CDMTCS Research Report Series

A QUBO Formulation for the Tree Containment Problem

Michael J. Dinneen
Pankaj S. Ghodla
Simone Linz

School of Computer Science,
The University of Auckland,
Auckland, New Zealand

CDMTCS-560
February 2022

Centre for Discrete Mathematics and
Theoretical Computer Science

A QUBO Formulation for the Tree Containment Problem

Michael J. Dinneen[a], Pankaj S. Ghodla[a] and Simone Linz[a,*]

[a] School of Computer Science, University of Auckland, Auckland, New Zealand

ARTICLE INFO

Keywords:
Quantum computing
QUBOs
Phylogenetic trees and networks
Tree Containment

ABSTRACT

Phylogenetic (evolutionary) trees and networks are leaf-labeled graphs that are widely used to represent the evolutionary relationships between entities such as species, languages, cancer cells, and viruses. To reconstruct and analyze phylogenetic networks, the problem of deciding whether or not a given rooted phylogenetic network embeds a given rooted phylogenetic tree is of recurring interest. This problem, formally know as Tree Containment, is NP-complete in general and polynomial-time solvable for certain classes of phylogenetic networks. In this paper, we connect ideas from quantum computing and phylogenetics to present an efficient Quadratic Unconstrained Binary Optimization formulation for Tree Containment in the general setting. For an instance $(\mathcal{N}, \mathcal{T})$ of Tree Containment, where $\mathcal{N}$ is a phylogenetic network with $n_{\mathcal{N}}$ vertices and $\mathcal{T}$ is a phylogenetic tree with $n_{\mathcal{T}}$ vertices, the number of logical qubits that are required for our formulation is $O(n_{\mathcal{N}} n_{\mathcal{T}})$.

1. Introduction

Phylogenetics is the study of evolutionary histories and relationships between different, often biological, entities such as species, genes, viruses, or languages that are generically referred to as taxa. Traditionally, rooted leaf-labeled trees, which are known as phylogenetic trees, have been widely used to represent and analyze evolutionary relationships that are dominated by tree-like processes like speciation [16]. A phylogenetic tree is reconstructed from biological sequence data (e.g. DNA or protein sequences) under various optimization criteria or evolutionary models so that the resulting tree $\mathcal{T}$ 'in some sense' best explains a given dataset. Each leaf of $\mathcal{T}$ is labeled by a taxon whereas all inner vertices of $\mathcal{T}$ are unlabeled. The latter can be thought of as hypothetical ancestors, including extinct species, for which no data is available. Although phylogenetic trees are an extensively used in evolutionary biology and researchers are now able to reconstruct such trees for thousands of taxa [24], phylogenetic trees cannot represent complex evolutionary relationships that are the result of non-treelike processes such as hybridization or horizontal gene transfer, which are common in many groups of organisms [31, 34, 36]. For example, it has been observed that horizontal gene transfer contributes to about 10%–20% of all genes in prokaryotes [26]. Non-treelike processes, collectively referred to as reticulation, result in species whose DNA is a mosaic of DNA derived from different ancestors. To accurately describe complex evolutionary histories that include reticulation, rooted leaf-labeled graphs, called phylogenetic networks [4, 5, 23], are now widely acknowledged to complement phylogenetic trees.

A particularly well-studied problem, which arises in the analysis of rooted phylogenetic networks through the lens of rooted phylogenetic trees, is the following embedding problem. Given a rooted phylogenetic tree $\mathcal{T}$ and a rooted phylogenetic network $\mathcal{N}$ that have both been reconstructed for the same set of taxa, does $\mathcal{N}$ embed $\mathcal{T}$? This decision problem, which we will make more precise in the next section, is called Tree Containment. Without imposing any structural constraints on $\mathcal{N}$, Tree Containment is NP-complete [25]. However, it has also been shown that Tree Containment is polynomial-time solvable, for example, for so-called tree-child or level k-networks [37]. Afterwards, Gunawan et al. [21], and Bordewich and Semple [6] independently showed that Tree Containment is solvable in cubic time for reticulation-visible networks, a superclass of tree-child networks. Since then, various algorithms have been developed to solve Tree Containment, with the fastest such algorithms having a running time that is linear in the number of taxa. For example, see a recent linear-time algorithm to solve Tree Containment for reticulation-visible networks [38] and references therein. Although these substantial improvements have turned Tree Containment into one of the most well-studied problems in mathematical research on phylogenetic networks, much less is known about how to 'efficiently' solve Tree Containment for arbitrary rooted phylogenetic networks.

*Corresponding author
mjd@cs.auckland.ac.nz (M.J. Dinneen); pgho580@aucklanduni.ac.nz (P.S. Ghodla); s.linz@auckland.ac.nz (S. Linz)
ORCID(s): 0000-0001-9977-525X (M.J. Dinneen); 0000-0003-0862-9594 (S. Linz)

In this paper, we take a quantum computing approach to Tree Containment. Quantum computers are known to be able to solve certain problems in significantly lower time complexity than corresponding current-best classical algorithms. Although it is still debatable whether or not quantum computers can solve NP-hard problems in polynomial time, they have provided efficient solutions for several instances of NP-hard problems [7, 8, 27, 28, 30].

In a talk about simulating the quantum mechanical process, Feynman argued about simulating physics using a quantum computer [17]. This talk sparked interest in building a quantum computer. Soon after Feyman's talk, Deutsch [12] developed the universal quantum model of computation called the *Quantum Gate Model* [30]. The central goal behind this new model of computation was to exploit the properties of quantum mechanics to get a *quantum-speedup* over the classical model of computation. Research conducted so far on this topic shows much promise with the two primary outstanding examples being Grover's and Shor's algorithms that we briefly describe next. Grover's algorithm [20] searches through an unsorted database of size N, of which only one record satisfies a particular property, in $O(\sqrt{N})$ steps. In contrast, any classical algorithm to solve this database problem certainly takes $O(N)$ steps as it needs to iterate through a significant fraction (on average $N/2$) of all records. Shor's algorithm [35] factorizes integers. Its computational complexity is polynomial in the number of digits of the integer to be factorized. On the other hand, no classical algorithm is known that factorizes integers in polynomial time.

Adiabatic Quantum Computing (AQC) is an alternative to the quantum gate model that was first described in [14]. Subsequently, Aharonov et al. [2] have developed an adiabatic simulation for any given quantum algorithm, which implies that AQC and the quantum gate model are polynomially equivalent. AQC is based on the evolution of a time-dependent Hamiltonian that transitions from an initial Hamiltonian to a final Hamiltonian. The initial Hamiltonian's ground state is easy to construct, and the final Hamiltonian's ground state encodes the solution to a given problem. For a detailed explanation of how this evolution between initial and final Hamiltonian occurs, we refer the interested reader to [14]. The primary advantage of AQC over the quantum gate model is that a relatively easy to build *quantum annealer* [30], which is based on AQC, can be used to identify the minimum of an objective function.

D-Wave Systems Inc. is a Canadian quantum computing company that has developed the following quantum annealers for AQC.

Annealer	Number of qubits	Number of couplers
D-Wave One (2011)	128	352
D-Wave Two (2012)	512	1,472
D-Wave 2X (2015)	1000	3,360
D-Wave 2000Q (2017)	2048	6,016
D-Wave Advantage (2019)	5640	40,484

In the table above, the number of qubits refers to the number of physical qubits available and the number of couplers refers to the number of connections between the physical qubits in a quantum annealer. Currently, both D-Wave 2000Q and D-Wave Advantage can be accessed through the D-Wave's website[1] and can solve problems for which an equivalent *Ising* or *Quadratic Unconstrained Binary Optimization (QUBO)* formulation exist.

The main contribution of this paper is a QUBO formulation of Tree Containment for when the input is not restricted to a particular class of rooted phylogenetic networks. To the best of our knowledge, this is the first time that unconventional computing is used to approach a problem from phylogenetics. For two recent surveys on potential future applications of quantum computing in computational biology, we refer the reader to [15, 32].

The remainder of the paper is organized as follows. Section 2 contains the basic definitions from phylogenetics, followed by a brief introduction to QUBO and the general methodology in Section 3. Then, in Section 4, we present a reduction of Tree Containment to QUBO. We finish with an example and some experimental results in Section 5, and a short conclusion in Section 6.

2. Preliminaries from Phylogenetics

This section introduces notation and terminology from phylogenetics. Throughout the paper, X denotes a non-empty finite set. Furthermore, all logarithms are base 2, and we write $\lg(x)$ to refer to $\log_2(x)$.

Let G be a directed graph, with vertex set $V(G)$ and edge set $E(G)$. Let u and v be two vertices of G. We say that u is a *parent* of v and that v is a *child* of u if $(u, v) \in E(G)$. A *directed path* of G is a sequence $(v_1, v_2, \ldots, v_k)$

[1] Website: `https://www.dwavesys.com/`

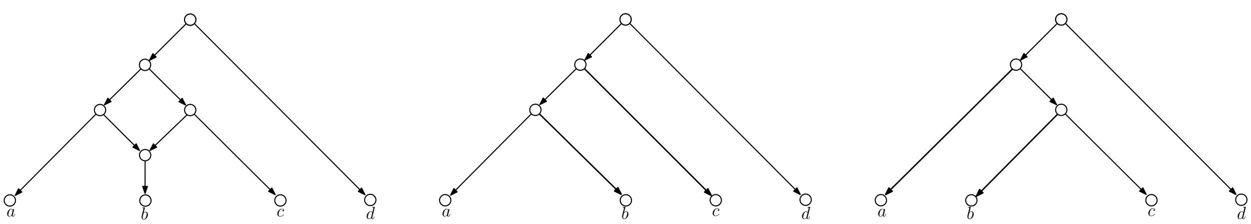
of distinct vertices in $V(G)$ such that $(v_i, v_{i+1}) \in E(G)$ for each $i \in \{1, 2, \ldots, k-1\}$. Similarly, a *path* of G is a sequence $(v_1, v_2, \ldots, v_k)$ of distinct vertices in $V(G)$ such that (v_i, v_{i+1}) or (v_{i+1}, v_i) is an element in $E(G)$ for each $i \in \{1, 2, \ldots, k-1\}$. We say that G is *weakly connected* (or short, *connected*) if there is a path between any two vertices in G. A vertex $u \in V(G)$ is called a *root* if u has in-degree 0 and there exists a directed path from u to v for all $v \in V(G)\backslash\{u\}$. Furthermore, $x \in V(G)$ is called a *terminal vertex* if it has out-degree 0. Lastly, with (v_1, v_2, v_3) being a directed path in G such that v_2 has in-degree 1 and out-degree 1, the operation of *suppressing* v_2 in G results in a new directed graph with vertex set $V(G)\backslash\{v_2\}$ and edge set $(E(G)\backslash\{(v_1, v_2), (v_2, v_3)\}) \cup \{(v_1, v_3)\}$.

We now turn to a particular class of directed graphs that will play an important role in what follows. A *rooted binary phylogenetic network* $\mathcal{N}$ on X is a rooted acyclic directed graph with no two edges in parallel that satisfies the following three properties.

1. The (unique) root has in-degree 0 and out-degree 2.
2. A vertex with out-degree 0 has in-degree 1, and the set of vertices with out-degree 0 is X.
3. All remaining vertices have either in-degree 1 and out-degree 2, or in-degree 2 and out-degree 1.

