the set of forbidden P_3s that would be destroyed as a result of editing an edge e, with $\mathfrak{D}_e \subseteq Q_r$, where $r \in F$. Additionally we write $\mathfrak{P}_e$ for the set of species triples introduced by editing an edge e.

Finding the set of all species triples R along with the corresponding P_3s, Q_r for all $r \in R$ and the set of all induced K_3s with pairwise distinct colors, K, of a properly colored graph G, is done in $\mathcal{O}(|E||V|)$. We get the set of forbidden triples of G, F, by applying BUILD with weighted mincut on $[R, L_R]$ which, as mentioned previously, has a runtime of $\mathcal{O}(|L_R|^4)$. Thus the initialization of the following algorithms is done in $\mathcal{O}(|E||V|+|L_R|^4)$.

Algorithm 2 `weightEdges` weights deletion and insertion edges of G.

1: **Initialize:**
  $G \leftarrow$ A properly colored graph
  $R \leftarrow$ A set of triples extracted from G
  for $r \in R$ **do**
    $Q_r \leftarrow$ the set of P_3s forming the triple r
  $Q \leftarrow \bigcup_{r \in R} Q_r$
  $K \leftarrow$ A set of all induced K_3s (pairwise distinct colors) of G
  $F \leftarrow$ A set of forbidden triples
2: **procedure** $\textsc{weightEdges}(G, R, K, Q, F)$
3:   **for** $e \in E(G)$ **do**
4:     $\mathfrak{P}_e \leftarrow \emptyset$
5:     $\mathfrak{D}_e \leftarrow \emptyset$
6:   **for** $AB|C \in F$ **do**
7:     **for** $(a, b, c) \in Q_{AB|C}$ **do**
8:       $e_1 \leftarrow (a, b)$ ▷ insertion edge
9:       $e_2 \leftarrow (a, c)$ ▷ deletion edge
10:       $e_3 \leftarrow (b, c)$ ▷ deletion edge
11:       **for** $e \in \{e_1, e_2, e_3\}$ **do**
12:         $\mathfrak{D}_e \leftarrow \mathfrak{D}_e \cup \langle a, c, b\rangle$
13:       $R^\bullet \leftarrow$ `potentialTriples`(e_1) ▷ species triples from inserting e_1
14:       **for** $r \in R^\bullet$ **do**
15:         **if** $r \notin R$ **or** $r \in F$ **then**
16:           $\mathfrak{P}_{e_1} \leftarrow \mathfrak{P}_{e_1} \cup r$
17:       **for** $k \in K$ **do**
18:         **if** $(a, c) \in E(k)$ **then**
19:           **if** $AC|* \notin R$ **or** $AC|* \in F$ **then** ▷ $*$ is used as a wildcard
20:             $\mathfrak{P}_{e_2} \leftarrow \mathfrak{P}_{e_2} \cup AC|*$
21:         repeat steps 18 to 20 for e_3
22:   Set the weight of each edge e to $(|\mathfrak{P}_e \cap F|, |\mathfrak{P}_e \setminus R|, |\mathfrak{D}_e|)$

Lemma 4. *Algorithm 2 terminates and correctly weights insertion and deletion edges of a given graph $G = (V, E)$ based on their occurrences in P_3s that need to be destroyed in order to destroy all species triples in F, and the amount of forbidden and new species triples they would introduce if deleted or inserted. The runtime for algorithm 2 is $\mathcal{O}(|F||Q|n^3 + |L_R|^4)$, where $n = |V|$.*

Proof. Using observation 1 we know that removing an edge from a K_3 with vertices whose colors are pairwise distinct is the only way to introduces a new P_3 that forms a species triples. By going through all K_3s that share a given edge, we can correctly weight deletion edges. We simply check, for all K_3s, if removing an edge e introduces species triples that are either not in R or are forbidden species triples, in which case these species triples are included in $\mathfrak{P}_e$. Similarly, for insertion edges, we check, for all $r \in R^\bullet$, if r either not in R or if it is in F, in which case r is included in $\mathfrak{P}_e$. Consequently, we can get the amount of forbidden and new species triples an edge e would introduce if edited, by $|\mathfrak{P}_e \cap F| = x$ and $|\mathfrak{P}_e \setminus R| = y$, respectively. As for the amount of P_3s an edge e would destroy as a result of being edited, we simply include all $q \in Q_r$, $r \in F$, in $\mathfrak{D}_e$. Thus we can correctly weight all of the edges $e \in E(G)$ by setting the weight of e to $(x, y, |\mathfrak{D}_e|)$. Since the input graph is finite, so are all the sets that are iterated through, thus algorithm 2 terminates. we begin by initializing the attributes by going through all edges and then for every P_3 $\langle a, c, b \rangle$ in every forbidden triple, we weight all of the edges of the P_3 including the insertion edge (a, b). To weight the insertion edge we first get the species triples that would be introduced by inserting the edge, $R^\bullet$, in $\mathcal{O}(|V|)$ and then iterate through $R^\bullet$, of which we have the upper bound $|R^\bullet| \leq \binom{n}{3} = n^3 - 2n^2 + 2n$ for $n = |V| \geq 3$, resulting in $\mathcal{O}(n^3)$. We can also use this as an upper bound for $|K|$. For deletion edges we iterate through K, hence the product $(|V| + |R^\bullet| + |K|)$. Thus we have the runtime $\mathcal{O}(|E| + |F||Q|(|V| + |R^\bullet| + |K|)) = \mathcal{O}(|E| + |F||Q|n^3) = \mathcal{O}(|F||Q|n^3)$ since $|E| \leq \binom{n}{2} < \binom{n}{3}$ for $n \geq 5$. Additionally, when accounting for the initialization, the resulting runtime is $\mathcal{O}(|F||Q|n^3 + |E||V| + |L_R|^4) = \mathcal{O}(|F||Q|n^3 + |L_R|^4)$ since $n^3 \geq |V||E| = n * \binom{n}{2} = n * \frac{n*(n-1)}{2} = \frac{n^3-n^2}{2}$.