We call X the *leaf set* of $\mathcal{N}$. For technical reasons, if $|X| = 1$, then we allow $\mathcal{N}$ to consist of the single vertex in X. Let $\mathcal{N}$ be a rooted binary phylogenetic network. A vertex of $\mathcal{N}$ is called a *tree vertex* if it has out-degree 2. Similarly, a vertex of $\mathcal{N}$ is called a *reticulation vertex* if it has in-degree 2 and out-degree 1. To illustrate, see Figure 1(a) for an example of a rooted binary phylogenetic network with one reticulation vertex, four tree vertices (one is also root), and four leaves. We call $\mathcal{N}$ a *rooted binary phylogenetic X-tree* if $\mathcal{N}$ is a rooted binary phylogenetic network with no reticulation vertex. To ease reading, we will refer to a rooted binary phylogenetic network and a rooted binary phylogenetic tree as a *phylogenetic network* and a *phylogenetic tree*, respectively, since all such networks and trees in this paper are rooted and binary.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two phylogenetic networks on X with vertex and edge sets V_1 and E_1, and V_2 and E_2, respectively. We say that $\mathcal{N}_1$ is *isomorphic* to $\mathcal{N}_2$ if there is a bijection $\varphi : V_1 \to V_2$ such that $\varphi(x) = x$ for all $x \in X$, and $(u, v) \in E_1$ if and only if $(\varphi(u), \varphi(v)) \in E_2$ for all $u, v \in V_1$.



(a) $\mathcal{N}$ (b) $\mathcal{T}_1$ (c) $\mathcal{T}_2$

Figure 1: (a) A rooted binary phylogenetic network $\mathcal{N}$ on $X = \{a, b, c, d\}$ with a single reticulation vertex. (b) and (c) The two rooted binary phylogenetic X-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ displayed by $\mathcal{N}$.

Now, let $\mathcal{N}$ be a phylogenetic network on X, and let $\mathcal{T}$ be a phylogenetic X-tree. We say $\mathcal{N}$ *displays* $\mathcal{T}$ if $\mathcal{T}$ can be obtained from $\mathcal{N}$ by deleting vertices and edges, and by suppressing any resulting vertices of in-degree 1 and out-degree 1. Referring back to Figure 1, the phylogenetic network that is shown in (a) displays the two phylogenetic trees that are shown in (b) and (c).

We are now in a position to formally state the Tree Containment decision problem.

Tree Containment $(\mathcal{N}, \mathcal{T})$
Input: A phylogenetic X-tree $\mathcal{T}$ and a phylogenetic network $\mathcal{N}$ on X.
Output: Does $\mathcal{N}$ display $\mathcal{T}$?

3. Quadratic Unconstrained Binary Optimization (QUBO)

In this section, we give a brief introduction to QUBO and discuss the general methodology to solve a problem using AQC.

3.1. What is QUBO?

QUBO is an NP-hard combinatorial optimization problem [19, 33]. Instances of many classical NP-hard problems such as finding a maximum cut or a minimum vertex coloring of a graph can be reduced to equivalent instances of QUBO. More specifically, QUBO minimizes a quadratic objective function $F : \mathbb{B}^n \rightarrow \mathbb{R}$. Using matrix notation, the quadratic objective function has the form $H(\mathbf{x}) = \mathbf{x}^T Q \mathbf{x}$, where $\mathbf{x}^T = [x_1, x_2, \dots, x_n]$ is a row vector of n binary variables and Q is and upper triangular $n \times n$ matrix. Then the QUBO problem is that of solving the following equation

$$x^* = \min_{\mathbf{x}} \sum_{i=1}^{n} \sum_{j=i}^{n} Q_{i,j} x_i x_j = \min_{\mathbf{x}} \mathbf{x}^T Q \mathbf{x},$$

where the minimum is taken over all binary vectors $\mathbf{x}$. We use x^* to denote the minimum of H and $\mathbf{x}^*$ to denote a binary vector that yields x^*. In the quantum annealing model of QUBO, the matrix Q represents the problem Hamiltonian and each x_i in $\mathbf{x}$ represents a logical qubit. The logical qubits are different from the physical qubits (qubits on a quantum annealer) as several physical qubits could be required to represent a single logical qubit when we embed a given QUBO (non-zero entries represent adjacency structure) onto the host graph (physical qubits as vertices and couplers as edges) of a quantum annealer. The non-zero off-diagonal entries, i.e. $Q_{i,j}$ where $i < j$, correspond to the coupler biases between x_i and x_j. Furthermore, the diagonal entries correspond to the qubit biases, which refer to the external magnetic fields applied on the qubits.

3.2. Methodology

Suppose that we want to solve a problem P using the AQC model. First we need to establish a polynomial-time reduction that reduces a given instance of P to an instance Q(P) of QUBO form. Second, we need to ensure that the $n \times n$ matrix in Q(P) is as small and sparse as possible. This is because the size and density of the QUBO matrix (more precisely, the density of the graph whose weighted adjacency matrix is the QUBO matrix) have a significant impact on the probability of the system being in the (minimum) ground state in the final Hamiltonian. We want to have a high enough probability for the system to be in the ground state in the final Hamiltonian so that we can efficiently query D-Wave's quantum annealers to solve Q(P). Third, viewing the QUBO matrix in Q(P) as a weighted adjacency matrix of a graph G, we (minor) embed this graph G onto the host graph of a D-Wave's quantum annealer, i.e., either the Chimera graph (D-Wave 2000Q) or the Pegasus graph (D-Wave Advantage). A *minor embedding* of a graph G onto a graph H is a function $\phi : V(G) \rightarrow 2^{V(H)}$ that satisfies the following properties:

1. The set of vertices $\phi(u)$ and $\phi(v)$ are disjoint for all $u, v \in V(G)$, where $u \neq v$.
2. For each $u \in V(G)$, there exists a subset $E' \subset E(H)$ such that the subgraph $H' = (\phi(u), E')$ of H is connected.
3. For each $\{u, v\} \in E(G)$, there exit vertices $u', v' \in V(H)$ such that $u' \in \phi(u)$, $v' \in \phi(v)$, and $\{u', v'\} \in E(H)$.

If G is bigger than the host graph or if G cannot be embedded onto the host graph because it is too dense, then the package *qbsolv* that is provided by D-Wave can be used to break the $n \times n$ matrix in Q(P) into sub-matrices and solve them separately before combining the results to get a solution for the initial problem. This package uses techniques from well-known paradigms such as divide-and-conquer and dynamic programming; for more information, see [11]. In the last step, a quantum annealer is queried to compute x^* and $\mathbf{x}^*$.

A problem one might immediately see is that finding a minor embedding of a graph G onto a host graph is an NP-hard problem in itself. Furthermore, if we can find an embedding there are often better ones (e.g. those that minimize the maximum or average cardinality of $\phi(u)$); here the chance of the quantum annealer successfully solving Q(P) increases with better embeddings due to hardware limitations. The extended optimization problem of finding an embedding with maximum mapping size at most k (i.e. $|\phi(u)| \leq k$ for each $u \in V(G)$)) is also NP-hard. Note that, if we fix $k = 1$, then we solve the NP-hard subgraph isomorphism problem and if we fix $k = |V(H)|$, then we solve the original minor-embedding problem. However, as it is not necessary to find an optimal embedding with respect to some criteria, we can use a probabilistic algorithm (on a classical computer) with a polynomial-time overhead to find a 'good' minor embedding of G.

After getting the results from a quantum annealer, we need a way to decode the input values $\mathbf{x}^*$ that yield x^* for an instance of a problem P. In practice, the final solution is probabilistic. Since we might not get the optimal answer the first time that we query a quantum annealer, we may need to query it several times (often more than 100 times, in practice) to increase the probability of getting the optimal answer. If P is a decision problem in NP, as it is in the case of Tree Containment, then it is often possible to reduce P to Q(P) such that the optimal value after post-processing is 0 if the answer to P is ‘yes’. Post-processing is the process of adding an offset to the optimal value x^*. Using this approach, we know when we get the optimal solution from a quantum annealer. We use this approach in our reduction from Tree Containment to QUBO in Section 4.

3.3. Reducing Higher-Order Functions into QUBO

Some problems naturally reduce to a binary cubic, or binary higher-order function. Although such a function does not immediately fit into the QUBO framework, it can be recast as a binary quadratic function and then be solved with D-Wave’s quantum annealers. We introduce additional binary variables during the recast and replace the higher-order terms with additional penalty functions that have binary quadratic order terms. See [7] for a detailed example of converting a traditional Integer Linear Programming formulation for a constrained optimization problem to QUBO form, which uses auxiliary variables for reducing higher-order functions and models non-binary variables as sets of binary variables.

The following lemma and proposition reduces a binary cubic function to a binary quadratic function. The lemma has been taken from [19], where no proof is given. For reasons of completeness, we next establish a formal proof.

Lemma 1. *Let x_1, x_2, and y be three binary variables. Furthermore, let $P = x_1x_2 - 2x_1y - 2x_2y + 3y$. Then $P = 0$ if and only if $y = x_1x_2$.*

Proof. ($\Longrightarrow$) Suppose that $P = 0$. We consider four cases.

1. If $x_1 = 1, x_2 = 1$ and $P = 0$, then $0 = 1 - 2y - 2y + 3y$ and hence $y = 1$.
2. If $x_1 = 1, x_2 = 0$ and $P = 0$, then $0 = 0 - 2y + 3y$ and hence $y = 0$.
3. If $x_1 = 0, x_2 = 1$ and $P = 0$, then $0 = 0 - 2y + 3y$ and hence $y = 0$.
4. If $x_1 = 0, x_2 = 0$ and $P = 0$, then $0 = 0 + 3y$ and hence $y = 0$.

From each case, it follows that $y = x_1x_2$.

($\Longleftarrow$) Now suppose that $y = x_1x_2$. Then, since $x^2 = x$ for any binary variable, we have

$$P = x_1x_2 - 2x_1x_1x_2 - 2x_2x_1x_2 + 3x_1x_2 = x_1x_2 - 2x_1x_2 - 2x_1x_2 + 3x_1x_2 = 0.$$

The lemma follows. □

Proposition 2. *In a QUBO framework, a binary cubic term can be converted into an equivalent binary quadratic term.*

Proof. Let $x_1x_2x_3$ be a binary cubic term. Furthermore, let y be a new binary variable, and let

$$P = x_1x_2 - 2x_1y - 2x_2y + 3y$$

be a binary quadratic penalty term. It follows by Lemma 1 that $P = 0$ if and only if $y = x_1x_2$. Hence, substituting $x_1x_2x_3$ with $x_3y + P$ replaces a binary cubic term with four binary quadratic terms and one binary linear term. Specifically, if we minimize P when we minimize the resulting quadratic function, then y represent x_1x_2. Applying this substitution technique repeatedly to each binary cubic term of a binary cubic function results in a binary quadratic function. □

Note that we can recursively apply Proposition 2 to reduce any binary higher-order function to a binary quadratic function.

4. Reducing Tree Containment to QUBO

In this section, we present a reduction from Tree Containment to QUBO. For an instance of Tree Containment that consists of a phylogenetic network $\mathcal{N}$ on X and a phylogenetic X-tree $\mathcal{T}$, the QUBO formulation requires $O(n_{\mathcal{T}}n_{\mathcal{N}})$ logical qubits, where $n_{\mathcal{N}}$ is the number of vertices in $\mathcal{N}$ and $n_{\mathcal{T}}$ is the number of vertices in $\mathcal{T}$.

4.1. QUBO Formulation

Throughout this section, let $\mathcal{N}$ be a phylogenetic network on X, and let $\mathcal{T}$ be a phylogenetic X-tree. Let $E(\mathcal{N})$ be the set of edges and $V(\mathcal{N}) = \{v_0, v_1, v_2, \dots, v_{n_\mathcal{N}-1}\}$ be the set of vertices of $\mathcal{N}$, and let $E(\mathcal{T})$ be the set of edges and $V(\mathcal{T}) = \{u_0, u_1, u_2, \dots, u_{n_\mathcal{T}-1}\}$ be the set of vertices of $\mathcal{T}$. Without loss of generality, we may assume that u_0 is the root of $\mathcal{T}$. Additionally, let $u_{n_\mathcal{T}}$ be a vertex that is not an element in $V(\mathcal{T})$.

Intuitively, if $\mathcal{N}$ displays $\mathcal{T}$, then there exists a mapping that maps each vertex of $\mathcal{T}$ to a vertex of $\mathcal{N}$ and each edge of $\mathcal{T}$ to a directed path of $\mathcal{N}$. The following QUBO formulation for Tree Containment establishes a mapping (detailed below) that maps each vertex of $V(\mathcal{T})$ to at least one vertex of $\mathcal{N}$. In this mapping, $u_{n_\mathcal{T}}$ is mapped to each vertex of $\mathcal{N}$ that is not in the image of any vertex in $V(\mathcal{T})$.

Now, let

$$m = n_\mathcal{N}(n_\mathcal{T}+1) + (n_\mathcal{T}-1)\left(1 + \lfloor \lg(n_\mathcal{N} - n_\mathcal{T}) \rfloor\right) + n_\mathcal{T}(\alpha+\beta) + 2\beta\gamma,$$

and let $\mathbf{x} \in \mathbb{B}^m$ be a vector of binary variables, where $n_\mathcal{N} = |V(\mathcal{N})|$, $n_\mathcal{T} = |V(\mathcal{T})|$, $\gamma = n_\mathcal{T} - |X|$, and α (resp. β) equals the number of reticulation vertices (resp. tree vertices) of $\mathcal{N}$. It immediately follows that $\mathbf{x}$ contains $O(n_\mathcal{N} n_\mathcal{T})$ binary variables.

We next describe the binary variables that are represented by $\mathbf{x}$ and their encoding. More precisely, for $0 \le i \le n_\mathcal{T}$ and $0 \le j < n_\mathcal{N}$, $x_{i,j} = 1$ encodes that $u_i \in V(\mathcal{T}) \cup \{u_{n_\mathcal{T}}\}$ is mapped to $v_j \in V(\mathcal{N})$ and, similarly, $x_{i,j} = 0$ encodes that $u_i \in V(\mathcal{T}) \cup \{u_{n_\mathcal{T}}\}$ is not mapped to $v_j \in V(\mathcal{N})$. Additionally, we introduce three types of slack variables.

1. For each vertex $u_i \in V(\mathcal{T}) \backslash \{u_0\}$, we have $1 + \lfloor \lg(n_\mathcal{N} - n_\mathcal{T}) \rfloor$ slack variables that are denoted by $y_{i,r}$ for $0 \le r \le \lfloor \lg(n_\mathcal{N} - n_\mathcal{T}) \rfloor$.
2. For each $u_i \in V(\mathcal{T})$, we have $\alpha + \beta$ slack variable that are denoted by $z_{i,j}$ for each index j such that $v_j \in V(\mathcal{N})$ is either a tree or reticulation vertex.
3. For each $u_i \in V(\mathcal{T})$ that is not a leaf, we have 2β slack variables that are denoted by $\hat{z}_{i,2j}$ and $\hat{z}_{i,2j+1}$ for each index j such that $v_j \in V(\mathcal{N})$ is a tree vertex.

For the following Hamiltonian, we assume without loss of generality that $|V(\mathcal{N})| \ge |V(\mathcal{T})|$. Indeed, if this is not the case, then $\mathcal{N}$ does not display $\mathcal{T}$. Let v_j be a tree vertex of $\mathcal{N}$, and let v_{j_1} and v_{j_2} be the two children of v_j, where j_1 and j_2 are the indices of the children of v_j. We use $c_1(v_j)$ and $c_2(v_j)$ to denote v_{j_1} and v_{j_2}, respectively. Now, let v_j be a reticulation vertex of $\mathcal{N}$. Similarly to the children of a tree vertex, let v_{j^1} and v_{j^2} be the two parents of v_j, where j^1 and j^2 are the indices of the two parents of v_j. Again, we use $p_1(v_j)$ and $p_2(v_j)$ to denote v_{j^1} and v_{j^2}, respectively. Furthermore, we define two functions f and g as follows.

$$f(u_i, u_l) = \begin{cases} 1, \text{if there exists an edge from } u_i \text{ to } u_l \text{ in } \mathcal{T} \\ 0, \text{otherwise} \end{cases}$$

$$g(v_j, v_k) = \begin{cases} 1, \text{if there exists an edge from } v_j \text{ to } v_k \text{ in } \mathcal{N} \\ 0, \text{otherwise} \end{cases}$$

We are now in a position to define the Hamiltonian $H(\mathbf{x})$, also sometimes called the objective function, as follows, where $A, B \in \mathbb{R}^+$. The choice of A and B is detailed in Section 4.2. However, we already note here that B is sufficiently larger than A. The coefficients of the terms $x_i x_j$ of the following binary objective function $H(\mathbf{x})$ correspond to the entries in the QUBO matrix Q, as defined in Section 3.1.

$$H(\mathbf{x}) = B \cdot \left(\sum_{I=1}^{10} P_I(\mathbf{x}) \right) + A \cdot P_{11}(\mathbf{x}) + P_{12}(\mathbf{x})$$

where

$$P_1(\mathbf{x}) = \left(1 - \sum_{j=0}^{n_\mathcal{N}-1} x_{0,j}\right)^2 + \sum_{i=1}^{n_\mathcal{T}-1} \left(1 - \sum_{j=0}^{n_\mathcal{N}-1} x_{i,j} + \sum_{r=0}^{\lfloor \lg(n_\mathcal{N} - n_\mathcal{T}) \rfloor} 2^r y_{i,r}\right)^2$$

$$P_2(\mathbf{x}) = \sum_{j=0}^{n_{\mathcal{N}}-1} \left(\sum_{i=0}^{n_{\mathcal{T}}} x_{i,j} - 1 \right)^2$$

$$P_3(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} \left(x_{i,j_1} x_{i,j_2} - 2x_{i,j_1} z_{i,j} - 2x_{i,j_2} z_{i,j} + 3z_{i,j} \right)$$

$$P_4(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} x_{i,j} z_{i,j}$$

$$P_5(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{v_j \text{ is a reticulation} \\ \text{vertex of } \mathcal{N}}} \left(x_{i,j^1} x_{i,j^2} - 2x_{i,j^1} z_{i,j} - 2x_{i,j^2} z_{i,j} + 3z_{i,j} \right)$$

$$P_6(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{v_j \text{ is a reticulation} \\ \text{vertex of } \mathcal{N}}} x_{i,j} z_{i,j}$$

$$P_7(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l \neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(\sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} x_{i,j} z_{l,j} \right) \right)$$

$$P_8(\mathbf{x}) = \sum_{\substack{u_i \text{ is not a} \\ \text{leaf of } \mathcal{T}}} \sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} \Big(x_{i,j} x_{i,j_1} - 2x_{i,j} \hat{z}_{i,2j} - 2x_{i,j_1} \hat{z}_{i,2j} + 3\hat{z}_{i,2j}$$
$$+ x_{i,j} x_{i,j_2} - 2x_{i,j} \hat{z}_{i,2j+1} - 2x_{i,j_2} \hat{z}_{i,2j+1} + 3\hat{z}_{i,2j+1} \Big)$$

$$P_9(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l \neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(\sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} \left(\hat{z}_{i,2j} x_{l,j_2} + \hat{z}_{i,2j+1} x_{l,j_1} \right) \right) \right)$$

$$P_{10}(\mathbf{x}) = \sum_{\substack{u_i \text{ is a} \\ \text{leaf of } \mathcal{T}}} \left(1 - x_{i,j} \right)^2 \text{, where } j \text{ is the index of } u_i \text{ in } \mathcal{N}$$

$$P_{11}(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{j=0}^{n_{\mathcal{N}}-1} \left(x_{i,j} \left(1 - \sum_{\substack{k=0 \\ k \neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,k} \right) \right) - n_{\mathcal{T}}$$

$$P_{12}(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l \neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(1 - \sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k \neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} \right) \right)$$

For notational convenience, we use a map d that maps a tuple $(\mathbf{x}, u_i)$ to a subset of $V(\mathcal{N})$. More precisely, we define

$$d : (\mathbb{B}^m, V(\mathcal{T}) \cup \{u_{n_{\mathcal{T}}}\}) \to 2^{V(\mathcal{N})}$$

to decode the subset of vertices of $\mathcal{N}$ such that $d(\mathbf{x}, u_i) = \{v_j \in V(\mathcal{N}) \mid x_{i,j} = 1\}$. We interpret this as the vertex u_i of $\mathcal{T}$ being mapped to the the subset $\{v_j \in V(\mathcal{N}) \mid x_{i,j} = 1\}$ of $V(\mathcal{N})$. We also sometimes interpret $d(\mathbf{x}, u_i)$ as an induced subgraph of $\mathcal{N}$ whose vertex set is $d(\mathbf{x}, u_i)$ and whose edge set is $\{(v_c, v_d) \in E(\mathcal{N}) \mid v_c, v_d \in d(\mathbf{x}, u_i)\}$.

We now return to the Hamiltonian $H(\mathbf{x})$ as defined above and establish a lemma for each of $P_1(\mathbf{x}), P_2(\mathbf{x}), \ldots, P_{12}(\mathbf{x})$. These lemmas provide some insight into different parts of $H(\mathbf{x})$. Lemmas 5, 7, and 10 follow from the proof of Lemma 1, whereas the proofs of Lemmas 3, 4, and 12 are straightforward and omitted.

The first penalty function P_1 ensures that the root of $\mathcal{T}$ is mapped to exactly one vertex of $\mathcal{N}$ and each other vertex of $\mathcal{T}$ is mapped to at least one vertex of $\mathcal{N}$.

Lemma 3. $P_1(\mathbf{x}) = 0$ *if and only if, for each vertex* $u_i \in V(\mathcal{T})$*, we have* $|d(\mathbf{x}, u_i)| > 0$ *and* $|d(\mathbf{x}, u_0)| = 1$.

Next we ensure that at most one vertex of $\mathcal{T}$ is mapped to each vertex of $\mathcal{N}$.

Lemma 4. $P_2(\mathbf{x}) = 0$ *if and only if, for each vertex* $v_j \in V(\mathcal{N})$*, there exists exactly one vertex* $u_i \in V(\mathcal{T}) \cup \{u_{n_\mathcal{T}}\}$ *such that* $v_j \in d(\mathbf{x}, u_i)$.

The penalty function P_3 establishes equivalence between slack variables and the product of two non-slack variables.

Lemma 5. $P_3(\mathbf{x}) = 0$ *if and only if* $z_{i,j} = x_{i,j_1} x_{i,j_2}$*, where* $v_j \in V(\mathcal{N})$ *is a tree vertex and* $u_i \in V(\mathcal{T})$.

We now require that no vertex of $\mathcal{T}$ maps to a tree vertex in $\mathcal{N}$ and its two children.

Lemma 6. *Suppose that* $P_3(\mathbf{x}) = 0$*. Then* $P_4(\mathbf{x}) = 0$ *if and only if there does not exist a tree vertex* $v_j \in V(\mathcal{N})$ *and a vertex* $u_i \in V(\mathcal{T})$ *such that* $\{v_j, c_1(v_j), c_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$.

Proof. As $P_3(\mathbf{x}) = 0$, it follows from Lemma 5 that $z_{i,j} = x_{i,j_1} x_{i,j_2}$. Thus, $P_4(\mathbf{x}) = 0$ if and only if

$$\sum_{i=0}^{n_\mathcal{T}-1} \sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} x_{i,j} x_{i,j_1} x_{i,j_2} = 0 \tag{1}$$

It now follows that Equation (1) holds if and only if there exists no tree vertex $v_j \in V(\mathcal{N})$ such that $\{v_j, c_1(v_j), c_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$ for some $u_i \in V(\mathcal{T})$. □

The penalty function P_5 is similar to P_3.