□

For every P_3 we want to remove, we will choose the best edge to edit out of the choices (i), (ii) and (iii), based on the priority

1. edge that introduces the least amount of forbidden triples,
2. edge that introduces the least amount of new triples,
3. edge with the most occurrences in all $q \in Q_r$, for all $r \in F$.

Finally we introduce an algorithm that finds and returns the best edge to edit.

Algorithm 3 `bestEdge` finds the best edge to delete or insert out of three choices (i), (ii) and (iii).

Input: Three weighted edges and one (optional) constraint.
Output: An optimal edge to edit, out of the three input edges.

1: **procedure** BESTEDGE($e_1, e_2, e_3, insertion = False, deletion = False$)
2:     **if** insertion **then**
3:         **return** e_1
4:     **if** deletion **then**
5:         do steps 9 to 15 for e_2 and e_3 only
6:     $(i_1, j_1, k_1) \leftarrow \omega(e_1)$
7:     $(i_2, j_2, k_2) \leftarrow \omega(e_2)$
8:     $(i_3, j_3, k_3) \leftarrow \omega(e_3)$
9:     $E^* \leftarrow \{e_1, e_2, e_3\}$
10:     for $e \in E^*$ with weight $\omega(e) = (i, j, k)$, exclude from E^*, all e such that $i \neq min\{i_1, i_2, i_3\}$
11:     **if** $|E^*| > 1$ **then**
12:         exclude from E^*, all e such that $j \neq min\{j_1, j_2, j_3\}$
13:         **if** $|E^*| > 1$ **then**
14:             exclude from E^*, all e such that $k \neq max\{k_1, k_2, k_3\}$
15:     **return** an arbitrary edge $e \in E^*$

Lemma 5. *Algorithm 3 correctly finds and returns the best deletion or insertion edge based on the priority described above. It runs in constant time.*

Proof. To ensure algorithm 3 returns the best edge in terms of the priorities above, we simply choose the edge e with weight $\omega(e) = (i, j, k)$ such that i is minimum, and in the case where there is more than one such edge, we choose, out of those edges, the edge e such that j is minimum. If there is more than one such edge, we choose, out of those edges, an edge e such that k is maximum. In the case where multiple edges have weights (i,j,k) where i,j are of minimum value, and k is of maximum value, we simply choose an arbitrary edge, out of those edges. This ensure we correctly choose the best edge. We simply compare the attributes of three different edges against each other and as such the runtime is $\mathcal{O}(1)$. If we are given a constraint then we simply check if $insertion = True$, in which case we return e_1 as it is the only insertion edge. For deletion, we exclude e_1 from the comparisons and compare e_2 and e_3 similarly to how they are compared without constraints. □

Algorithm 4 `TriplesEditing` edits a properly colored graph G by removing all P_3s forming forbidden species triples.

Initialize:
  $G \leftarrow$ A properly colored graph
  $R \leftarrow$ A set of triples extracted from G
  for $r \in R$ **do**
    $Q_r \leftarrow$ the set of P_3s forming the triple r
  $Q \leftarrow \bigcup_{r \in R} Q_r$
  $K \leftarrow$ A set of all induced K_3s (pairwise distinct colors) of G
  $F \leftarrow$ A set of forbidden triples
procedure TRIPLESEDITING$(G,R,K,Q,F,deletion = False, insertion = False)$
  `weightEdges`(G,R,K,Q,F)
  for $AB|C \in F$ **do**
    for $a,c,b \in Q_{AB|C}$ **do**
      $e_1 \leftarrow (a,b)$
      $e_2 \leftarrow (a,c)$
      $e_3 \leftarrow (b,c)$
      $e \leftarrow$ `bestEdge`$(e_1,e_2,e_3,deletion,insertion)$
      if $e = e_1$ **then**
        insert e to G
      else
        remove e from G

Algorithm 4 weights the edges of G using algorithm 2 and then destroys all species triples in F by destroying their corresponding P_3s. Since F and Q are finite sets, we can ensure this procedure terminates. However, it does not always correctly edit G into G^* such that R^*_G is consistent. This is due to the fact that it is possible that out of all of the choices we have to destroy a P_3, all of those choices could introduce forbidden triples.

In order to ensure R_{G^*} is consistent, we can repeat algorithm 4 n times or until R_{G^*} becomes consistent. The reason we limit this to n iterations is because it is not guaranteed to terminate without a limit. This is due to the fact that we are allowed to edit the same edge at different iterations and as such we may end up removing and inserting edges repeatedly. However, if we were to restrict ourselves to only inserting or only deleting edges, then we would be able to ensure the procedure terminates and correctly yields a graph G^* such that R_{G^*} is consistent.