Lemma 7. $P_5(\mathbf{x}) = 0$ *if and only if* $z_{i,j} = x_{i,j^1} x_{i,j^2}$*, where* $v_j \in V(\mathcal{N})$ *is a reticulation vertex and* $u_i \in V(\mathcal{T})$.

We next require that no vertex of $\mathcal{T}$ maps to a reticulation vertex of $\mathcal{N}$ and its two parents.

Lemma 8. *Suppose that* $P_5(\mathbf{x}) = 0$*. Then* $P_6(\mathbf{x}) = 0$ *if and only if there does not exist a reticulation vertex* $v_j \in V(\mathcal{N})$ *and a vertex* $u_i \in V(\mathcal{T})$ *such that* $\{v_j, p_1(v_j), p_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$.

Proof. The proof is analogous to that of Lemma 6. As $P_5(\mathbf{x}) = 0$, it follows from Lemma 7 that $z_{i,j} = x_{i,j^1} x_{i,j^2}$. Thus, $P_6(\mathbf{x}) = 0$ if and only if

$$\sum_{i=0}^{n_\mathcal{T}-1} \sum_{\substack{v_j \text{ is a reticulation} \\ \text{vertex of } \mathcal{N}}} x_{i,j} x_{i,j^1} x_{i,j^2} = 0 \tag{2}$$

It now follows that Equation (2) holds if and only if there exists no reticulation vertex $v_j \in V(\mathcal{N})$ such that $\{v_j, p_1(v_j), p_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$ for some $u_i \in V(\mathcal{T})$. □

Furthermore, we ensure that there exists no edge (u_i, u_l) in $\mathcal{T}$ such that u_i maps to a tree vertex v_j of $\mathcal{N}$ and u_l maps to both children of v_j.

Lemma 9. *Suppose that* $P_3(\mathbf{x}) = 0$*. Then* $P_7(\mathbf{x}) = 0$ *if and only if there does not exist a tree vertex* $v_j \in V(\mathcal{N})$ *and an edge* $(u_i, u_l) \in E(\mathcal{T})$ *such that* $v_j \in d(\mathbf{x}, u_i)$ *and* $\{c_1(v_j), c_2(v_j)\} \subseteq d(\mathbf{x}, u_l)$.

Proof. As $P_3(\mathbf{x}) = 0$ it again follows from Lemma 5 that $z_{l,j} = x_{l,j_1} x_{l,j_2}$, where $l = i$. Hence, $P_7(\mathbf{x}) = 0$ if and only if

$$\sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l \neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(\sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} x_{i,j} x_{l,j_1} x_{l,j_2} \right) \right) = 0 \tag{3}$$

The lemma now follows from Equation 3. □

The next penalty function P_8 introduces additional slack variables to avoid binary cubic terms in P_9.

Lemma 10. *$P_8(\mathbf{x}) = 0$ if and only if $\hat{z}_{i,2j} = x_{i,j} x_{i,j_1}$ and $\hat{z}_{i,2j+1} = x_{i,j} x_{i,j_2}$, where $u_i \in V(\mathcal{T}) \backslash X$ and $v_j \in V(\mathcal{N})$ is a tree vertex.*

The next lemma shows that two vertices that are incident with a given edge in $\mathcal{T}$ are not mapped to two distinct vertices in $\mathcal{N}$ that have a common parent.

Lemma 11. *Suppose that $P_8(\mathbf{x}) = 0$. Then $P_9(\mathbf{x}) = 0$ if and only if, there does not exist a tree vertex $v_j \in V(\mathcal{N})$ and an edge $(u_i, u_l) \in E(\mathcal{T})$ such that $\{v_j, c_1(v_j)\} \subseteq d(\mathbf{x}, u_i)$ and $c_2(v_j) \in d(\mathbf{x}, u_l)$, or $\{v_j, c_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$ and $c_1(v_j) \in d(\mathbf{x}, u_l)$.*

Proof. Since $P_8(\mathbf{x}) = 0$, it follows from Lemma 10 that $P_9(\mathbf{x}) = 0$ if and only if

$$\sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l \neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(\sum_{\substack{v_j \text{ is a tree} \\ \text{vertex of } \mathcal{N}}} \left(x_{i,j} x_{i,j_1} x_{l,j_2} + x_{i,j} x_{i,j_2} x_{l,j_1} \right) \right) \right) = 0 \tag{4}$$

The lemma now follows from Equation 4. □

The penalty function P_{10} ensures that the leaf labels X match in both $\mathcal{T}$ and $\mathcal{N}$.

Lemma 12. *$P_{10}(\mathbf{x}) = 0$ if and only if, for each leaf u_i of $\mathcal{T}$, there exists a leaf v_j in $\mathcal{N}$ such that u_i and v_j have the same label and $v_j \in d(\mathbf{x}, u_i)$.*

We now restrict the mapping of each vertex of $\mathcal{T}$ to induce a directed path in $\mathcal{N}$.

Lemma 13. *Suppose that $P_I(\mathbf{x}) = 0$ for each $3 \leq I \leq 6$. Then $P_{11}(\mathbf{x}) = 0$ if and only if, for each vertex $u_i \in V(\mathcal{T})$, $d(\mathbf{x}, u_i)$ is a directed path of $\mathcal{N}$.*

Proof. We first notice that $P_{11}(\mathbf{x})$ adds a penalty if and only if, for two vertices $u_i \in V(\mathcal{T})$ and $v_j \in V(\mathcal{N})$, we have $v_j \in d(\mathbf{x}, u_i)$ but no child of v_j in $\mathcal{N}$ is contained in $d(\mathbf{x}, u_i)$. Since there is no directed cycle in $\mathcal{N}$, $P_{11}(\mathbf{x})$ adds at least a penalty of 1 for each vertex in $V(\mathcal{T})$.

($\Longrightarrow$) Suppose that $P_{11}(\mathbf{x}) = 0$. Towards a contradiction, assume that there exists a vertex $u_a \in V(\mathcal{T})$ such that $d(\mathbf{x}, u_a)$ does not form a directed path in $\mathcal{N}$. This implies that there exists a vertex $v \in d(\mathbf{x}, u_a)$ such that v has two parents that are both contained in $d(\mathbf{x}, u_a)$, v has two children that are both contained in $d(\mathbf{x}, u_a)$, or $d(\mathbf{x}, u_a)$ is disconnected. Since $P_3(\mathbf{x}) = P_4(\mathbf{x}) = P_5(\mathbf{x}) = P_6(\mathbf{x}) = 0$, it follows from Lemmas 5–8, that each vertex in $d(\mathbf{x}, u_a)$ has at most one child in $\mathcal{N}$ that is contained in $d(\mathbf{x}, u_a)$ and at most one parent in $\mathcal{N}$ that is contained in $d(\mathbf{x}, u_a)$. Thus, $d(\mathbf{x}, u_a)$ is disconnected, and there exist $v_c, v_d \in d(\mathbf{x}, u_a)$ such that no vertex of $\mathcal{N}$ that is a child of v_c or v_d is contained in $d(\mathbf{x}, u_a)$. Then

$$\sum_{\substack{k=0 \\ k \neq c}}^{n_{\mathcal{N}}-1} g(v_c, v_k) x_{a,k} = 0 \text{ and } \sum_{\substack{k=0 \\ k \neq d}}^{n_{\mathcal{N}}-1} g(v_d, v_k) x_{a,k} = 0.$$

Therefore, u_a adds a penalty of at least 2 and every vertex in $V(\mathcal{T}) \backslash \{u_a\}$ adds a penalty of at least 1. As we subtract $n_{\mathcal{T}}$ in $P_{11}(\mathbf{x})$ it follows that $P_{11}(\mathbf{x}) > 0$; a contradiction. Hence, if $P_{11}(\mathbf{x}) = 0$, then $d(\mathbf{x}, u_i)$ is a directed path of $\mathcal{N}$ for each $u_i \in V(\mathcal{T})$.

($\Longleftarrow$) Let $u_i \in V(\mathcal{T})$. Suppose that $d(\mathbf{x}, u_i)$ is a directed path of $\mathcal{N}$. As $\mathcal{N}$ is acyclic, there exists exactly one vertex $v \in d(\mathbf{x}, u_i)$ such that no child of v in $\mathcal{N}$ is contained in $d(\mathbf{x}, u_i)$. This implies that u_i adds a penalty of 1. In total, we have $n_{\mathcal{T}}$ vertices in $\mathcal{T}$ and, so a total penalty of $n_{\mathcal{T}}$. Since we subtract $n_{\mathcal{T}}$ in $P_{11}(\mathbf{x})$ it follows that $P_{11}(\mathbf{x}) = 0$. The lemma now follows. □

Finally we restrict the two directed paths in $\mathcal{N}$ that correspond to two adjacent vertices of $\mathcal{T}$ to be separated by exactly one edge in $\mathcal{N}$. That is, the subgraph of $\mathcal{N}$ that is induced collectively by all vertices in $\mathcal{T}$ is a tree.

Lemma 14. *Suppose that $P_I(\mathbf{x}) = 0$ for each $3 \leq I \leq 9$ and $P_{11}(\mathbf{x}) = 0$. Then $P_{12}(\mathbf{x}) = 0$ if and only if, for each edge $(u_i, u_l) \in E(\mathcal{T})$, there exists exactly one edge $(v_c, v_d) \in E(\mathcal{N})$ such that $v_c \in d(\mathbf{x}, u_i)$ and $v_d \in d(\mathbf{x}, u_l)$.*

Proof. Let $(u_i, u_l) \in E(\mathcal{T})$. We start by noticing that $P_{12}(\mathbf{x})$ adds a penalty if and only if there does not exist an edge from a vertex in $d(\mathbf{x}, u_i)$ to a vertex in $d(\mathbf{x}, u_l)$ in $\mathcal{N}$.

($\implies$) Suppose that $P_{12}(\mathbf{x}) = 0$. Towards a contradiction, assume that there exist at least two edges $(v_c, v_d), (v_{c'}, v_{d'}) \in E(\mathcal{N})$ such that $\{v_c, v_{c'}\} \subseteq d(\mathbf{x}, u_i)$ and $\{v_d, v_{d'}\} \subseteq d(\mathbf{x}, u_l)$. By Lemma 13, $d(\mathbf{x}, u_i)$ and $d(\mathbf{x}, u_l)$ are two directed paths of $\mathcal{N}$. We consider two cases.

Case 1. Assume that $v_c \neq v_{c'}$. Then, without loss of generality, we may assume that $v_{c'}$ precedes v_c on the directed path $d(\mathbf{x}, u_i)$. Hence, $v_{c'}$ is a tree vertex of $\mathcal{N}$. Furthermore, we have $\{v_{c'}, c_a(v_{c'})\} \subseteq d(\mathbf{x}, u_i)$ and $c_b(v_{c'}) \in d(\mathbf{x}, u_l)$, where $\{a, b\} = \{1, 2\}$. This setup is shown in Figure 2. As $P_8(\mathbf{x}) = 0$, it follows by Lemma 11 that $P_9(\mathbf{x}) > 0$; a contradiction.

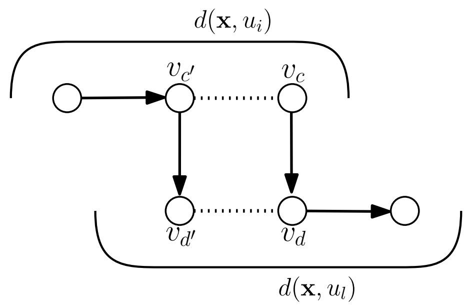


Figure 2: Setup as described in Case 1 of the proof of Lemma 14. The vertices of the top directed path and the bottom directed path represent $d(\mathbf{x}, u_i)$ and $d(\mathbf{x}, u_l)$, respectively. The tree vertex $v_{c'}$ has one child in $d(\mathbf{x}, u_i)$ and the other child in $d(\mathbf{x}, u_l)$.