Lemma 6. *Algorithm 4 has a runtime of* $\mathcal{O}(|F||Q|n^3+|L_R|^4)$*, where* n *is the amount of vertices in the given graph.*

Proof. Since we initialize all of the necessary sets and use `weightEdges` we have the runtime $\mathcal{O}(|F||Q|n^3+|L_R|^4)$. We then go through every P_3 of every forbidden triple and as such we get $\mathcal{O}(|F||Q|n^3+|L_R|^4+|F||Q|) = \mathcal{O}(|F||Q|n^3+|L_R|^4)$. $\square$

Lemma 7. *If editing of* G *is restricted to insertion or deletion, repeating algorithm 4 until* R_G *becomes consistent will terminate.*

Proof. Consider the case where only deletion is allowed. At every iteration, we either have an empty set F in which case R_G becomes consistent and the algorithm terminates, or we have at least one P_3 that needs to be destroyed. Thus we end up deleting at least one edge at every iteration. We can repeat this process until we end up with an empty edge set, in which case, the set of triples $R_G = \emptyset$ and as such has become consistent. This ensures termination and that the resulting graph G^* has a consistent set of triples, for deletion only.

Now consider the case where only insertion is allowed. Similarly to deletion only, we insert at least one edge at every iteration. We do this until R_{G^*} becomes consistent or G^* becomes a complete multipartite graph. If G^* is a complete multipartite graph then it does not contain a P_3 as an induced subgraph and R_{G^*} is consistent. This ensures termination and that the resulting graph has a consistent set of triples, for insertion only. □

3.3 LDT editing

Now that we have the tools to edit a given properly colored graph G into G^* such that R_{G^*} becomes consistent, we can try to combine this with the existing cograph editing. We use the triples editing and restrict the editing to insertions or deletions only in order to guarantee R_{G^*} becomes consistent. We then check if G^* is a cograph, in which case it is an LDT graph. If not, we apply cograph editing and then make sure it is properly colored, i.e., if it is not properly colored we disconnect any vertices x, y such that $\sigma(x) = \sigma(y)$. Finally we check once again if G^* is a cograph and if R_{G^*} remains consistent.

Algorithm 5 `LDTediting` attempts to edit a properly colored graph G into an LDT graph. Returns $True$ if G was edited into an LDT graph and $False$ otherwise.

Initialize:
 $G \leftarrow$ A properly colored graph
 $R \leftarrow$ A set of triples extracted from G
 for $r \in R$ **do**
  $Q_r \leftarrow$ the set of P_3s forming the triple r
 $Q \leftarrow \bigcup_{r \in R} Q_r$
 $K \leftarrow$ A set of all induced K_3s (pairwise distinct colors) of G
 $F \leftarrow$ A set of forbidden triples
 $del, ins \leftarrow$ optional booleans for editing restrictions ($False$ by default)
procedure $\text{LDTEDITING}(G, R, K, Q, F)$
 `triplesEditing`$(G, R, K, Q, F, del, ins)$
 if G is a cograph **then**
  return $True$
 else
  `cographEditing`(G)
  if G is not properly colored **then**
   remove all $(x, y) \in E(G)$ such that $\sigma(x) = \sigma(y)$
  if G is a cograph and R_G is consistent **then**
   return $True$
 return False
return `LDTediting(G, R, K, Q, F)`

Lemma 8. *Algorithm 5 has a runtime of $\mathcal{O}(|F||Q|n^3 + |L_R|^4)$, where n is the amount of vertices in the given graph.*

Proof. The initialization along with `triplesEditing` is done in $\mathcal{O}(|F||Q|n^3 + |L_R|^4)$. Checking if a given graph G is a cograph is done in linear time. In the case where G needs to be edited into a cograph, we do this in $\mathcal{O}(n^2)$. additionally, checking for consistency is done in $\mathcal{O}(|R||L_R|)$, and editing G into a properly colored graph is done in $\mathcal{O}(n)$. Thus the resulting runtime is $\mathcal{O}(|F||Q|n^3 + |L_R|^4 + n + n^2 + n + n + |R||L_R|) = \mathcal{O}(|F||Q|n^3 + |L_R|^4)$. □

Lemma 9. *Repeating Algorithm 5 until G becomes an LDT graph by means of deletion, terminates and correctly returns an LDT graph.*

Proof. Similar to the proof for lemma 6, we know that at each iteration of algorithm 5 we remove at least one edge if editing is restricted to deletion, because in each iteration we have an LDT graph or at least one of the three properties of an LDT graph is not satisfied. We repeat this until G becomes an LDT graph or until G has no edges, in which case G is an LDT graph, because G will be properly colored, R_G will be empty and no induced P_4 will exist in G. □

The reason lemma 9 does not apply when restricted to insertion is because cograph editing does not take into account the coloring of the vertices and as such, while we may end up with a cograph whose set of triples is consistent, the graph may not be properly colored. Even if in this case, deletion is allowed to edit the resulting graph G^* such that it becomes properly colored, G^* is not guaranteed to remain a cograph. Thus we cannot ensure termination.

4 Results

To benchmark all the heuristics, we use the Asymmetree library to simulate species and gene trees from which we extract LDT graphs.

```
import asymmetree.treeevolve as te
S = te.simulate_species_tree(10, model='innovation')
TGT = te.simulate_dated_gene_tree(S, dupl_rate=0.5,
                    loss_rate=0.5, hgt_rate=0.5,
                    prohibit_extinction='per_family',
                    replace_prob=0.0)
OGT = te.observable_tree(TGT)
ldt = ldt_graph(OGT, S)
```

We simulate 100 pairs of species and gene tree with $n \in \{10, 14, 18, 30, 40, 50\}$ surviving genes, i.e., 600 pairs in total. These pairs will give us LDT graphs $G = (V, E)$ with $|V| = n \in \{10, 14, 18\}$. For each such pair, we extract an LDT graph which we then perturb using probabilities $(p_{ins}, p_{del}) = p \in P$ where p_{ins} and p_{del} denotes the probability to insert and remove an edge, respectively, and

$$P = \{(0.15, 0.15), (0.3, 0.3), (0.5, 0.5), (0.15, 0.5), (0.5, 0.15)\}.$$

To clarify, each extracted LDT graph is perturbed six times, once for each $p \in P$, such that it remains properly colored, which results in the graphs $G_1, \ldots, G_5$.

```
from asymmetree.tools.GraphTools import disturb_graph
G_1=disturb_graph(ldt, insertion_prob=0.15, deletion_prob=0.15)
G_2=disturb_graph(ldt, insertion_prob=0.3, deletion_prob=0.3)
G_3=disturb_graph(ldt, insertion_prob=0.5, deletion_prob=0.5)
G_4=disturb_graph(ldt, insertion_prob=0.15, deletion_prob=0.5)
G_5=disturb_graph(ldt, insertion_prob=0.5, deletion_prob=0.15)
```

For each perturbed graph $G_{n,p,i}$ for $1 \leq i \leq 100$, we edit it using the different heuristics independently to see how well they perform in terms of reconciling the other properties of LDT graphs, e.g., if we apply cograph editing to G, how often does R_{G^*} become consistent as a result of this editing. We begin by looking at cograph editing, triples editing with $k = 100$ iterations (to ensure termination), triples editing with deletion and triples editing with insertion. When the resulting graph G^* becomes an LDT graph, we will denote the edit distance between G_i and G_i^* by A_i, and we will denote the edit distance between G_i and G_i' by X_i where G_i' is the graph obtained by the ILP. For triples editing restricted to deletion or insertion, we compare A_i to the minimum edit distance obtained by the ILP being restricted to deletion and insertion, respectively. Due to time constraints, we have only generated ILP solutions for graphs with 10, 14 and 18 vertices since the time required for the ILP increases significantly as the size of the input graph increases. As such we are only able to compare edit distances for heuristics applied to graphs of those sizes.

4.1 Cograph and triples editing

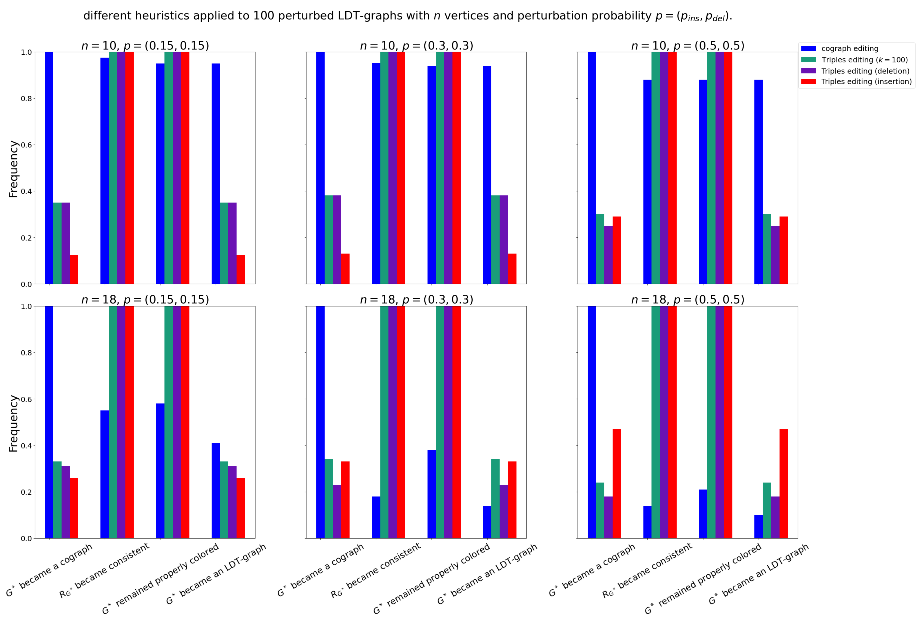

Figure 6: Results of different heuristics being applied to non LDT graphs with $n = 10, 18$ vertices and three different perturbations for each value of n. These plots show the frequency of each heuristic correcting properties of LDT graphs.

For cograph and triples editing, we look at three different perturbation probabilities,

$$p \in \{(0.15, 0.15), (0.3, 0.3), (0.5, 0.5)\}$$

and $n \in \{10, 18, 40, 50\}$ and we see in figure 6 and 7 that the frequency of these heuristics, editing G into an LDT graph, decreases as the vertex count increases. We also see that the frequency of cograph editing correcting the other properties of LDT graphs decreases quite significantly as the vertex count increases and the same is true for triples editing making G^* into a cograph. The frequency of triples editing with deletion turning G^* into a cograph does however seem to decrease very slowly as n increases, regardless of the perturbation probability. In terms of the frequency of editing G into an LDT graph, cograph editing seems to perform better on graphs with lower perturbations, but for graphs with $n \geq 40$ it seems that cograph editing no longer manages to edit G into an LDT graph, because the resulting graph does not remain properly colored, nor does it have a consistent set of triples. This seems to be the case regardless of the perturbation probability. Triples editing with $k = 100$ slightly outperforms triples editing with deletion on smaller graphs regardless of the perturbation probability, as we can see in figure 7, but looking at figure 8, we see that triples editing with deletion outperforms triples editing with $k = 100$ on larger graphs in all but one case, when $n = 40$ and $p = (0.15, 0.15)$. Triples editing with insertion performs the worst on larger graphs, as is seen in figure 7. It does however perform the best when $n = 18$ and $p = (0.5, 0.5)$.