Case 2. Assume that $v_c = v_{c'}$. It follows that v_c is a tree vertex of $\mathcal{N}$, $v_c \in d(\mathbf{x}, u_i)$, and $\{c_1(v_c), c_2(v_c)\} \subseteq d(\mathbf{x}, u_l)$. This setup is shown in Figure 3. Now, as $P_3(\mathbf{x}) = 0$, Lemma 9 implies that $P_7(\mathbf{x}) > 0$; again a contradiction.

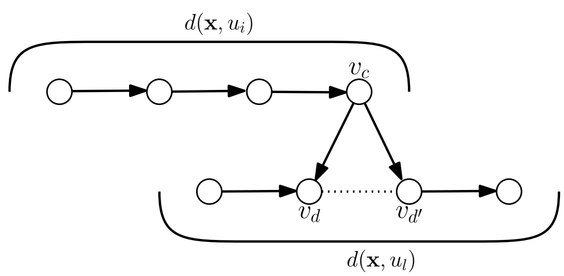


Figure 3: Setup as described in Case 2 of the proof of Lemma 14. The vertices of the top directed path and the bottom directed path represent $d(\mathbf{x}, u_i)$ and $d(\mathbf{x}, u_l)$, respectively. The tree vertex v_c has both children in $d(\mathbf{x}, u_l)$.

By combining both cases, there exists at most one edge from a vertex in $d(\mathbf{x}, u_i)$ to a vertex in $d(\mathbf{x}, u_l)$ in $\mathcal{N}$. Thus

$$\sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k \neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} \leq 1.$$

Moreover, as $P_{12}(\mathbf{x}) = 0$, we have

$$\sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k \neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} = 1.$$

Hence, for (u_i, u_l) there exists exactly one edge in $\mathcal{N}$ that has one endpoint in $d(\mathbf{x}, u_i)$ and the other endpoint in $d(\mathbf{x}, u_l)$.

($\Longleftarrow$) Suppose that, for each edge $(u_i, u_l) \in E(\mathcal{T})$, there exists exactly one edge $(v_c, v_d) \in E(\mathcal{N})$ such that $v_c \in d(\mathbf{x}, u_a)$ and $v_d \in d(\mathbf{x}, u_b)$. Then,

$$\sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k\neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} = 1$$

in $P_{12}(\mathbf{x})$ and, so, $P_{12}(\mathbf{x}) = 0$.

□

The next corollary follows from Lemmas 11 and 14.

Corollary 15. *Suppose that $P_I(\mathbf{x}) = 0$ for all $I \in \{1, 2, \dots, 9, 11, 12\}$. Let $(u_i, u_l) \in E(\mathcal{T})$, then there exists an edge from the terminal vertex of the directed path induced by $d(\mathbf{x}, u_i)$ to a vertex of the directed path induced by $d(\mathbf{x}, u_l)$ in $\mathcal{N}$.*

4.2. Proof of Correctness

In this section, we show that the Hamiltonian $H(\mathbf{x})$ as presented in Section 4.1 correctly encodes instances of Tree Containment. We start by detailing the choice of constants A and B in the definition of $H(\mathbf{x})$.

Lemma 16. *Let $\mathbf{x} \in \mathbb{B}^m$. If $P_I(\mathbf{x}) = 0$ for all $0 \leq I \leq 10$ then $P_{12}(\mathbf{x}) > -2n_{\mathcal{N}}$.*

Proof. Let u_i be a vertex of $\mathcal{T}$ that is not a leaf, and let $(u_i, u_l), (u_i, u_h) \in E(\mathcal{T})$. Consider the subgraph G_i of $\mathcal{N}$ that is induced by $d(\mathbf{x}, u_i)$. It follows from Lemmas 6 and 8 that each connected component of this subgraph is a directed path. Let c_i be the number of connected components of G_i. Then, both sums

$$\sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k\neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} \text{ and } \sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k\neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{h,k}$$

in $P_{12}(\mathbf{x})$ are each at most c_i because, by Lemmas 9 and 11, for each connected component there exists at most one edge from a vertex of this component to a vertex in $d(\mathbf{x}, u_l)$ and at most one edge from a vertex of this component to a vertex in $d(\mathbf{x}, u_h)$. Moreover, summing over all non-leaf vertices of $\mathcal{T}$ we have

$$\sum_{\substack{u_i \text{ is not a} \\ \text{leaf of } \mathcal{T}}} c_i \leq n_{\mathcal{N}}$$

and hence

$$\sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l\neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(- \sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k\neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} \right) \right) \geq -2n_{\mathcal{N}}$$

$$\Longrightarrow P_{12}(\mathbf{x}) = \sum_{i=0}^{n_{\mathcal{T}}-1} \sum_{\substack{l=0 \\ l\neq i}}^{n_{\mathcal{T}}-1} \left(f(u_i, u_l) \left(1 - \sum_{j=0}^{n_{\mathcal{N}}-1} \sum_{\substack{k=0 \\ k\neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,j} x_{l,k} \right) \right) > -2n_{\mathcal{N}}$$

□

Now, with the last lemma in mind, throughout the remainder of this section, let $A = 2n_{\mathcal{N}}$, and let

$$B = 4n_{\mathcal{N}}^2 n_{\mathcal{T}}^2 > 2 \cdot \min\{A \cdot P_{11}(\mathbf{x}), P_{12}(\mathbf{x})\} \quad \text{for each } \mathbf{x} \in \mathbb{B}^m.$$

We next establish three lemmas.

Lemma 17. *Let $\mathbf{x} \in \mathbb{B}^m$. If $H(\mathbf{x}) = 0$ then $P_I(\mathbf{x}) = 0$ for each $1 \leq I \leq 10$.*

Proof. Suppose that $H(\mathbf{x}) = 0$. By the definition of $H(\mathbf{x})$, we have $P_I(\mathbf{x}) \geq 0$ for all $1 \leq I \leq 10$. Now assume that $P_I(\mathbf{x}) > 0$ for some $1 \leq I \leq 10$. Then

$$H(\mathbf{x}) = B \cdot \left(\sum_{I=1}^{10} P_I(\mathbf{x}) \right) + A \cdot P_{11}(\mathbf{x}) + P_{12}(\mathbf{x}) > 0;$$

a contradiction and the lemma follows. □

Lemma 18. *Let* $\mathbf{x} \in \mathbb{B}^m$. *If* $H(\mathbf{x}) = 0$ *then* $P_{11}(\mathbf{x}) = 0$.

Proof. Suppose that $H(\mathbf{x}) = 0$. By Lemma 17, it follows that $P_I(\mathbf{x}) = 0$ for each $1 \leq I \leq 10$. First, assume that $P_{11}(\mathbf{x}) < 0$. There are two cases to consider.
Case 1. There exists a vertex $u_i \in V(\mathcal{T})$ such that $|d(\mathbf{x}, u_i)| = 0$. Then, $P_1(\mathbf{x}) > 0$; a contradiction.
Case 2. There exist a vertex $u_i \in V(\mathcal{T})$ and a vertex $v_j \in V(\mathcal{N})$ with $x_{i,j} = 1$ such that

$$\sum_{\substack{k=0 \\ k \neq j}}^{n_{\mathcal{N}}-1} g(v_j, v_k) x_{i,k} \geq 2$$

in $P_{11}(\mathbf{x})$. Hence, there exist two edges $(v_j, v_a), (v_j, v_b) \in E(\mathcal{N})$ such that $\{v_j, v_a, v_b\} \subseteq d(x, u_i)$. As, $P_3(\mathbf{x}) = 0$, this implies that $P_4(\mathbf{x}) > 0$; another contradiction.

Second, assume that $P_{11}(\mathbf{x}) > 0$. By Lemma 16, $P_{12}(\mathbf{x}) > -2n_{\mathcal{N}}$. Now, as $P_{11}(\mathbf{x}) > 0$ and $A \cdot P_{11}(\mathbf{x}) + P_{12}(\mathbf{x}) > 0$, with $A = 2n_{\mathcal{N}}$, it follows that $P_{11}(\mathbf{x}) > 0$ implies that $H(\mathbf{x}) > 0$; a final contradiction. This establishes the lemma. □

Lemma 19. *Let* $\mathbf{x} \in \mathbb{B}^m$. *If* $H(\mathbf{x}) = 0$ *then* $P_{12}(\mathbf{x}) = 0$.

Proof. Suppose that $H(\mathbf{x}) = 0$. It follows from Lemmas 17 and 18 that $P_I(\mathbf{x}) = 0$ for each $1 \leq I \leq 11$. Hence, if $H(\mathbf{x}) = 0$, then $P_{12}(\mathbf{x}) = 0$. □

For the next theorem, we need a new definition. Let $\mathcal{T}$ be a leaf-labeled rooted tree, and let X be the leaf set of $\mathcal{T}$. For a vertex u of $\mathcal{T}$, we use $C_{\mathcal{T}}(u)$ to denote the subset of X that precisely contains each element of X that is a descendants of u. Note that, if u is a leaf of $\mathcal{T}$ with label x, then $C_{\mathcal{T}}(u) = \{x\}$. Moreover, if $\mathcal{T}$ is a phylogenetic tree, then it immediately follows that $C_{\mathcal{T}}(u) \neq C_{\mathcal{T}}(u')$ for two distinct vertices of $\mathcal{T}$. We are now in a position to establish the main result of this section.

Theorem 20. *Let $\mathcal{N}$ be a phylogenetic network on X, and let $\mathcal{T}$ be a phylogenetic X-tree. Then, for each* $\mathbf{x} \in \mathbb{B}^m$, $H(\mathbf{x}) = 0$ *if and only if $\mathcal{N}$ displays $\mathcal{T}$.*

Proof. ($\Longrightarrow$) Suppose that $H(\mathbf{x}) = 0$. Then, by Lemmas 17–19, it follows that $P_I(\mathbf{x}) = 0$ with $1 \leq I \leq 12$. We start by deleting edges and vertices in $\mathcal{N}$ as described by the following 2-step process.

(1) Delete each vertex in $d(\mathbf{x}, u_{n_{\mathcal{T}}})$ and each edge that is incident with at least one vertex in $d(\mathbf{x}, u_{n_{\mathcal{T}}})$ in $\mathcal{N}$. Let $\mathcal{N}_1$ be the resulting graph.
(2) For each ordered pair (u_a, u_b) of vertices in $V(\mathcal{T})$ such that $(u_a, u_b) \notin E(\mathcal{T})$, delete each edge from a vertex in $d(\mathbf{x}, u_a)$ to a vertex in $d(\mathbf{x}, u_b)$ in $\mathcal{N}_1$. Let $\mathcal{N}_2$ be the resulting graph.