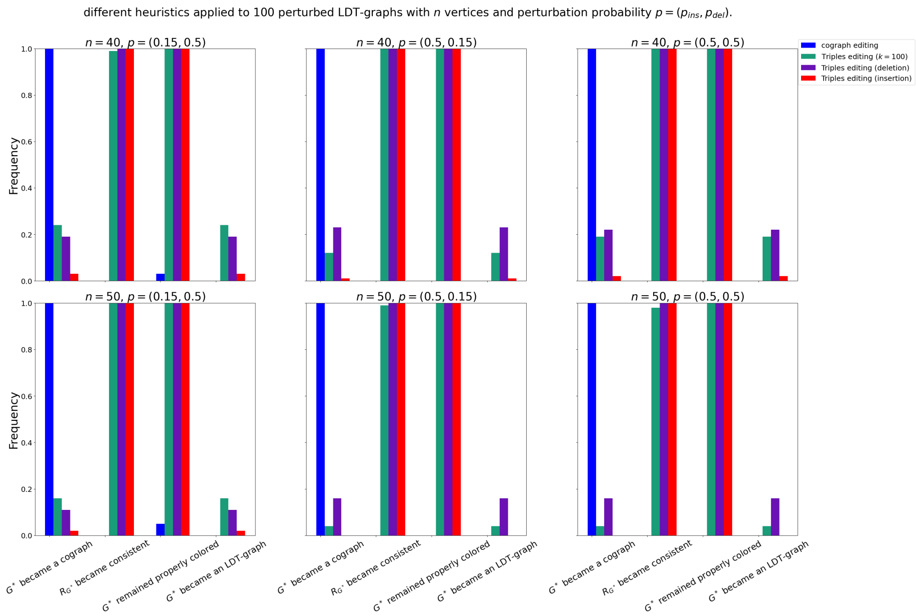

Figure 7: Results of different heuristics being applied to non LDT graphs with $n = 40, 50$ vertices and three different perturbations for each value of n. These plots show the frequency of each heuristic correcting properties of LDT graphs.

How well these heuristics perform in terms of the edit distance to an LDT graph is seen in figure 8. We observe that cograph editing is the best performing one for graphs of these sizes, regardless of the perturbation probability. Exactly how these methods perform on larger graphs is unclear as we have yet to benchmark them on larger graphs such as $n = 50$ or $n = 100$, although it does seem like triples editing with $k = 100$ and deletion perform very similarly with triples editing (deletion) being slightly better. Comparing the triples editing variations for $n = 18$, we see that for $p = (0.15, 0.15)$, triples editing (deletion) performs the best out of the three, while for $p = (0.5, 0.5)$, triples editing (insertion) is the one that performs the best out of the three. This means that one could apply triples editing with deletion or insertion based on what the perturbation probability is, in order to to attain a more optimal edit distance, since these variations seem to have a certain probability p for which the median of ratios is minimum out of the variations of triples editing.

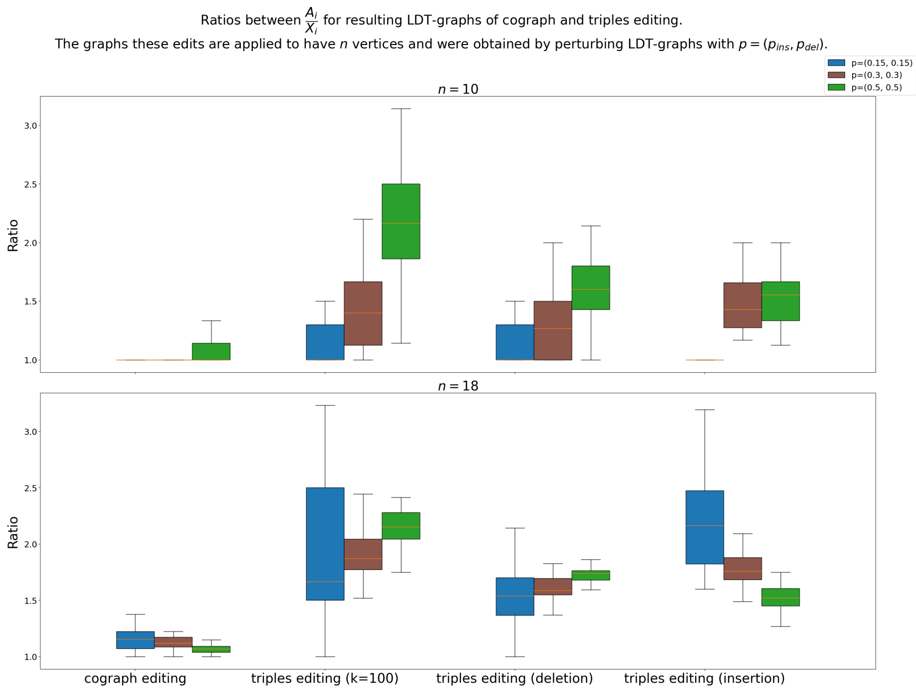

Figure 8: Results of different heuristics being applied to non LDT graphs with $n = 10, 18$ vertices and three different perturbations for each value of n. These plots show the ratio of the edit distances A_i and X_i.