Recall that, by Lemma 3, $|d(\mathbf{x}, u_0)| = 1$, where u_0 is the root of $\mathcal{T}$ and, by Lemma 12, for each leaf u_i of $\mathcal{T}$, $d(\mathbf{x}, u_i)$ contains the leaf of $\mathcal{N}$ that has the same label as u_i. We next obtain a graph $\mathcal{N}_3$ from $\mathcal{N}_2$ such that $\mathcal{N}_3$ is a subdivision of $\mathcal{T}$. For each $u_l \in V(\mathcal{T}) \setminus \{u_0\}$, it follows from Lemma 13, that $d(\mathbf{x}, u_l)$ is a directed path of $\mathcal{N}$ and therefore, by construction, also of $\mathcal{N}_2$. Let u_i be the unique ancestor of u_l in $\mathcal{T}$. Let $p = p_1, p_2, \ldots, p_k$ be the directed path of $\mathcal{N}$ that is induced by $d(\mathbf{x}, u_i)$ and, similarly, let $p' = p'_1, p'_2, \ldots, p'_{k'}$ be the directed path of $\mathcal{N}$ that is induced by $d(\mathbf{x}, u_l)$. By Lemma 14 and Corollary 15, there exists exactly one edge e in $\mathcal{N}$ that joins a vertex of p to a vertex of p' and, in particular, e is directed out of p_k. Hence $e = (p_k, p'_j)$ for some $1 \leq j \leq k'$. As u_i is the unique parent of u_l in $\mathcal{T}$, any edge in $\mathcal{N}$ that is directed into a vertex in $\{p'_1, p'_2, p'_{j-1}, p'_{j+1}, \ldots, p'_{k'}\}$ and does not lie on p' has been deleted in Step (2) above. Moreover, again by Lemma 14 and Corollary 15, any edge in $\mathcal{N}$ that joins a vertex in $d(\mathbf{x}, u_l)$ with a vertex in $d(\mathbf{x}, u_{l'})$, where $u_{l'}$ is a child of u_l in $\mathcal{T}$, is directed out of $p'_{k'}$. Hence, any edge in $\mathcal{N}$ that is directed out of a vertex in $\{p'_1, p'_2, \dots, p'_{k'-1}\}$ and does not lie on p' has also been deleted in Step (2) above. For the upcoming construction step, we call $\{p'_1, p'_2, \dots, p'_{j-1}\}$ the set of *dangling* vertices in $\mathcal{N}_2$ with respect to u_l. This set may or may not be empty. Now obtain a graph $\mathcal{N}_3$ from $\mathcal{N}_2$ by deleting the set of dangling vertices in $\mathcal{N}_2$ for each vertex $u_l \in V(\mathcal{T})\backslash\{u_0\}$. It is straightforward to check that $\mathcal{N}_3$ is a subdivision of $\mathcal{T}$ and that $\mathcal{T}$ can be obtained from this subdivision by suppressing each vertex with in-degree 1 and out-degree 1 in $\mathcal{N}_3$. Thus, if $H(\mathbf{x}) = 0$, then $\mathcal{N}$ displays $\mathcal{T}$.

($\Longleftarrow$) Suppose that $\mathcal{N}$ displays $\mathcal{T}$. Then there exists a subset V of $V(\mathcal{N})$ and a subset E of $E(\mathcal{N})$ such that $\mathcal{T}$ can be obtained from $\mathcal{N}$ by deleting each vertex in V and each edge in E from $\mathcal{N}$ and, subsequently, suppressing any resulting vertex of in-degree 1 and out-degree 1. Without loss of generality, we choose V such that its size $|V|$ is maximized. Let $\{u_0, u_1, \dots, u_{n_\mathcal{T}-1}\}$ be the vertices of $\mathcal{T}$, and let $\{v_0, v_1, \dots, v_{n_\mathcal{N}-1}\}$ be the vertices of $\mathcal{N}$. Furthermore, let $\mathcal{T}'$ be the graph obtained from $\mathcal{N}$ by deleting each vertex in V and each edge in E from $\mathcal{N}$. By construction, $\mathcal{T}'$ is a subdivision of $\mathcal{T}$. Consider the map $m : V(\mathcal{T}) \to 2^{V(\mathcal{T}')}$ that maps each $u_i \in V(\mathcal{T})$ to a subset of $V(\mathcal{T}')$ such that $m(u_i)$ contains precisely each vertex v_j of $\mathcal{T}'$ with $C_{\mathcal{T}'}(v_j) = C_\mathcal{T}(u_i)$. Now we define $\mathbf{x} \in \mathbb{B}^m$. For each $x_{i,j}$ with $0 \le i \le n_\mathcal{T} - 1$ and $0 \le j \le n_\mathcal{N} - 1$, we set $x_{i,j} = 1$ if $v_j \in m(u_i)$ and, $x_{i,j} = 0$ otherwise. Moreover, for each $0 \le j \le n_\mathcal{N} - 1$, we set $x_{n_\mathcal{T},j} = 1$ if $v_j \in V$ and $x_{n_\mathcal{T},j} = 0$ otherwise. Lastly, to keep with the notation introduced in Section 4.1, let $d(\mathbf{x}, u_i) = m(u_i)$ for each i with $0 \le i \le n_\mathcal{T} - 1$, and let $d(\mathbf{x}, u_{n_\mathcal{T}}) = V$.

We complete the proof by showing that $P_I(\mathbf{x}) = 0$ for each $1 \le I \le 12$.

(1) As $\mathcal{T}'$ is a subdivision of $\mathcal{T}$, the root of $\mathcal{T}'$ is the only vertex of $\mathcal{T}'$ whose set of leaf descendants is X and, so $|d(\mathbf{x}, u_0)| = 1$. Furthermore, by construction, $|d(\mathbf{x}, u_i)| > 0$ for each $u_i \in V(\mathcal{T})$. By Lemma 3, $P_1(\mathbf{x}) = 0$ follows.

(2) For each vertex $v_j \in V(\mathcal{T}')$ there exists exactly one vertex $u_i \in V(\mathcal{T})$ such that $C_{\mathcal{T}'}(v_j) = C_\mathcal{T}(u_i)$. Moreover $d(\mathbf{x}, u_{n_\mathcal{T}}) = V$. Hence $P_2(\mathbf{x}) = 0$ follows from Lemma 4.

(3) Set $z_{i,j} = x_{i,j_1} x_{i,j_2}$, where $u_i \in V(\mathcal{T})$ and $v_j \in V(\mathcal{N})$ is tree vertex. By Lemma 5, this implies that $P_3(\mathbf{x}) = 0$. Similarly, set $z_{i,j} = x_{i,j^1} x_{i,j^2}$, where $u_i \in V(\mathcal{T})$ and $v_j \in V(\mathcal{N})$ is reticulation vertex. Then, by Lemma 7, we have $P_5(\mathbf{x}) = 0$. Lastly, set $\hat{z}_{i,2j} = x_{i,j} x_{i,j_1}$ and $\hat{z}_{i,2j+1} = x_{i,j} x_{i,j_2}$, where $u_i \in V(\mathcal{T})$ and $v_j \in V(\mathcal{N})$ is a tree vertex. It then follows by Lemma 10 that $P_8(\mathbf{x}) = 0$.

(4) Since $\mathcal{T}'$ is a subdivision of $\mathcal{T}$, it follows from the definition of m (and consequently d) that, for each $0 \le i \le n_\mathcal{T} - 1$, the subgraph of $\mathcal{N}$ that is induced by the vertices in $d(\mathbf{x}, u_i)$ is a directed path. Now, towards a contradiction, assume that there exist a vertex u_i in $\mathcal{T}$ and a vertex v_j in $\mathcal{N}$ such that $\{\{v_j, c_1(v_j), c_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$. Since the subgraph of $\mathcal{N}$ that is induced by the vertices in $d(\mathbf{x}, u_i)$ is a directed path, we may assume without loss of generality that there is a directed path $(c_1(v_j) = p_1, p_2, \dots, p_k = c_2(v_j))$ in $\mathcal{N}$ with $k \ge 2$. In particular, each vertex on this path is contained in $d(\mathbf{x}, u_i)$. It now follows that we can obtain a subdivision of $\mathcal{T}$ from $\mathcal{N}$ by deleting all vertices in $V \cup \{p_1, p_2, \dots, p_{k-1}\}$ and edges in

$$(E\backslash(v_j, p_k)) \cup \{(v_j, p_1), (p_1, p_2), \dots, (p_{k-1}, p_k)\}.$$

This contradicts the maximality of $|V|$. Hence, there exist no two vertices u_i and v_j such that $\{v_j, c_1(v_j), c_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$. An analogous contradiction can be used to establish that there exist no two vertices u_i in $\mathcal{T}$ and v_j in $\mathcal{N}$ such that $\{v_j, p_1(v_j), p_2(v_j)\} \subseteq d(\mathbf{x}, u_i)$ Thus, by Lemmas 6, 8, and 13, we have $P_4(\mathbf{x}) = P_6(\mathbf{x}) = P_{11}(\mathbf{x}) = 0$.

(5) Again, since $\mathcal{T}'$ is a subdivision of $\mathcal{T}$, for each $(u_a, u_b) \in E(\mathcal{T})$, there exists an edge that joins the terminal vertex v_r of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_a)$ to the first vertex v_s of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_b)$. We now move towards several contradictions. Assume that there exist more than one edge from a vertex of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_a)$ to a vertex of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_b)$. This is, there exist $v_q, v_r \in d(\mathbf{x}, u_a)$ and $v_s, v_t \in d(\mathbf{x}, u_b)$ such that $(v_q, v_t), (v_r, v_s) \in E(\mathcal{N})$. Let $v_q = p_1, p_2, \dots, p_k = v_r$ be the directed path from v_q to v_r in $\mathcal{N}$ and, similarly, let $v_s = p'_1, p'_2, \dots, p'_{k'} = v_t$ be the directed path from v_s to v_t in $\mathcal{N}$. Since $\mathcal{N}$ has no edges in parallel, $k \ge 2$ or $k' \ge 2$. If $v_q = v_r$, then $v_s \ne v_t$ and, consequently, a subdivision of $\mathcal{T}$ can be obtained from $\mathcal{N}$ by deleting all vertices in $V \cup \{p'_1, p'_2, \dots, p'_{k'-1}\}$ and all edges in $E \cup \{(p'_1, p'_2), (p'_2, p'_3), \dots, (p'_{k'-1}, p'_{k'})\}$; thereby contradicting the maximality of $|V|$. Similarly, if $v_s = v_t$, then we obtain a subdivision of $\mathcal{T}$ from $\mathcal{N}$ by deleting all vertices in $V \cup \{p_2, p_3, \dots, p_k\}$ and all edges in $E \cup \{(p_1, p_2), (p_2, p_3), \dots, (p_{k-1}, p_k)\}$; again contradicting the maximality of $|V|$. We may therefore assume without loss of generality that $k \ge 2$ and $k' \ge 2$. Now recall that (v_q, v_t) and (v_r, v_s) are edges in $\mathcal{N}$. Then a subdivision of $\mathcal{T}$ can be obtained from $\mathcal{N}$ by by deleting all vertices in $V \cup \{p_2, p_3, \dots, p_k, p'_1, p'_2, \dots, p'_{k'-1}\}$ and all edges in $E \cup \{(p_1, p_2), (p_2, p_3), \dots, (p_{k-1}, p_k), (p'_1, p'_2), (p'_2, p'_3), \dots, (p'_{k'-1}, p'_{k'})\}$; a final contradictions. It follows that there exists exactly one edge from a vertex in $d(\mathbf{x}, u_a)$ to a vertex in $d(\mathbf{x}, u_b)$. In particular, this edge joins the terminal

vertex of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_a)$ to the first vertex of the directed path in $\mathcal{N}$ induced by $d(\mathbf{x}, u_b)$. Hence, by Lemmas 9, 11, and 14, we have $P_7(\mathbf{x}) = P_9(\mathbf{x}) = P_{12}(\mathbf{x}) = 0$.

(6) Clearly, for each $u_i \in V(\mathcal{T})$ that is a leaf, we have $v_j \in d(\mathbf{x}, u_i)$, where v_j is the leaf vertex of $\mathcal{T}'$ (and $\mathcal{N}$) whose label is identical to that of u_i. Hence, by Lemma 12, it follows that $P_{10}(\mathbf{x}) = 0$.

Therefore, if $\mathcal{N}$ displays $\mathcal{T}$, then $H(\mathbf{x}) = 0$. □

We end this section by noting that $H(\mathbf{x})$ contains constant and linear term. Strictly speaking, $H(\mathbf{x})$ is therefore not in QUBO form. However, $H(\mathbf{x})$ can easily be converted into QUBO form by replacing each linear term x_i with the quadratic term x_i^2 and deleting any constant term as it has no impact on the optimization.