4.2 LDT editing

We also benchmark LDT editing (i) with no restrictions and up to $k = 100$ iterations of triples editing, LDT editing (ii) with deletion restriction for triples editing and LDT editing (iii) with insertion restriction for triples editing. We apply these edits to graphs obtained by perturbing LDT graphs with $p_1 = (0.15, 0.15)$, $p_2 = (0.3, 0.3)$, $p_3 = (0.5, 0.5)$, $p_4 = (0.15, 0.5)$, $p_5 = (0.5, 0.15)$ and then look at the ratio $\frac{A_i}{X_i}$ where A_i is the edit distance between G_i^* and G_i, and X_i is the edit distance between G_i' and G_i, G_i' being the graph obtained by the ILP and G_i^* the resulting graph of LDT editing. These comparisons are only made for those G_i^* that become LDT graphs.

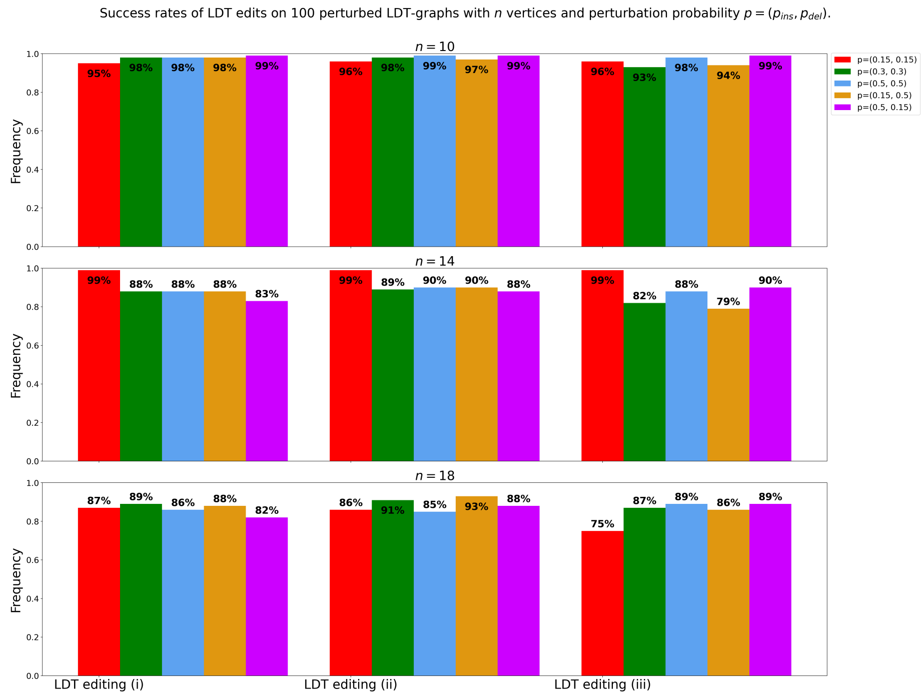

Figure 9: Shows how often LDT editing (i) with no restriction, LDT editing (ii) with deletion, and LDT editing (iii) with insertion, return an LDT graph. This figure shows the success rate for 10, 14 and 18 vertices with varying perturbation probabilities.

In terms of how often all variations of LDT editing successfully edits G into an LDT graph, we see by comparing the frequencies in figure 9 and 10 that the frequency does indeed decrease as n becomes larger. In some cases it decreases slower, such as for LDT editing (ii) with $p = (0.15, 0.5)$. For smaller graphs, such as those with $n \leq 18$, the effect of perturbation probability p on the frequency seems to be quite insignificant, but for graphs with $n \geq 30$, p does seem to have more of an effect on the frequency for LDT editing (i) and (ii). For LDT editing (iii), the frequencies do seem quite similar for different perturbation probabilities. While there may be one variation of LDT editing that generally has a higher success rate, the difference seem very negligible for most n. In the case where $n = 50$, LDT editing (iii) seems to generally be a better choice than LDT editing (i), i.e., for all probabilities p tested. Additionally, when $p = (0.15, 0.15)$, LDT editing (iii) has a 27% and 17% higher success rate than LDT editing (i) and (ii), respectively. This makes LDT editing (iii) the best performing, in terms of success rate, for $p = (0.15, 0.15)$ and $n = 50$.

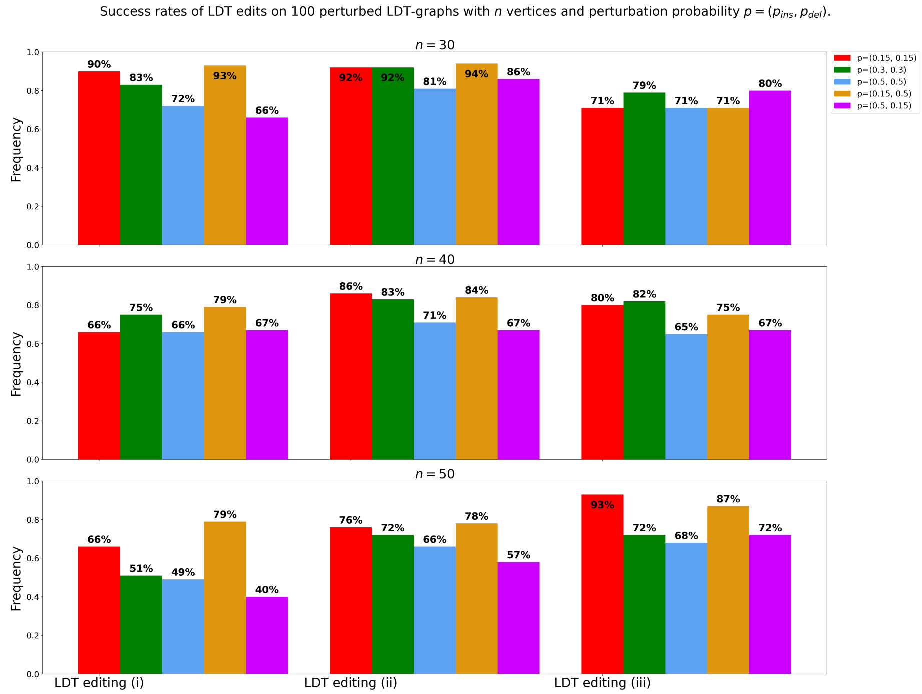

Figure 10: Shows how often LDT editing (i) with no restriction, LDT editing (ii) with deletion, and LDT editing (iii) with insertion, return an LDT graph. This figure shows the success rate for 30, 40 and 50 vertices with varying perturbation probabilities.

Looking at figure 11 we see that LDT editing (iii) performs the worst, in terms of edit distance, when the perturbation probability is low such as $p = (0.15, 0.15)$, and it gets considerably worse as n gets slightly larger. Looking at the same probability for LDT editing (i) and (ii), we see that while the median ratio increases between $n = 14$ and $n = 18$, it is not as significant as it is for LDT editing (iii). For $p = (0.15, 0.5)$, LDT editing (iii) also performs significantly worse as n increases, compared to LDT editing (i) and (ii). Additionally, we see that LDT editing (i) and (ii) perform quite similarly, with (ii) generally performing slightly better as the vertex count increases. As such, LDT editing (ii) seems to generally be the better option out of the three variations for larger graphs. We do however note that when $p = (0.5, 0.15)$, LDT editing (iii) does perform the best. This is true for all values of n tested as we can see by comparing the purple boxes in figure 11.

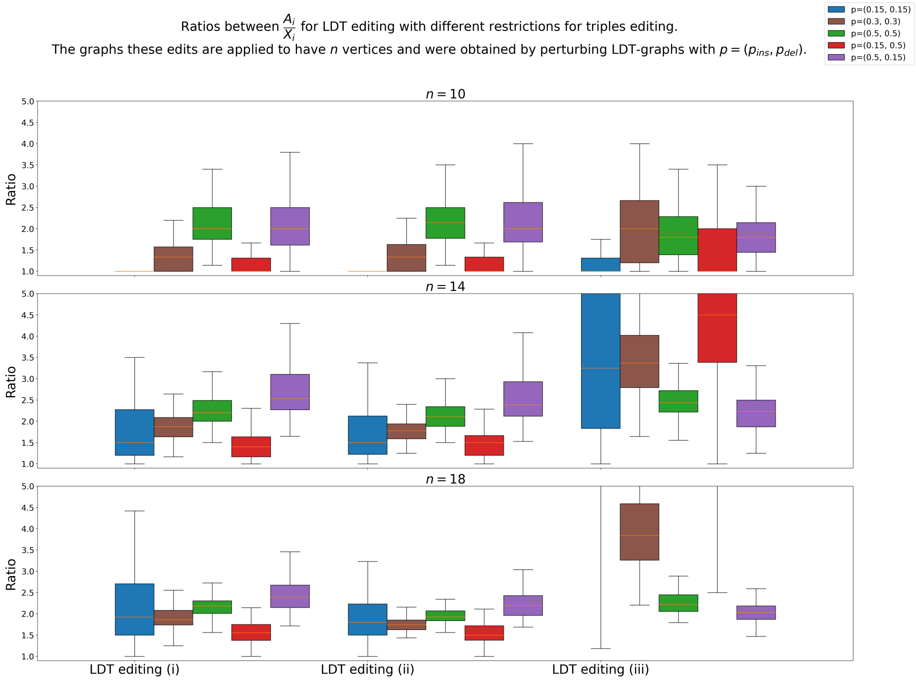

Figure 11: Edit distance to resulting LDT graph of LDT editing compared to the minimum edit distance to an LDT graph obtained by the ILP.

4.3 BUILD

Finally, we look at how often BUILD with weighted mincut results in T being binary. We do this for the sake of potential improvements to triples editing. Looking at table 1, we see that T is mainly binary for all variations. The rate at which T is binary for triples editing with insertion when $n = 40$, is

n	p	triples editing ($k=100$)	triples editing (deletion)	triples editing (insertion)
20	$(0.15,0.5)$	64%	81%	55%
20	$(0.5,0.15)$	66%	66%	82%
20	$(0.5,0.5)$	81%	68%	70%
40	$(0.15,0.5)$	57%	83%	89%
40	$(0.5,0.15)$	79%	69%	99%
40	$(0.5,0.5)$	84%	64%	100%

Table 1: Shows the percentage of how often the tree T that BUILD with weighted mincut returns when a set of triples is consistent, is binary. This is tested on 100 LDT graphs, for each vertex count n, that were each perturbed three times with different probabilities.

very high, and interestingly, when $p = (0.5, 0.5)$, T seems to always be binary.