5. Example and Results

In this section, we present an example to illustrate the QUBO formulation of Tree Containment and some minor embedding results.

5.1. Example

Consider the phylogenetic network $\mathcal{N}$ on X and the two phylogenetic X-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ as shown in Figure 4. We wish to answer the following two questions: (i) Does $\mathcal{N}$ display $\mathcal{T}_1$? (ii) Does $\mathcal{N}$ display $\mathcal{T}_2$?

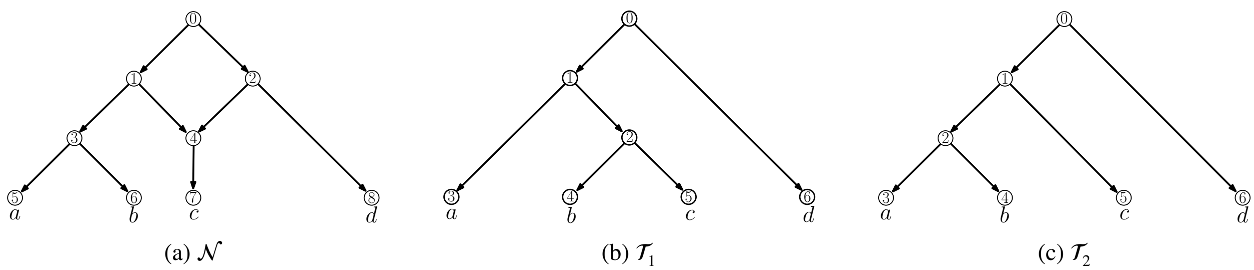


Figure 4: A phylogenetic network $\mathcal{N}$ on X in (a) and two phylogenetic X-trees $\mathcal{T}_1$ and $\mathcal{T}_2$ in (b) and (c), where $X = \{a, b, c, d\}$. Each vertex in $\mathcal{N}$, $\mathcal{T}_1$, and $\mathcal{T}_2$ has been assigned a number. The sole purpose of those numbers, which are not vertex labels, is the representation of $\mathcal{N}$, $\mathcal{T}_1$, and $\mathcal{T}_2$ as adjacency matrices.

After processing of $H(\mathbf{x})$ as described in the last paragraph of Section 4.2, the resulting objective function is in QUBO form. Recall that

$$\mathbf{x} \in \mathbb{B}^{n_{\mathcal{N}}(n_{\mathcal{T}}+1)+(n_{\mathcal{T}}-1)(1+\lfloor \lg(n_{\mathcal{N}}-n_{\mathcal{T}}) \rfloor)+n_{\mathcal{T}}(\alpha+\beta)+2\beta\gamma}.$$

Then, for each of (i) and (ii), we need

$$9 \cdot (7+1) + (7-1)(1+\lg(2)) + 7 \cdot (1+4) + 2 \cdot 4 \cdot 3 = 143$$

logical qubits and, so, the size of the output QUBO matrix is 143×143.

For Question (i), D-Wave's quantum annealer returned only one binary vector $\mathbf{x}^*$ as follows, where subscripts refer to the vertex numbers as shown in Figure 4: $x_{3,5} = x_{2,4} = x_{1,1} = x_{3,9} = x_{6,8} = x_{4,6} = x_{5,7} = x_{0,0} = x_{6,2} = x_{6,9} = x_{3,3} = 1$ and all other qubits are 0. After post-processing, we did not get 0 as minimum output x^*. We confirmed this result by querying D-Wave's quantum annealer 100 times and, each time, the result was $x^* = 1$. Therefore, there is a high probability that $\mathcal{N}$ does not display $\mathcal{T}_1$. Indeed, $\mathcal{N}$ does not display $\mathcal{T}_1$.

For Question (ii), D-Wave's quantum annealer returned a binary vector $\mathbf{x}^*$ with $x_{3,5} = x_{1,1} = x_{6,8} = x_{4,6} = x_{5,7} = x_{0,0} = x_{5,4} = x_{5,9} = x_{2,3} = x_{6,2} = x_{6,9} = 1$ and all other qubits are 0. After post-processing, we did get 0 as minimum output x^*. It follows that $\mathcal{N}$ displays $\mathcal{T}_2$ which is indeed the case.

5.2. Minor Embedding Results

As a proof-of-concept, we performed several experiments to investigate the number of logical and physical qubits that are needed to solve an instance of Tree Containment. We start by providing some information about the host graphs. After formulating an instance of Tree Containment as an instance of QUBO, we used the minor embedding algorithm

Table 1
The number of logical qubits and QUBO density.

$\lvert X\rvert$	r	Logical Qubits	Density
4	1	143	0.088348
4	3	221	0.066680
4	2	185	0.075088
5	3	320	0.052704
5	4	374	0.048530
5	2	274	0.058581
6	1	313	0.054211
6	5	557	0.037721
6	3	435	0.043551
7	2	500	0.040473
7	4	644	0.034558
7	3	566	0.037099
8	5	879	0.028493
8	4	803	0.030236
8	3	713	0.032309

provided by D-Wave [10]. For some of our test cases (described below), the QUBO matrix could not be embedded into the host graph of D-Wave Advantage. We therefore considered a larger Pegasus graph. This allows us to compare the current capacity of the D-Wave Advantage annealer with a possible future version of the D-Wave machine. Both host graphs that we considered were of Pegasus topology. The first host graph, D-Wave Advantage, has 5640 vertices and 40484 edges, and the second host graph has 23560 vertices and 172964 edges. The first and second host graphs are represented by $P16$ and $P32$, respectively. In comparison, $P32$ has about 4.17 times more vertices and about 4.27 times more edges than $P16$. For more information about the Pegasus topology, see [9].

In total, we have analyzed 15 small instances of Tree Containment, where each instance consists of a phylogenetic network $\mathcal{N}$ on X and a phylogenetic X-tree $\mathcal{T}$. In Table 1, the first column contains the size of X, the second column contains the number r of reticulation vertices in $\mathcal{N}$ and the third column contains the number of binary variables in the QUBO instance obtained from applying the approach described in Section 4.1 to $\mathcal{N}$ and $\mathcal{T}$. Finally, the last column contains the density of the graph G whose weighted adjacency matrix is the QUBO matrix, where the density is defined as the ratio of the size of G and the size of the complete graph whose order is the same as that of G. Note that, in the approach described in Section 4.1, the number of binary variables in the resulting QUBO instance depends only linearly on the total number of vertices in $\mathcal{N}$ and $\mathcal{T}$. As $\mathcal{N}$ and $\mathcal{T}$ both have leaf set X, it follows that $n_{\mathcal{T}} = 2|X| - 1$ and $n_{\mathcal{N}} = 2|X| + 2r - 1$ vertices [29, Lemma 2.1]. From our initial experiments, we can see some good news in that the number of logical qubits are expectantly small and that the densities are relatively low. This later fact implies that fewer qubit connections will be required and non-completely connected quantum annealers will have a better chance of embedding the logical structure onto a physical structure. In Table 2, we present the minor embedding results for both host graphs $P16$ and $P32$. The first two columns of this table are identical to the first two columns of Table 1. The *Physical Qubits* column contains our experimental results indicating the number of physical qubits required to embed a QUBO instance, depending on which host graph was used. Lastly, the *Max Chain Size* column contains the maximum number of physical qubits a single logical qubit was mapped onto in our experiments. The entries marked by '-' correspond to the cases where the minor embedding algorithm was not able to find a minor embedding. Regrettably, but not unexpectedly, the number of physical qubits grows beyond the capabilities of current quantum annealing architectures such as those used by D-Wave, even for small test cases. However, it is worth noting that we do not have exponential growth with respect to the input sizes. Furthermore, the relatively large maximum chain sizes is of a concern, but with advances in the expected qubit interconnection density of future hardware (and possibly improved embedding algorithms) this can likely be mitigated in practice.

Table 2
Minor embedding results.

$\|X\|$	r	Physical Qubits		Max Chain Size	
		$P16$	$P32$	$P16$	$P32$
4	1	721	692	14	14
4	3	1396	1371	21	21
4	2	1144	1024	18	18
5	3	2602	2819	28	29
5	4	3526	3031	31	26
5	2	2073	1988	24	24
6	1	2930	3255	29	34
6	5	-	7375	-	47
6	3	4480	5170	38	40
7	2	-	6859	-	50
7	4	-	9797	-	56
7	3	-	8350	-	49
8	5	-	16442	-	75
8	4	-	16749	-	89
8	3	-	13276	-	77

6. Conclusion

In this paper, we have discussed the AQC model, which has gained popularity in recent years, to solve Tree Containment. Currently, the size of problem instances (of Tree Containment as well as other problems) that can be solved using quantum annealers is one of the main setbacks that is hindering quantum computing from becoming a mainstream technology. The development of different approaches to build bigger and more efficient quantum annealers is an ongoing and major effort.

In Section 4, we have established an efficient reduction from Tree Containment to QUBO. A Python program that reads in a phylogenetic network and tree and outputs the resulting QUBO instance is given in Appendix A. We have shown that an instance of Tree Containment, say $(\mathcal{N}, \mathcal{T})$, can be reduced to an instance of QUBO whose number of logical qubits is $O(n_{\mathcal{N}} n_{\mathcal{T}})$ Furthermore, the reduction from Tree Containment to QUBO takes polynomial time. Our reduction has the following special property: solving (minimizing) the QUBO returns 0 after post-processing if and only if $\mathcal{N}$ displays $\mathcal{T}$. In addition, we note that the above QUBO formulation is also correct if $\mathcal{T}$ has leaf set X' and $\mathcal{N}$ has leaf set X such that $X' \subset X$.

In Section 5, we have experimentally evaluated the efficiency of the QUBO formulation for Tree Containment. We have compared the efficiency in terms of logical qubits, physical qubits, density, and the maximum chain size. Our experiments indicate that current quantum annealers can only solve small instances of Tree Containment, which we note can also be solved with a classical approach. Nevertheless, our results provide a first indication that other problems arising in studying phylogenetic trees and networks could potentially also be attacked within AQC. For example, many problems that arise in the reconstruction of phylogenetic networks have underlying NP-hard optimization problems. Current algorithms in this area either do not scale up to large data sets or are heuristics with no guarantee on the optimality of the solution [22]. Hence, AQC offers a promising and alternative approach to solving such problems. Furthermore, it would be interesting to explore quantum-classical hybrid approaches, as discussed in [1]. An immediate next step could be the development of a QUBO for problems that are closely related to Tree Containment such as the problem of deciding if a given phylogenetic network contains a given phylogenetic tree as a so-called base tree, or the problem of deciding if two phylogenetic networks display the same set of phylogenetic trees [3, 13, 18]. How different are QUBO formulations for these problems from the QUBO presented in this paper?

Acknowledgements. We thank Cristian Calude and Richard Hua for helpful discussions.

References

[1] A. A. Abbott, C. S. Calude, M. J. Dinneen, and R. Hua. A hybrid quantum-classical paradigm to mitigate embedding costs in quantum annealing. *International Journal of Quantum Information*, 17(05):1950042, 2019.

[2] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM Review*, 50(4):755–787, 2008.

[3] M. Anaya, O. Anipchenko-Ulaj, A. Ashfaq, J. Chiu, M. Kaiser, M. S. Ohsawa, M. Owen, E. Pavlechko, K. S. John, S. Suleria, et al. On determining if tree-based networks contain fixed trees. *Bulletin of Mathematical Biology*, 78(5):961–969, 2016.

[4] E. Bapteste, L. van Iersel, A. Janke, S. Kelchner, S. Kelk, J. O. McInerney, D. A. Morrison, L. Nakhleh, M. Steel, L. Stougie, et al. Networks: expanding evolutionary thinking. *Trends in Genetics*, 29(8):439–441, 2013.

[5] C. Blais and J. M. Archibald. The past, present and future of the tree of life. *Current Biology*, 31(7):R314–R321, 2021.

[6] M. Bordewich and C. Semple. Reticulation-visible networks. *Advances in Applied Mathematics*, 78:114–141, 2016.

[7] C. S. Calude and M. J. Dinneen. Solving the broadcast time problem using a D-wave quantum computer. In A. Adamatzky, editor, *Advances in Unconventional Computing. Emergence, Complexity and Computation*, volume 22, pages 439–453. Springer, 2017.

[8] C. S. Calude, M. J. Dinneen, and R. Hua. QUBO formulations for the graph isomorphism problem and related problems. *Theoretical Computer Science*, 701:54–69, 2017.

[9] D-Wave. D-Wave QPU architecture: Topologies. `https://docs.dwavesys.com/docs/latest/c_gs_4.html`.

[10] D-Wave. Minorminer. `https://docs.ocean.dwavesys.com/en/stable/docs_minorminer/source/sdk_index.html`.

[11] D-Wave. qbsolv. `https://docs.ocean.dwavesys.com/projects/qbsolv/en/latest`.

[12] D. Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[13] J. Döcker, S. Linz, and C. Semple. Displaying trees across two phylogenetic networks. *Theoretical Computer Science*, 796:129–146, 2019.

[14] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[15] A. Fedorov and M. Gelfand. Towards practical applications in quantum computational biology. *Nature Computational Science*, 1(2):114–119, 2021.

[16] J. Felsenstein. *Inferring phylogenies*. Sinauer Associates Sunderland, 2004.

[17] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488, 1982.

[18] A. R. Francis and M. Steel. Which phylogenetic networks are merely trees with additional arcs? *Systematic Biology*, 64(5):768–777, 2015.

[19] F. Glover, G. Kochenberger, and Y. Du. A tutorial on formulating and using QUBO models. *arXiv preprint arXiv:1811.11538*, 2019.

[20] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[21] A. D. Gunawan, B. DasGupta, and L. Zhang. Locating a tree in a reticulation-visible network in cubic time. In *Proceedings of the 20th Annual Conference on Research in Computational Molecular Biology*, page 266, 2016.

[22] H. A. Hejase and K. J. Liu. A scalability study of phylogenetic network inference methods using empirical datasets and simulations involving a single reticulation. *BMC Bioinformatics*, 17(1):1–12, 2016.

[23] D. H. Huson, R. Rupp, and C. Scornavacca. *Phylogenetic networks: concepts, algorithms and applications*. Cambridge University Press, 2010.

[24] W. Jetz, G. H. Thomas, J. B. Joy, K. Hartmann, and A. O. Mooers. The global diversity of birds in space and time. *Nature*, 491(7424):444–448, 2012.

[25] I. A. Kanj, L. Nakhleh, C. Than, and G. Xia. Seeing the trees and their branches in the network is hard. *Theoretical Computer Science*, 401:153–164, 2008.

[26] E. V. Koonin, K. S. Makarova, and L. Aravind. Horizontal gene transfer in prokaryotes: quantification and classification. *Annual Reviews in Microbiology*, 55(1):709–742, 2001.

[27] A. Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2:5, 2014.

[28] A. Mahasinghe, R. Hua, M. J. Dinneen, and R. Goyal. Solving the hamiltonian cycle problem using a quantum computer. In *Proceedings of the Australasian Computer Science Week Multiconference*, pages 1–9, 2019.

[29] C. McDiarmid, C. Semple, and D. Welsh. Counting phylogenetic networks. *Annals of Combinatorics*, 19(1):205–224, 2015.

[30] C. C. McGeoch. Adiabatic quantum computation and quantum annealing: Theory and practice. *Synthesis Lectures on Quantum Computing*, 5(2):1–93, 2014.

[31] J. Ottenburghs. Multispecies hybridization in birds. *Avian Research*, 10(1):1–11, 2019.

[32] C. Outeiral, M. Strahm, J. Shi, G. M. Morris, S. C. Benjamin, and C. M. Deane. The prospects of quantum computing in computational molecular biology. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 11(1):e1481, 2021.

[33] P. M. Pardalos and S. Jha. Complexity of uniqueness and local search in quadratic 0–1 programming. *Operations Research Letters*, 11(2):119–123, 1992.

[34] A. O. Richardson and J. D. Palmer. Horizontal gene transfer in plants. *Journal of Experimental Botany*, 58(1):1–9, 2007.

[35] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[36] S. M. Soucy, J. Huang, and J. P. Gogarten. Horizontal gene transfer: building the web of life. *Nature Reviews Genetics*, 16(8):472–482, 2015.

[37] L. van Iersel, C. Semple, and M. Steel. Locating a tree in a phylogenetic network. *Information Processing Letters*, 110(23):1037–1043, 2010.

[38] M. Weller. Linear-time tree containment in phylogenetic networks. In *RECOMB International Conference on Comparative Genomics*, pages 309–323. Springer, 2018.

A. Python Program that Generates the QUBO Formulation for an Instance of Tree Containment

```
5       # order of T
1 4     # adjacency list of T
2 3



7       # order of N
1 2     # adjacency list of N
3 4
3 6
5



3       # number of leaves
2 8     # which  leaf in T corresponds  to which  leaf in N.
3 3
4 4
```

Listing 1: Sample input of a phylogenetic tree and networks. Remove the comments before passing to the program.

```
import networkx as nx
import sys
import math
from pyqubo import Array


def read_input():
    n = int(sys.stdin.readline().strip())
    T = nx.empty_graph(n, create_using=nx.DiGraph())
    for u in range(n):
        neighbors = sys.stdin.readline().split()
        for v in neighbors:
            T.add_edge(u, int(v))

    n = int(sys.stdin.readline().strip())
    N = nx.empty_graph(n, create_using=nx.DiGraph())
    for u in range(n):
        neighbors = sys.stdin.readline().split()
        for v in neighbors:
            N.add_edge(u, int(v))

    # leaf_correspondence represents which leaf in T corresponds to which leaf in N.
    # e.g. {5:7, 2:3} means leaf with index 5 in T corresponds to leaf with index 7 in N.
    leaf_correspondence = {}
    n = int(sys.stdin.readline().strip())
    for i in range(n):
        leaves = sys.stdin.readline().split()
        leaf_correspondence[int(leaves[0])] = int(leaves[1])
    return (T, N, leaf_correspondence)


def edge(u, v, graph):
    if v in list(graph.neighbors(u)):
        return True
    return False


def get_tree_vertices(graph):
    tree_vertices = []
    for i in range(graph.order()):
        if graph.out_degree(i) == 2:
            tree_vertices.append(i)
```

```
    return tree_vertices


def get_reticulation_vertices(graph):
    reticulation_vertices = []
    for i in range(graph.order()):
        if graph.in_degree(i) == 2:
            reticulation_vertices.append(i)

    return reticulation_vertices


def get_leaf_vertices(graph):
    leaf_vertices = []
    for i in range(graph.order()):
        if graph.out_degree(i) == 0:
            leaf_vertices.append(i)

    return leaf_vertices


def generate_qubo(T, N, leaf_correspondence):
    a = T.order() + 1
    if T.order() > N.order():
        raise Exception('N must have at least as many vertices as T.'
                        )
    elif T.order() == N.order():
        b = N.order()
    else:
        b = N.order() + 1 + math.floor(math.log2(N.order() - T.order()))
    x = Array.create('x', shape=(a, b), vartype='BINARY')
    z = Array.create('z', shape=(T.order(), N.order()), vartype='BINARY'
                     )
    zhat = Array.create('zhat', shape=(T.order(), 2 * N.order()),
                        vartype='BINARY')

    A = 2 * N.order()
    B = 4 * N.order() ** 2 * T.order() ** 2

    P_1 = 0
    for i in range(T.order()):
        temp = 1
        for j in range(N.order()):
            temp += -x[i, j]
        if i != 0:
            for r in range(N.order(), b):
                temp += 2 ** (r - N.order()) * x[i, r]
        P_1 += temp ** 2

    P_2 = 0
    for j in range(N.order()):
        temp = 1
        for i in range(T.order() + 1):
            temp += -x[i, j]
        P_2 += temp ** 2

    P_3 = 0
    for i in range(T.order()):
        for j in get_tree_vertices(N):
            children = list(N.neighbors(j))
            P_3 += x[i, children[0]] * x[i, children[1]] - 2 * x[i,
                    children[0]] * z[i, j] - 2 * x[i, children[1]] \
                * z[i, j] + 3 * z[i, j]

    P_4 = 0
    for i in range(T.order()):
```

```
        for j in get_tree_vertices(N):
            P_4 += x[i, j] * z[i, j]

    P_5 = 0
    for i in range(T.order()):
        for j in get_reticulation_vertices(N):
            parents = list(N.predecessors(j))
            P_5 += x[i, parents[0]] * x[i, parents[1]] - 2 * x[i,
                    parents[0]] * z[i, j] - 2 * x[i, parents[1]] * z[i,
                    j] + 3 * z[i, j]

    P_6 = 0
    for i in range(T.order()):
        for j in get_reticulation_vertices(N):
            P_6 += x[i, j] * z[i, j]

    P_7 = 0
    for i in range(T.order()):
        for l in range(T.order()):
            if i != l and edge(i, l, T):
                for j in get_tree_vertices(N):
                    P_7 += x[i, j] * z[l, j]

    P_8 = 0
    for i in range(T.order()):
        if i in get_leaf_vertices(T): continue
        for j in get_tree_vertices(N):
            children = list(N.neighbors(j))
            P_8 += x[i, j] * x[i, children[0]] - 2 * x[i, j] \
                * zhat[i, 2 * j] - 2 * x[i, children[0]] \
                * zhat[i, 2 * j] + 3 * zhat[i, 2 * j]
            P_8 += x[i, j] * x[i, children[1]] - 2 * x[i, j] \
                * zhat[i, 2 * j + 1] - 2 * x[i, children[1]] \
                * zhat[i, 2 * j + 1] + 3 * zhat[i, 2 * j + 1]

    P_9 = 0
    for i in range(T.order()):
        for l in range(T.order()):
            if i != l and edge(i, l, T):
                for j in get_tree_vertices(N):
                    children = list(N.neighbors(j))
                    P_9 += zhat[i, 2 * j] * x[l, children[1]] + zhat[i,
                            2 * j + 1] * x[l, children[0]]

    P_10 = 0
    for i in get_leaf_vertices(T):
        P_10 += 1 - x[i, leaf_correspondence[i]]

    P_11 = 0
    for i in range(T.order()):
        for j in range(N.order()):
            temp = 1
            for k in range(N.order()):
                if k != j and edge(j, k, N):
                    temp += -x[i, k]
            P_11 += x[i, j] * temp
    P_11 += -T.order()

    P_12 = 0
    for i in range(T.order()):
        for l in range(T.order()):
            if i != l and edge(i, l, T):
                temp = 1
                for j in range(N.order()):
                    for k in range(N.order()):
                        if j != k and edge(j, k, N):
                            temp += -x[i, j] * x[l, k]
```

```
                P_12 += temp

    H = B * (P_1 + P_2 + P_3 + P_4 + P_5 + P_6 + P_7 + P_8 + P_9
             + P_10) + A * P_11 + P_12

    model = H.compile()
    (qubo, offset) = model.to_qubo()
    return (qubo, model, offset)


(T, N, leaf_correspondence) = read_input()
(qubo, model, offset) = generate_qubo(T, N, leaf_correspondence)
```

Listing 2: Python code to generate the QUBO for an instance of Tree Containment