5 Conclusion

The success rate of the different variations of LDT editing decreases as the size of the input graph increases and as such these variations of the LDT editing heuristic are less reliable for larger graphs. In such a case where the input graph G is large, restricting LDT editing to deletion would make it completely reliable as G^* is guaranteed to be an LDT graph. This is also true when LDT editing is restricted to insertion only, although cograph editing would need to be modified to account for the coloring of the vertices so that G^* remains properly colored. For smaller graphs G such as those with $n \leq 18$ vertices, we can slightly modify LDT editing based on n in order to attain a more optimal edit distance. Since cograph editing performed the best in terms of the frequency of editing a given properly colored graph G into an LDT graph, as well as the edit distance, for graphs with ($n \leq 10$), it would be best to apply cograph editing first when LDT editing such graphs, as this would be enough to edit G into an LDT graph about 80% of the time, such that the edit distance is near optimal. Similarly, when $n \leq 18$ and the probability of inserting an edge is high while the probability of deleting an edge is low, LDT editing (iii) is the better choice, since it results in a more optimal edit distance, compared to the other variations of LDT editing. If the perturbation probability p is unknown then LDT editing (ii) would be the better option as it did generally perform the best, i.e., it resulted in a more optimal edit distance for most probabilities p.

Since BUILD with weighted mincut returns a binary tree T in most cases when used in triples editing, and a binary tree output by BUILD displays exactly one triple for each $\{a, b, c\} \in \binom{L(T)}{3}$, as $\mathfrak{R}(T)$ is strictly dense [7], we can use this to identify new triples as forbidden or not in triples editing. We know, from the inference rules presented in section 2, which additional triples are allowed, and we know that at least any new triple r with $L_r \in \binom{L(T)}{3}$ is forbidden.

To summarize, we have looked at a phylogenetic method of inferring horizontal gene transfer and we have provided a formulation as well as an implementation (`https://github.com/Rezuxi/LDT_ILP`) of an ILP that edits a given properly colored graph into an LDT graph such that the edit distance is minimal. Furthermore, we have provided an algorithm that edits a given properly colored graph into an LDT graph when restricted to deletion or insertion. The data set used for benchmarking the edit distance to an LDT graph of the heuristics presented consists of smaller graphs and as such no conclusion could be made about how well these heuristics perform on larger graphs, i.e., graphs with up to 100 or 200 vertices. It is therefore necessary that solutions are generated for larger graphs in order for future heuristics to be properly benchmarked, as it takes a lot of time to generate solutions using the ILP.

References

[1] A. Aho, Y. Sagiv, T. G. Szymanski, and J. Ullman. "Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions". In: *SIAM J. Comput.* 10 (1981), pp. 405–421.

[2] Jennifer Becq, Cécile Churlaud, and Patrick Deschavanne. "A Benchmark of Parametric Methods for Horizontal Transfers Detection". In: *PLOS ONE* 5.4 (Apr. 2010), pp. 1–9. DOI: 10.1371/journal.pone.0009989. URL: https://doi.org/10.1371/journal.pone.0009989.

[3] Juan Bello-López, Omar Cabrero-Martínez, Gabriela Cervantes, Cecilia Hernández-Cortez, Leda Pelcastre-Rodríguez, Luis Gonzalez-Avila, and Graciela Castro-Escarpulli. "Horizontal Gene Transfer and Its Association with Antibiotic Resistance in the Genus Aeromonas spp". In: *Microorganisms* 7 (Sept. 2019), p. 363. DOI: 10.3390/microorganisms7090363.

[4] Jaroslaw Byrka, Sylvain Guillemot, and Jesper Jansson. "New results on optimizing rooted triplets consistency". In: *Discrete Applied Mathematics* 158.11 (2010), pp. 1136–1147. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2010.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0166218X10000983.

[5] Christophe Crespelle. *Linear-Time Minimal Cograph Editing*. 2019. URL: https://perso.ens-lyon.fr/christophe.crespelle/publications/SUB_minimal-cograph-editing.pdf.

[6] Ralph P Grimaldi. *Discrete and combinatorial mathematics*. Fifth. Pearson Addison Wesley, 2004. Chap. 11.

[7] Marc Hellmuth, Nicolas Wieseke, Marcus Lechner, Hans-Peter Lenhof, Martin Middendorf, and Peter Stadler. "Phylogenomics with Paralogs". In: *Proceedings of the National Academy of Sciences* (Feb. 2015). DOI: 10.1073/pnas.1412770112.

[8] Matt Ravenhall, Nives Škunca, Florent Lassalle, and Christophe Dessimoz. "Inferring Horizontal Gene Transfer". In: *PLOS Computational Biology* 11.5 (May 2015), pp. 1–16. DOI: 10.1371/journal.pcbi.1004095. URL: https://doi.org/10.1371/journal.pcbi.1004095.

[9] David Schaller, Manuel Lafond, Peter F. Stadler, Nicolas Wieseke, and Marc Hellmuth. *Indirect Identification of Horizontal Gene Transfer*. 2021. arXiv: 2012.08897v2 [q-bio.PE].

[10] Charles Semple and Mike Steel. *Phylogenetics*. Oxford University Press, 2003. Chap. 6.4, pp. 118–121.

[11] Peter F. Stadler, Manuela Geiß, David Schaller, Alitzel López Sánchez, Marcos González Laffitte, Dulce I. Valdivia, Marc Hellmuth, and Maribel Hernández Rosales. "From pairs of most similar sequences to phylogenetic best matches". In: *Algorithms for Molecular Biology* 15 (2020). ISSN: 1748-7188. DOI: 10.1186/s13015-020-00165-2. URL: https://doi.org/10.1186/s13015-020-00165-2.

[12] Mechthild Stoer and Frank Wagner. "A Simple Min-Cut Algorithm". In: *J. ACM* 44.4 (July 1997), pp. 585–591. ISSN: 0004-5411. DOI: 10.1145/263867.263872. URL: https://doi.org/10.1145/263867.263872.

Datalogi
www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm