Exact and Efficient Inference of Tumor Phylogenies via Novel Pruning Techniques

Juan Luque
Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Jacob Gilbert
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD 20894, USA

Arjun Subramanian
The Pingry School, Basking Ridge, NJ 07920, USA

Aravind Srinivasan*
Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Salem Malikic*
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD 20894, USA

S. Cenk Sahinalp*
Cancer Data Science Laboratory, National Cancer Institute, Bethesda, MD 20894, USA

* Joint last authors

Abstract

Reconstructing the evolutionary history of tumors using single-cell sequencing (SCS) data presents significant computational challenges. Existing approaches are either computationally intractable for emerging large-scale datasets or rely on heuristics that lack optimality guarantees. In this work, we propose a novel, time-efficient algorithm that constructs the phylogenetic tree of tumor evolution with a provable guarantee of optimality.

Our main result is a branch-and-bound algorithm that reconstructs the most likely tumor evolutionary history up to two orders of magnitude faster than the previous best algorithm. To achieve this, we use efficient and well-known 2-approximation algorithms for the Vertex Cover problem to prune the branch-and-bound tree effectively. Unlike previous works' polynomial-time branch-and-bound bounding strategies, our bounding algorithm provides strong worst-case theoretical guarantees, leading to faster reconstruction of the tumor evolution.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Branch and Bound, Vertex Cover, Linear Programming, Tumor Evolution, Single-Cell Sequencing

Digital Object Identifier 10.4230/LIPIcs.WABI.2026.11

Supplementary Material *Software*: `https://github.com/jdluque/faster-tphyl-reconstruction`

Funding S.C.S., S.M. and J.G. were supported by the Intramural Research Program of the National Cancer Institute (NCI), National Institutes of Health.

26th International Conference on Algorithms for Bioinformatics (WABI 2026).
Editors: Nadia El-Mabrouk and Fabio Vandin; Article No. 11; pp. 11:0–11:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Cancer is an evolutionary disease, driven by the progressive accumulation of somatic mutations, which give rise to genetically distinct subclones within a tumor. A history of tumor progression can be represented as a phylogenetic tree of tumor evolution, which captures the lineage relationships among tumor cells and provides insights into early driver mutations, patterns of metastatic seeding, mutually exclusive and co-occurring mutations, and clonal expansions.

The inference of trees of tumor evolution has been one of the very important problems in cancer genomics research. Early methods predominantly relied on bulk sequencing data, which provides trees of limited resolution and has well known limitations [11, 14]. However, advances in single-cell sequencing (SCS) technologies have enabled much more detailed studies of tumor evolution. Consequently, there has been rapid progress in the development of computational methods for inferring trees of tumor evolution from this type of data.

Existing methods for inferring tumor evolutionary trees from SCS data can broadly be classified into two main categories. The first category consists of heuristic-based approaches, including techniques such as Monte Carlo Markov Chain (MCMC) sampling [10, 17, 21, 26, 23], nearest-neighbor interchange [25], and greedy search [12]. While these methods typically return some solution tree, they do not offer any guarantees of optimality for the reported solutions. Their notable limitations include terminating at locally optimal but globally suboptimal solutions, and, in some cases, sensitivity to minor variations in the input (e.g., permutations in the order of cells in the input can lead to different outputs, despite the inputs being effectively equivalent).

To address some of the limitations discussed above, a second category of methods has been developed. These methods are typically based on Integer Linear Programming (ILP) [16, 7, 19, 6, 5], and rely on established ILP solvers such as Gurobi or CPLEX. Unlike heuristic-based approaches, ILP-based methods can offer guarantees of optimality when they successfully converge to optimal solutions. However, they are often prohibitively slow on larger datasets containing hundreds of mutations or cells, which significantly limits their practical applicability on emerging single-cell datasets. To mitigate this scalability issue, some methods [7] apply pre-clustering of cells as a preprocessing step. While this can reduce computational complexity, it often comes at the cost of reduced accuracy and/or lower resolution in the inferred evolutionary trees [12].

One potential direction for improving the efficiency of ILP-based methods involves designing specialized algorithms that exploit structural properties of the tumor phylogeny inference problem. The goal of such algorithms is to accelerate the solution of the underlying ILP without sacrificing accuracy. A notable example of this approach is the recently developed PhISCS-BnB, which leverages branch and bound techniques to efficiently search the solution space [18]. PhISCS-BnB has been shown to match the accuracy of other ILP-based methods on smaller datasets, while outperforming them on larger instances. In particular, for some large-scale inputs, it was the only method capable of producing an optimal solution (along with a certificate of optimality) within a reasonable time.

While PhISCS-BnB marked a significant advancement over previous state-of-the-art methods in terms of performance, it continued to face a critical limitation: in the worst case, it may explore an exponential number of search tree nodes, solving an NP-complete problem at each node. This limitation constrains its scalability, especially in the context of increasingly large and complex datasets. As a result, the development of more effective pruning strategies—particularly those with *polynomial-time* complexity—remained a key

open challenge.

Building upon the principled framework and formal optimality guarantees introduced in PhISCS-BnB, our work advances the field by enabling even faster phylogeny reconstruction. Specifically, we replace the computationally intensive SAT solvers previously used for branch-and-bound pruning with efficient, polynomial-time Linear Programming (LP) solvers. Furthermore, we provide optimality guarantees for our newly introduced pruning algorithms, laying the groundwork for continued improvements in both the scalability and robustness of phylogeny inference methods.

Notably, the use of PDLP [1], Google's freely-available recent Linear Programming (LP) solver, enabled our algorithms to achieve the fastest phylogeny reconstruction times in our experimental evaluations. In this context, our work not only initiates research into provably more powerful pruning algorithms for phylogeny reconstruction but is also expected to benefit from ongoing advancements in LP solvers.

2 Background and problem definition

We assume that a single-cell sequencing experiment has been conducted, in which single cells $C = \{C_1, C_2, \ldots, C_n\}$ were sampled from a tumor and sequenced. We also assume that, after the mutation calling step, we are given a set of mutations $M = \{M_1, M_2, \ldots, M_m\}$ that are reported to be present in at least one of these cells. Therefore, our input can be summarized as an *observed genotype matrix*, denoted in this work as I, consisting of n rows (corresponding to cells) and m columns (corresponding to mutations). In this matrix, $I_{ip} = 1$ indicates that the mutation calling step reported mutation M_p to be present in cell C_i, while $I_{ip} = 0$ indicates its absence. Given I, our goal is to reconstruct the most parsimonious evolutionary history of the tumor.

Before providing a rigorous mathematical definition of the computational problem we aim to solve, we first present the model of evolution used in this work and define how we represent the evolutionary history of a tumor. To simplify our presentation, we focus on a hypothetical case in which the ground truth is available. In the ground truth, cells from C and mutations from M define the *ground truth genotype matrix* T, which, analogous to I, is a binary matrix of size $n \times m$, with 0 and 1 respectively indicating the absence and presence of a mutation in a cell. An example of such a matrix is shown in Figure 1a.

In this work, we assume the commonly used Infinite Sites Assumption (ISA), which states that each mutation is acquired at most once during the course of tumor evolution and, once acquired by a cell, is never lost in any of its descendants. This assumption implies that tumor evolution follows the Perfect Phylogeny Model, in which mutations are inherited along the edges of a rooted tree. More precisely, under the ISA, the ground truth matrix T defines a unique *tree of tumor evolution* (referred to as the tree in the remainder of this paper, unless otherwise specified), which is a rooted tree consisting of n leaves, each labeled with a distinct cell from C. The root of the tree represents the germline (i.e., normal cells), which we assume contains no somatic mutations. The edges are labeled with mutations such that each edge that is not incident to a leaf or the root contains at least one mutational label, whereas the other edges may or may not contain such labels.[1] A mutation assigned to the edge connecting a node v to its parent is assumed to be present in the cell labeling v (if v

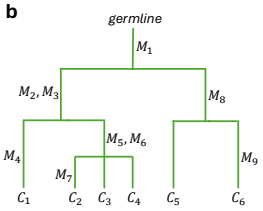
[1] Note that the ground truth evolutionary history of the sequenced cells is a binary tree; however, in our case, we may not have sufficient signal to reconstruct a fully resolved binary tree from T. In other words, the trees used in this work may contain polytomies (i.e., internal nodes with more than two children).

is a leaf), or in all cells labeling the descendant leaves of v (if v is an internal node). This implies that the set of mutations harbored by cell C_i can be obtained by taking the union of all mutations appearing on the unique path connecting the leaf labeled by C_i to the root of the tree. Note that, for each cell, the presence/absence pattern of mutations in that cell is consistent between the ground truth matrix T and what is implied by the tree corresponding to T. In general, when this property holds for a binary matrix and a tree of tumor evolution, we say that the tree and the matrix are *consistent*. The tree consistent with the ground truth matrix shown in Figure 1a is presented in Figure 1b.

a

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
C_1	1	1	1	1	0	0	0	0	0
C_2	1	1	1	0	1	1	1	0	0
C_3	1	1	1	0	1	1	0	0	0
C_4	1	1	1	0	1	1	0	0	0
C_5	1	0	0	0	0	0	0	1	0
C_6	1	0	0	0	0	0	0	1	1

b



c

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
C_1	1	1	1	1	0	0	0	0	0
C_2	1	1	1	0	1	1	1	0	0
C_3	1	1	1	0	1	0	0	0	0
C_4	1	0	1	0	1	1	0	0	0
C_5	1	0	0	0	0	0	0	0	0
C_6	0	0	0	0	0	0	0	1	1

Figure 1 **a**. Ground truth genotype matrix. **b**. Tree of tumor evolution consistent with the ground truth matrix shown in (a). **c**. Observed genotype matrix obtained from real single-cell sequencing data. This matrix typically differs from the ground truth (red entries) and often contains conflicts, which imply that no tree is consistent with the matrix. An example of such a conflict is highlighted in yellow.

While it is straightforward to reconstruct the tree consistent with the ground truth matrix (and this can be done in polynomial time using the algorithm introduced in [9]), the ground truth matrix is unknown in practice. As discussed above, what we are given instead as input is an observed genotype matrix I obtained from a single-cell sequencing experiment. An example of such a matrix is shown in Figure 1c. Due to sequencing noise and mutation calling errors, the observed genotype matrix usually differs from the ground truth. With existing single-cell sequencing technologies, the most prevalent type of noise in I is false negative mutation calls, where a mutation M_p that is present in the ground truth in cell C_i (i.e., $T_{ip} = 1$) is reported as absent in I (i.e., $I_{ip} = 0$). Although the observed matrix I may also contain some false positives, their occurrence is typically rare—with reported rates ranging between 0.0001 and 0.001 [8]. Previous work has shown that such low false positive rates have minimal impact on the accuracy of the reconstructed trees when using methods that explicitly model only false negatives, such as the approach introduced in [18].[2] Given this, and in line with [18], we assume that the input data in this work contains only false negatives. Nevertheless, we also assess the performance of our method under realistic levels of false positives and show that it remains largely robust to this type of noise.

Due to the presence of erroneous mutation calls, there typically does not exist a tree that is consistent with the observed matrix I. More precisely, it is well known that any binary matrix H containing a triplet of rows (i, j, h) and a pair of columns (p, q) such that

$$\begin{bmatrix} H_{ip} & H_{iq} \\ H_{jp} & H_{jq} \\ H_{hp} & H_{hq} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

[2] Alternatively we can handle a small ($\leq c$ for a constant c) number of false positives by solving our problem for every subset of $\leq c$ columns removed from I [16] and returning the solution with the best objective. While a serial implementation of this approach could increase the running time by a factor exponential in c, a trivial parallelization would result in a speedup linear with the number of processors.

is not consistent with any tree under the Perfect Phylogeny Model [15]. We say that such a pair of mutations and triplet of cells forms a *conflict*, and we refer to binary matrices that contain no conflicts as *perfect phylogeny matrices*. It is well known that for every perfect phylogeny matrix there exists a tree consistent with it [9].

Given the presence of conflicts in the observed matrix I, existing methods for reconstructing tumor evolutionary trees aim to correct I by flipping some of its entries to obtain a perfect phylogeny matrix, from which a tree can be reconstructed. These methods approach the problem either explicitly, by directly operating in the space of binary matrices, or implicitly, by searching in the space of trees and scoring them using the perfect phylogeny genotype matrices they imply (see [15] for more details).

This leads us to the formal definition of the *Perfect Phylogeny Reconstruction Problem* that we aim to solve in this work.

▶ **Definition 1** (Perfect Phylogeny Reconstruction Problem)**.** *Given a binary genotype matrix I, determine the minimum set of zero entries in I whose conversion to ones yields a perfect phylogeny matrix.*

For brevity, we use PP to denote *perfect phylogeny matrix* and "most likely PP" to denote the *most likely perfect phylogeny matrix*; i.e., the perfect phylogeny matrix obtained by flipping a minimum number of zero entries in the observed matrix.

The most likely perfect phylogeny problem has been studied before under the name of the i-flip variant of the minimum-flip problem [3]. Here the authors prove NP-completeness and give an $O(2^k mn)$ fixed-parameter tractable (FPT) algorithm for the decision problem "does there exist a perfect phylogeny requiring at most k flips?" However, translating this FPT algorithm to the most likely perfect phylogeny problem would lead to exploring the full, unpruned search tree considered in this work. This would lead to an algorithm orders of magnitude slower and is not feasible at the size of input considered in this work. There are approximate and FPT algorithms for the minimum-flip problem allowing both types of flips [3, 2, 22]. However, these methods allow both one-to-zero and zero-to-one flips with equal weight, which is not well suited for single-cell sequencing data where false positive errors are orders of magnitude rarer than false negative errors. Direct ILP formulations for variants of this minimum flip problem have been studied in [16, 4], but these approaches' scalability is limited by the need to solve large ILPs.

3 Preliminaries

Throughout the remainder of the paper, let H denote an arbitrary binary matrix, which, depending on the context, may represent either the original input matrix I or an intermediate matrix generated during the branch-and-bound procedure. As an intermediary step towards the full perfect phylogeny reconstruction problem, we attempt to get a handle on $\mathrm{OPT}_{\mathrm{init}}(H)$, the minimum number of flips required to remedy all *initial* conflicts in H, disregarding any potential newly introduced conflicts (i.e., all initial conflicts are resolved if for each row triplet with configuration (0,1), (1,0), (1,1), at least one of the (0,1) or (1,0) is flipped to a (1,1)). Note that the details on computing $\mathrm{OPT}_{\mathrm{init}}(H)$ will be provided later (e.g., in Section 4.2). Analogously, let $\mathrm{OPT}(H)$ be the minimum number of flips necessary to make H a PP. Observe that, since a PP obtained by flipping some entries of H must resolve all initial conflicts in H, as well as any conflicts introduced in the process of resolving those initial conflicts, we always have $\mathrm{OPT}_{\mathrm{init}}(H) \leq \mathrm{OPT}(H)$. Next, a binary matrix D is said to be a *descendant* of a binary matrix H if D can be obtained by flipping some zeros in H. Additionally,

define $F_{0\to1}(H, D)$ as the number of flips required to turn H into D and $\mathrm{PP}(H) = \{D : D \text{ is a descendant of } H \text{ and } D \text{ is a PP}\}$. Let $R_{01}(p, q) = \{i : H_{ip} = 0 \wedge H_{iq} = 1\}$; i.e. the set of $(0, 1)$ rows in columns p and q. Similarly, define $R_{10}(p, q) = \{i : H_{ip} = 1 \wedge H_{iq} = 0\}$ and $R_{11}(p, q) = \{i : H_{ip} = 1 \wedge H_{iq} = 1\}$. Note that, strictly speaking, the sets $R_{01}(p, q)$, $R_{10}(p, q)$, and $R_{11}(p, q)$ should also include the matrix H in the notation, but we omit this dependence throughout the paper for notational simplicity.

3.1 Vertex Cover approximation algorithms

Vertex Cover (VC) is a fundamental problem in combinatorial optimization. Given an undirected graph (V, E), where V is the set of vertices and E is the set of edges, the goal of VC is to find a subset $C \subseteq V$ of minimum cardinality such that for every edge $(u, v) \in E$, at least one of the endpoints u or v belongs to C. VC is NP-complete and cannot be solved in polynomial-time unless P = NP. Nevertheless, VC admits efficient 2-approximation algorithms; or 2-approximations, for short. That is, VC admits algorithms that, in polynomial-time, return a vertex cover C with cardinality provably at most twice the cardinality of a minimum vertex cover C^*; i.e., $|C| \leq 2|C^*|$. VC has a very rich history and admits two remarkably simple 2-approximations that we make use of in this work. Interested readers can find detailed treatment of these 2-approximations in any combinatorial optimization reference; e.g., [20].

The first 2-approximation finds a maximal matching M in time $O(|E|)$ by simply iterating over each $(u, v) \in E$ and adding (u, v) to M if both endpoints are unmatched. In the end, it returns $C = \bigcup_{(u,v)\in M}\{u, v\}$. The second VC 2-approximation formulates VC as an LP.

$$\min \quad \sum_{v\in V} x_v \qquad \text{(LP-VC)}$$

$$\text{subj. to} \quad x_u + x_v \geq 1, \qquad \forall (u, v) \in E \qquad (1)$$

$$x_v \geq 0, \qquad \forall v \in V \qquad (2)$$

Next, solutions $x^* \in [0, 1]^{|V|}$ of the above LP are rounded to $X^* \in \{0, 1\}^{|V|}$, where, for each $v \in V$, $X^*_v = 1$ if $x^*_v \geq 1/2$ and $X^*_v = 0$ otherwise. Finally, the algorithm returns the vertex cover $C = \{v \in V : X^*_v = 1\}$. (LP-VC) is well known to be *half-integral*, meaning that there always exists an optimal solution x^* satisfying $x^*_v \in \left\{0, \frac{1}{2}, 1\right\}$ for all $v \in V$.

3.2 Branch-and-bound and PhISCS-BnB

In order to prove optimality of the PP returned by PhISCS-BnB when given an input matrix I, the authors lean on the branch-and-bound paradigm. Branch-and-bound builds a search tree where each node corresponds to some binary matrix H, so for brevity we refer to search tree nodes and matrices interchangeably. The tree's root node is the input matrix I, its leaves are precisely PPs, and, for any D in the subtree rooted at H, D is a descendant of H. An internal node H' is not a PP and has parent node H. Here, H' is obtained by flipping zeros to fix all conflicts in H between some pair of columns p and q. In an optimal solution, conflicts between columns p and q must be fixed by either flipping all the zeros in column p and rows $R_{01}(p, q)$ or all the zeros in column q and rows $R_{10}(p, q)$.

By exploring this tree, branch-and-bound either explores or proves suboptimality for all PPs in $\mathrm{PP}(I)$. Given a matrix H, a (possibly suboptimal) PP B, and a lower bound $L(H)$ on the required number of flips to make H a PP, then all matrices D that are descendants of H can be ruled out (i.e., "pruned") if $F_{0\to1}(I, H) + L(H) \geq F_{0\to1}(I, B)$. In this event,

no PP D, which is a descendant of H, can beat B and therefore may be pruned out of the search tree. When an explored leaf beats B, it becomes the new best node and the value of B is updated; [18] also describes simple heuristics for finding an initial guess B. In this way, PhISCS-BnB takes a binary matrix H with conflicts, gradually introduces flips to explore a tree of all possibly optimal descendants while pruning nodes that can be ruled out, and, finally, terminates when it has found and certified a most likely PP.

Branch-and-bound algorithm Our branch-and-bound algorithm, provided in detail in Algorithm 1, builds on top of PhISCS-BnB. The key differences are the use of our new bounding algorithms (Section 4.2) and an additional step in which we check whether the "free" byproduct PPs produced by our algorithms improve upon the current incumbent PP, updating it when they do (Section 4.6).

3.2.1 Lower bounding mechanisms

PhISCS-BnB explores two formulations for its lower bounding mechanism by solving related instances of i) *minimum weighted SAT* (MWS) and ii) *maximum weight matching* (MWM). Despite MWS itself being NP-hard, and thus requiring super-polynomial time in the worst case, in practice, PhISCS-BnB with MWS terminates orders of magnitude faster than PhISCS-BnB with MWM.

MWM lower bounding mechanism. Given a matrix H, the MWM lower bounding mechanism constructs a graph (V, E) where vertices correspond to columns of H and edges correspond to column pairs with conflicts. Formally, MWM sets up a graph (V, E) with $V = \{p : 1 \leq p \leq m\}$ and $E = \{(p, q) : |R_{01}(p, q)| \geq 1 \wedge |R_{10}(p, q)| \geq 1 \wedge |R_{11}(p, q)| \geq 1\}$. Observe that, for any pair of columns p and q of matrix H, a conflict is implied by the presence of all three value pairs: $(0, 1)$, $(1, 0)$, and $(1, 1)$, when considering the values of these columns across all rows of H. Since we are only allowed to make flips from 0 to 1, resolving all conflicts between columns p and q requires flipping all 0's in column p to 1 in each row of $R_{01}(p, q)$ *or* flipping all 0's in column q and each row of $R_{10}(p, q)$. In other words, we need at least the smaller of the two numbers of flips in order to resolve all conflicts between columns p and q. Hence, for each $e = (p, q) \in E$ the edge weight w_e is set to $\min\{|R_{01}(p, q)|, |R_{10}(p, q)|\}$ when $|R_{11}(p, q)| > 0$ and $w_e = 0$ otherwise. The matching property ensures that flips that may fix multiple conflicts are not counted multiple times. Let $\text{MWM}(H)$ be the weight of such a MWM in the corresponding (V, E). Indeed, the weight of any matching M on (V, E) is a lower bound on $\text{OPT}_{\text{init}}(H)$ since w_e for $e = (p, q)$ is a lower bound on the number of flips from columns p or q that a PP must make to resolve all conflicts between p and q. Therefore, $\text{MWM}(H)$ is the best matching-based lower bound.

MWS lower bounding mechanism. Given a matrix H, the *minimum weighted SAT* (MWS) lower bounding mechanism writes a weighted SAT instance wherein the weight of a minimum satisfying assignment lower bounds OPT. PhISCS-BnB considers other MWS formulations. For brevity's sake, we refer interested readers to [18]. Like VC, MWS is NP-complete. However, unlike our algorithms, which solve VC approximately and in polynomial time, PhISCS-BnB solves MWS instances exactly in worst case super-polynomial time.

3.3 Outline of our contributions

We make key observations relating the perfect phylogeny reconstruction problem to vertex cover and use these insights to derive branch-and-bound lower bounding mechanisms inspired by well-known 2-approximation algorithms of vertex cover. In Section 4, we show that our new polynomial-time lower bounding mechanisms achieve provably stronger guarantees than the polynomial-time lower bounding mechanisms of PhISCS-BnB.

In Section 5, we benchmark our algorithm on synthetic datasets. Our experiments demonstrate that the proposed polynomial-time lower bounding mechanisms significantly accelerate branch-and-bound search, yielding speedups of up to 1000x over PhISCS-BnB.

While our best performing lower bounding algorithms can only account for initial conflicts, in Section 4.4, we introduce an additional lower bounding mechanism that gives a provably stronger lower bound that accounts for conflicts beyond only those that are initially present.

4 Lower bounding algorithms

4.1 Provably better polynomial-time lower bounds

To further motivate the search for better polynomial-time lower bound mechanisms, we take a closer look at MWM. Let $\text{MWM}(H)$ be the maximum weight matching computed on the graph corresponding to matrix H. Let $S_{n,m}$ be the $n \times m$ binary matrix consisting of $n-2$ copies of the row $(0,1,1,\ldots,1)$, one $(1,0,0,\ldots,0)$ row, and one $(1,1,\ldots,1)$ row; i.e.,

$$S_{n,m} = \begin{bmatrix} 0 & 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & 1 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1 \end{bmatrix}.$$

Observe that all conflicts in $S_{n,m}$ occur between column 1 and another column $p > 1$. Therefore, the corresponding MWM graph of $S_{n,m}$ is a star graph with center vertex corresponding to the first column. Similarly, because each conflict can be resolved via a single flip in the second to last row, all edge weights are 1. Hence, by these two facts, $\text{MWM}(S_{n,m}) = 1$. However, the optimal solution uses $\min\{n-2, m-1\}$ flips, indicating flipping all zeros in either the first column or the second to last row, respectively.

▶ Remark 2. For all $k \geq 3$, $\text{MWM}(S_{k,k+2}) = 1$ despite $\text{OPT}_{\text{init}}(S_{k,k+2}) = k-2$.

Even as OPT_{init} increases, MWM lower bounds may provide no information at all. A lower bound of 1 is merely an unhelpful "the current matrix is not a PP" message.

4.2 Vertex Cover formulation and lower bounding algorithms

Our algorithms take a binary matrix H and use the VC approximations described in Section 3. Let $Z(H)$ be a set of coordinates for pairs of zeros appearing in conflicts together; i.e.

$$Z(H) = \{((i,p),(j,q)) : (1 \leq p < q \leq m) \wedge (i \in R_{01}(p,q)) \wedge (j \in R_{10}(p,q)) \wedge (|R_{11}(p,q)| > 0)\}.$$

Define the graph (V_H, E_H) with $V_H = \{(i,p) \mid H_{ip} = 0\}$ and $E_H = Z(H)$; for comparison, H has nm entries, $|V| \leq nm$, and $|E| = O(n^2m^2)$. Algorithm 5 (provided in the Appendix) directly uses the maximal matching-based approximation on (V_H, E_H). Similarly, Algorithm 3, uses the LP-based approximation where LP-VC is written taking $(V,E) = (V_H, E_H)$.

Input: Original input matrix $I \in \{0,1\}^{n \times m}$
$B \leftarrow$ A simple PP (see Section 4.7) ; `// Best incumbent node`
$Q \leftarrow$ An empty priority queue;
Push I into Q;
while Q *is not empty* **do**
  $C \leftarrow$ pop next node from Q with minimum priority score;
  $p, q \leftarrow$ columns of C in conflict;
  `// Branch into two new nodes,` $S^{(1)}$ `and` $S^{(2)}$`, by fixing the` pq `conflict by flipping either` $(0,1)$`s or` $(1,0)$`s`
  $S^{(1)} \leftarrow C$, $S^{(2)} \leftarrow C$;
  for $r \in R_{01}(p,q)$ **do**
    $S^{(1)}_{r,p} \leftarrow 1$;
  end
  for $r \in R_{10}(p,q)$ **do**
    $S^{(2)}_{r,q} \leftarrow 1$;
  end
  for $i \in \{1,2\}$ **do**
    if $S^{(i)}$ *is a PP* **then**
      if $F_{0\to1}\left(I, S^{(i)}\right) < F_{0\to1}(I,B)$ **then**
        `//` $S^{(i)}$ `beats incumbent best node` B
        $B \leftarrow S^{(i)}$;
      end
    else
      `// Explore nodes that cannot be pruned away`
      $L \leftarrow$ lower bound on the number of flips to make $S^{(i)}$ a PP; e.g. from Section 4.2;
      if $L + F_{0\to1}\left(I, S^{(i)}\right) < F_{0\to1}(I,B)$ **then**
        Push $S^{(i)}$ in Q with priority score set to $L + F_{0\to1}\left(I, S^{(i)}\right)$;
      end
      `// Use "free" upper bound, if available, to update best incumbent node` B`; see Section 4.6`
      if *Lower bounding algorithm gives a "free" incumbent best node* **then**
        $B' \leftarrow$ candidate best node byproduct;
        if $F_{0\to1}(I,B') < F_{0\to1}(I,B)$ **then**
          $B \leftarrow B'$;
        end
      end
    end
  end
end
$Y \leftarrow B$;
return Y;

Algorithm 1 VC-PhISCS-BnB: Branch and bound method

315 ▶ **Lemma 3.** *$C \subseteq V_H$ is a VC of (V_H, E_H) if, and only if, flipping H_{ip} for all $(i,p) \in C$*
316 *fixes all initial conflicts of H.*

317 **Proof.** Observe that: a conflict between (i,p) and (j,q) in $Z(H)$ is fixed $\iff$ $H_{i,p}$ or $H_{j,q}$

is flipped $\iff$ $(i,p) \in C$ or $(j,q) \in C$ $\iff$ $((i,p),(j,q)) \in E$ is covered. As $Z(H)$ and E_H are in a one-to-one correspondence, the claim follows. ◀

▶ **Corollary 4.** *Let C^* be a minimum VC of (V_H, E_H). Then $|C^*| = \mathrm{OPT}_{\mathrm{init}}(H)$.*

Behind effective pruning algorithms, lie effective "lower bounding" algorithms. That is, given a matrix H, we need algorithms that produce strong lower bounds on the number of flips necessary to make H a PP. As seen in Remark 2, the MWM bound is not useful in the worst case. Unlike MWM, our lower bounding algorithms remain provably useful.

▶ **Theorem 5.** *Let L_{VC} and $L_{LP\text{-}VC}$ be the lower bound values returned by Algorithms 3 and 5, respectively, on a binary matrix H. Then $\mathrm{OPT}_{\mathrm{init}}(H)/2 \leq L_{VC}, L_{LP\text{-}VC} \leq \mathrm{OPT}_{\mathrm{init}}(H)$.*

Proof. $L_{\mathrm{VC}} = \lceil |C|/2 \rceil$ where $|C|$ is a VC of (V_H, E_H), $|C| \leq 2|C^*|$ for a minimum VC C^*. Applying Corollary 4 and the definition of 2-approximation,

$$\mathrm{OPT}_{\mathrm{init}}(H) = |C^*| \leq |C| \leq 2|C^*| = 2\,\mathrm{OPT}_{\mathrm{init}}(H).$$

Thus, dividing by two, $\mathrm{OPT}_{\mathrm{init}}(H)/2 \leq |C|/2 \leq \mathrm{OPT}_{\mathrm{init}}$. Since $\mathrm{OPT}_{\mathrm{init}}(H)$ is an integer, we can take a ceiling and complete the proof for the claim about L_{VC}.

For the next lower bound, $L_{\text{LP-VC}} = \left\lceil \sum_{(i,p)\in V_H} x^*_{ip} \right\rceil$ with x^* and X^* being the fractional and rounded LP solutions, respectively.

$$\sum_{i,p} x^*_{ip} \leq |C^*| = \mathrm{OPT}_{\mathrm{init}}(H) \leq \sum_{i,p} X^*_{ip} \leq 2 \sum_{i,p} x^*_{ip}.$$

The first inequality follows from the well known fact that $L_{\text{LP-VC}} \leq |C^*|$, the equality follows from Corollary 4, the second inequality follows from the fact that $\{(i,p) : X^*_{ip} = 1\} \subseteq V_H$ defines a VC, and, finally, the last inequality follows from the fact that $X^*_{ip} \leq 2x^*_{ip}$ for all $(i,p) \in V_H$. Like before, $\mathrm{OPT}_{\mathrm{init}}(H)$ is an integer, therefore we can take a ceiling and both bounds on $L_{\text{LP-VC}}$ follow. ◀

Between Algorithms 3 and 5, the fastest (and simplest) approach is arguably the maximal matching-based Algorithm 5. Since both algorithms rely on VC 2-approximations, why not stick to the simpler one? Computation time is only one aspect of an effective lower bounding algorithm; quality of the lower bound matters too. For the simple VC approach, the lower bound, $|C|/2$, and the VC size, $|C| \leq 2\,\mathrm{OPT}_{\mathrm{init}}$, are, pessimistically, always a factor of 2 apart. Meanwhile, for (LP-VC), instead of the analogous lower bound $\frac{1}{2}\sum_{i,p} X^*_{ip} = |C|/2$, we can use a greater $\sum_{i,p} x^*_{ip}$.

▶ Remark 6. $\sum_{i,p} x^*_{ip} - \frac{1}{2}\sum_{i,p} X^*_{ip} = \frac{N}{2}$ where $N = |\{(i,p) : x^*_{ip} = 1\}|$.

Proof. By the half-integrality of (LP-VC) we can split the summation as

$$\sum_{i,p;x^*_{ip}\in\left\{0,\frac{1}{2}\right\}} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) + \sum_{i,p;x^*_{ip}=1} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) = \sum_{i,p;x^*_{ip}=1} \left(x^*_{ip} - \frac{1}{2}X^*_{ip}\right) = \frac{N}{2}.$$

◀

In practice, a substantial $N/2$ enables Algorithm 3 to prune the search tree more aggressively and may even offset the additional computational cost of writing and solving an LP. This tradeoff between lower bound quality and computation speed is explored once more in the last of our lower bounding algorithms, Algorithm 4.

Input: H is the matrix at the current branch-and-bound node, F is a set of flips resolving all initial conflicts on H, $H \oplus F$ returns H after applying the flips of F.
function *CandidateBestNode*(H, F)
  $B \leftarrow$ Node requiring ∞ flips to become a PP;
  if $H \oplus F$ *is a PP* **then**
    $B \leftarrow H \oplus F$;
  end
  return B
end

Algorithm 2 Check if a set of flips makes the matrix a PP

Input: H is the matrix at the current branch-and-bound node and `solver` is an LP solver.
function *LPLowerBound*$(H,$ *solver*$)$
  $(V_H, E_H) \leftarrow$ VC instance from H as described in Section 4.2;
  $\Pi \leftarrow$ (LP-VC) instance written using $(V, E) = (V_H, E_H)$;
  $x^* \leftarrow$ optimal solution to Π, obtained using `solver`;
  for $H_{ip} = 0$ **do**
    $X^*_{ip} \leftarrow 1$ if $x^*_{ip} \geq \frac{1}{2}$ else 0;
  end
  $C \leftarrow \{(i,p) : X^*_{ip} = 1\}$;
  return $\left\lceil \sum_{i,p} x^*_{ip} \right\rceil$, *CandidateBestNode*$(H, C)$;
end

Algorithm 3 Lower bound using (LP-VC)

4.3 Considering only the initial conflicts in I

We now focus on Algorithm 4, stated fully in the Appendix, which is also an LP-based approach but deals with (LP-VC) defined by $(V, E) = (V_H, E_H \cap E_I)$ where I is the initial input matrix. Like before, $E_I = Z(I)$, so $E_I \cap E_H$ is a set of edges corresponding to conflicts that are present in *both* I and H. The compromise between computational speed and lower bound quality is as follows. We write the full (LP-VC) once with the graph (V_I, E_I). Then, at each node of the branch-and-bound tree, we only *partially* rewrite (LP-VC). That is, at each node, instead of rewriting the costly $O(m^2n^2)$ LP constraints, we just update the lower bounds of $O(mn)$ variables to ensure that the entries that have been flipped between I and H remain flipped. This results in a significant speedup, as $mn \ll (mn)^2$; owing to the high quality of LP solvers available, writing the LP itself can be much more time consuming than solving it. On the other hand, this approach weakens our lower bound as it will not account for conflicts introduced in the flips from I to H. An analogous guarantee to Theorem 5 holds for Algorithm 4.

▶ Remark 7. The lower bound of Algorithm 4 is at least half the number of flips necessary to fix conflicts present in both I and H.

4.4 Going beyond initial conflicts

The lower bounds on the number of flips we prove in Theorem 5 hits a ceiling at initial conflicts. That is, for any binary matrix H and lower bound $L(H)$ on the number of

flips necessary to make H a PP from Theorem 5, $L(H) \leq \mathrm{OPT}_{\mathrm{init}}(H)$. In this section we introduce a new LP formulation giving stronger lower bounds that can exceed $\mathrm{OPT}_{\mathrm{init}}(H)$. Let $\mathcal{I} := \{(0,1),(1,0),(1,1)\}$ and define our LP as follows.

$$\min \sum_{i,p:H_{ip}=0} x_{ip} \qquad \text{(LP-Extended)}$$

$$\text{subj. to} \quad -x_{ip} + x_{iq} \leq B(p,q,0,1), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (3)$$

$$x_{ip} - x_{iq} \leq B(p,q,1,0), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (4)$$

$$x_{ip} + x_{iq} \leq 1 + B(p,q,1,1), \qquad \forall 1 \leq i \leq n,\ 1 \leq p < q \leq m \qquad (5)$$

$$\sum_{(a,b)\in\mathcal{I}} B(p,q,a,b) \leq 2, \qquad \forall 1 \leq p < q \leq m \qquad (6)$$

$$0 \leq B(p,q,a,b) \leq 1, \qquad \forall 1 \leq p < q \leq m,\ (a,b) \in \mathcal{I} \qquad (7)$$

$$0 \leq x_{ip} \leq 1, \qquad \forall H_{ip} = 0 \qquad (8)$$

This LP is reminiscent of the integer program in [16], which is used for a closely related problem. Like in (LP-VC), the variable x_{ip} corresponds to the matrix entry H_{ip}, with $x_{ip} = 1$ denoting that H_{ip} should be flipped and $x_{ip} = 0$ saying to leave H_{ip} unflipped. The auxiliary variables $B(p,q,a,b)$ ensure, for each $(a,b) \in \mathcal{I}$, that if columns p and q have a row with values (a,b), then $B(p,q,a,b) = 1$. Lastly, Constraints (6) ensure an integer solution corresponds to a matrix that is a PP.

Given a matrix H, let $L_1 = \sum_{i,p} x^*_{ip}$ and $L_2 = \sum_{i,p} y^*_{ip}$ be LP objectives at optimal solutions of (LP-VC) and (LP-Extended), respectively.

▶ **Lemma 8.** *$L_1(H) \leq L_2(H)$.*

▶ **Lemma 9.** *$L_2(H) \leq \mathrm{OPT}(H)$.*

Proof. Let H^* be a most likely PP. It suffices to show that x^* with $x^*_{ip} = H^*_{ip} \in \{0,1\}$ is in the feasible region of (LP-Extended). As H^* is a PP, for any pair of columns p and q, the columns p and q do not contain (at least) one of the three rows $(0,1)$, $(1,0)$, or $(1,1)$; otherwise H^* would not be a PP. This means for every pair p and q, there exists an assignment of $B(p,q,a,b)$, for $(a,b) \in \mathcal{I}$ such that Constraints (3), (4), (5), and (6) are all satisfied. Therefore, x^* is in the feasible region of (LP-Extended). ◀

▶ **Lemma 10.** *There exist binary matrices U where $L_2(U) > \mathrm{OPT}_{\mathrm{init}}(U) \geq L_1(U)$.*

Note that proofs of Lemma 8 and Lemma 10 can be found in the Appendix. Finally, the following theorem follows from Lemmas 8–10.

▶ **Theorem 11.** *$L_1(H) \leq L_2(H) \leq \mathrm{OPT}(H)$. Moreover, there exist binary matrices U for which $L_2(U) > \mathrm{OPT}_{\mathrm{init}}(U)$.*

On the other hand, this LP does not always perform better. Let $I_{n,n}$ be the $n \times n$ identity matrix and let $\mathbf{1}$ be the matrix of size $n \times n$ with all of its entries equal to 1. The matrix $\mathbf{1} - I_{n,n}$ will have the associated VC instance be a complete graph on n vertices. Therefore, $\mathrm{OPT} = n - 1$ but both LP objectives will be $\mathrm{OPT}/2$.

4.5 Trading off between LP formulation and speed

We identify that rewriting an LP to account for the conflicts present in every node of the BnB search tree is a bottleneck. To this end, in Section 4.3 we consider only the initial

conflicts of I in its computed lower bound. At each node of the search tree we only need to adjust the bounds of $O(mn)$ variables, instead of rewriting the much larger LP. However, H may have introduced new conflicts. To consider these conflicts without needing to also rewrite the LP from scratch, we use (LP-Extended) in Section 4.4.

Alongside Algorithm 4, this lets us consider all initial conflicts at every matrix H, while avoiding the need to fully rewrite the LP at every node of the search tree. At the center of this tradeoff lies a question whose answer depends on the given input matrix I: "what fraction of the conflicts that must be resolved are present initially?" If few or no conflicts are to be re-introduced, then the larger LP is not worth the additional computation. However, if conflicts are introduced frequently enough, then the additional computation of (LP-Extended) can be worth it.

4.6 Upper bounding algorithms at low computational cost

There is one more ingredient to effectively pruning nodes: providing a good "best node" B guess (i.e. a possibly suboptimal PP), which promises the optimal solution can only take *at most* $F_{0\to 1}(I, B)$. In this way each PP B gives an *upper bound* on the number of required flips. In addition to lower bounds, our algorithms also use the byproduct VCs to check whether they lead to a new best node. In particular, if the flips suggested by the VC introduce no new conflicts, then the resulting matrix constitutes a PP, and we check whether this new PP improves upon the best one seen so far. This logic for updating the best-so-far node lies in Algorithm 2; checking whether a matrix is a PP is done in $O(nm)$ time using the algorithm of [9].

4.7 Hybrid algorithms

In many synthetic matrices, the MWS lower and upper bounding heuristic of PhISCS-BnB is able to terminate at the root node of the search tree. In other words, the lower bound produced at the root node matches the best node.

To take advantage of this, we build a hybrid branch-and-bound algorithm by borrowing this strategy only for the first best node guess. Then we use our own algorithms from Section 4 for the remainder of the branch-and-bound algorithm. Analogous to Algorithm 4, this is yet another algorithmic compromise between quality and speed of lower bounds that performs promisingly.

Polynomial time can be guaranteed for the initial guess by running both the MWS formulation and our polynomial-time algorithms in parallel and taking the better guess of the two after some (polynomial) time. As all of the lower bounding algorithms suggest a set of flips that solve initial conflicts, we can simply run the lower bounding algorithm, perform the suggested flips, and repeat until we achieve a PP. This is guaranteed to converge in polynomial-time because if the current matrix has any conflicts, then some nonzero number of flips will be suggested and the matrix of all ones is a PP.

5 Results

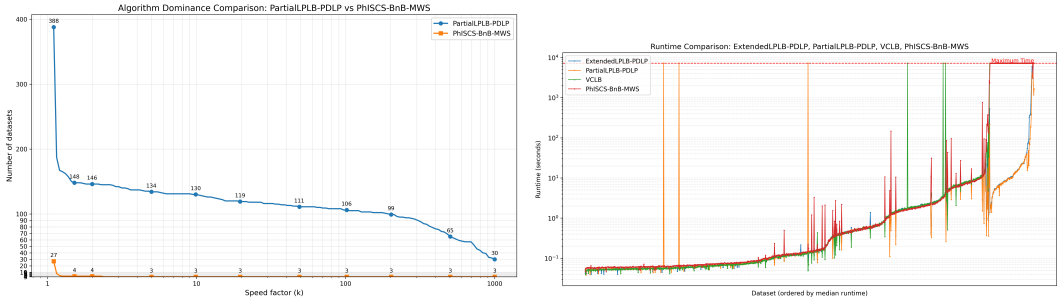
5.1 Experimental setup

All experiments presented in this section were run using Python 3.10 on an 8-core machine with an Intel Xeon Silver 4216 CPU and 64GB of RAM. The branch-and-bound logic is implemented using the library `pybnb`. Our code repository forks the code of [18] and is available at `https://github.com/jdluque/faster-tphyl-reconstruction`. To

implement the various lower bounding mechanisms we use the Python libraries `Cython`, `numpy`, `NetworkX`, and `pysat`. LPs are solved using Google's PDLP [1] through their open sourced `or-tools` package as well as Gurobi. We distinguish which lower bounding algorithm was used to branch-and-bound in the following way. LPLowerBound(Solver) uses Algorithm 3 and PartialLPLowerBound(PDLP) uses Algorithm 4, each using one of PDLP or Gurobi as the LP solver. ExtendedLPLB(PDLP) denotes Algorithm 4 using (LP-Extended) and PDLP as the underlying LP solver. VCLB uses Algorithm 5 and is run with $r = 5$ repetitions. All of our presented algorithms are the hybrid variants as discussed in Section 4.7. PhISCS-BnB using MWS and MWM as the underlying lower bounding algorithm are denoted by PhISCS-BnB-MWS and PhISCS-BnB-MWM, respectively.

To assess the performance of our algorithm, we simulated phylogenetic trees representing tumor evolution with a specified number of cells $n \in \{20, 30, 50, 100\}$, each labeling a distinct leaf of the simulated tree (see Figure 1b). We then randomly assigned a given number of mutations $m \in \{20, 30, 50, 100, 250, 500, 1000\}$ to the edges of the tree. This tree gives us a unique ground truth genotype matrix with $400 \leq mn \leq 100,000$ entries. To generate the noisy (observed) genotype matrix I used as input to our method, we introduced noise into the ground truth matrix as follows: for each entry equal to 1, we flipped it to 0 with probability $p \in \{0.05, 0.10, 0.15, 0.20\}$ (i.e., p denotes the false negative rate). For each combination of parameters (m, n, p) we generated 10 distinct instances.

In addition to the simulated datasets described above, we also evaluated our methods on the same mouse melanoma dataset used in PhISCS-BnB, derived from the study of [24] and available through Python's `scphylo-tools` library.



(a) Runtime comparison of PartialLPLB(PDLP) and PhISCS-BnB-MWS.

(b) Runtime comparison extended to include VCLB and ExtendedLPLB-PDLP.

Figure 2 **(a)** The runtime of each algorithm for the specified dataset on a log scale with different matrices at each value of the horizontal axis. Matrices are sorted by the median time to find the most likely PP between the competing algorithms. **(b)** Additionally considers algorithms VCLB and ExtendedLPLB-PDLP. The lines that go all the way up to the timeout line indicate a dataset that the algorithm failed to complete.

5.2 Experimental results

Each algorithm is allotted 48 hours total to work through the 1,120 generated matrices. Further, each algorithm is allowed a maximum of 2 hours per matrix before timing out. For the sake of clearer plots, we rank instances by the median time required for the various algorithms to compute a most likely PP.

A natural comparison benchmark is then to ask how many PP each algorithm finds within time constraints. This result is displayed in Table 1 with our LP-based methods being the clear winners, finding 98.8% of PPs in the largest matrices with $mn = 100,000$ entries. These are followed by PhISCS-BnB-MWS and VCLB, which find 89.5% and 89.0% of PPs

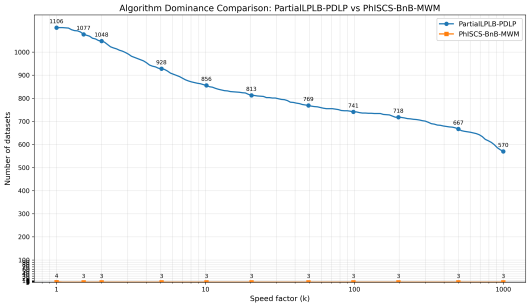


Figure 3 Comparison of speedup factors of PartialLPLB(PDLP) and PhISCS-BnB-MWM shows that our new method is a vast improvement over the previous best polynomial-time lower bound algorithm.

on matrices with as many as $mn = 50,000$ entries. Finally, all algorithms greatly outperform PhISCS-BnB-MWM which only finds PPs for 39.7% of matrices including matrices with up to $mn = 3,000$ entries.

We plot the speedup factor k, as well as the number of datasets for which this speedup factor is achieved (i.e., the number of matrices in which one algorithm finishes k times faster than the other).

Bringing up the rear, PhISCS-BnB-MWM finds PPs for 445 matrices with up to $nm = 3000$ entries. Next, VCLB and PhISCS-BnB-MWS, find PPs for 997 and 1002 matrices with up to $mn = 50000$ entries, respectively. Leading the pack, PartialLPLB(PDLP), ExtendedLPLB(Gurobi), ExtendedLPLB(PDLP) all find PPs for 1107 matrices with $mn = 100,000$ entries.

Table 1 Performance of various methods in finding PPs within time constraints.

Method	PPs found	Max Entries (mn)
ExtendedLPLB(PDLP)	**1107** (=98.8%)	**100,000**
PartialLPLB(PDLP)	**1107** (=98.8%)	**100,000**
PartialLPLB(Gurobi)	**1107** (=98.8%)	**100,000**
PhISCS-BnB-MWS	1002 (=89.5%)	50,000
VCLB	997 (=89.0%)	50,000
PhISCS-BnB-MWM	445 (=39.7%)	3,000

Our greatest improvements over PhISCS-BnB-MWS were generally achieved by PartialLPLB(PDLP) and ExtendedLPLB(PDLP), as shown in Figures 2 and 4 (note that Figure 4 is provided in the Appendix). Intuitively, it is not always optimal to disregard conflicts that are introduced later on, as PartialLPLB(PDLP) does. Still, this proves to be a valuable variant, as PartialLPLB(PDLP) outperforms ExtendedLPLB(PDLP) in many instances, as shown in Figure 4. On the other hand, ExtendedLPLB(PDLP) performs better on some of the harder instances, so there is no conclusive winner. Across the board, no algorithm outperformed every other algorithm on every dataset, as shown in Figure 2b. Relative to PhISCS-BnB-MWS, ExtendedLPLB(PDLP) achieves speedup factors as large as 1000x on 27 instances. The improvements are much steeper compared with the previous polynomial-time lower bounds of PhISCS-BnB-MWM. In Figure 3, PartialLPLB(PDLP) is orders of magnitude faster than PhISCS-BnB-MWM across nearly all instances.

As suggested by the number of solved instances in Table 1, VCLB also consistently outperforms PhISCS-BnB-MWM. Compared to PhISCS-BnB-MWS, VCLB has an edge on the easier instances, but does not keep up; as the difficulty increases, it times out more often making its overall performance slightly worse than PhISCS-BnB-MWS. Although VCLB is simpler and the bounding algorithm itself is pretty fast, this speedup does not offset the time cost of exploring more nodes of the search tree as a result of the lower bound's deterioration. This mirrors the discussion at the end of Section 4.2. Interestingly, PartialLPLB with PDLP and Gurobi perform comparably, with PDLP completing a few more instances faster, as shown in Figure 5. Lastly, we benchmarked our methods on the mouse melanoma dataset introduced above. On this dataset, our algorithms and PhISCS-BnB-MWS both reconstruct the most likely PP in 80 minutes.

5.3 Tree reconstruction accuracy in the presence of false positives

We also evaluated the performance of our method in the presence of false positive mutation calls in the input data. To this end, we conducted additional experiments in which, alongside false negatives, we also simulated false positives. Given that false positives typically occur at lower rates in practice [8], we simulated them at rates of 0.01, 0.001, and 0.0001.

As shown in Figure 6, PartialLPLB(PDLP) remains largely robust to the presence of false positives, with ancestor–descendant accuracy decreasing only slightly relative to the corresponding experiments without false positives. More specifically, the average ancestor–descendant accuracy at false positive rates of 0.01, 0.001, and 0.0001 were 0.984, 0.975, and 0.9240, respectively, corresponding to decreases between 0.01 and 0.06 compared to the baseline without false positives. By contrast, the reduction in different-lineage accuracy was even smaller, with decreases of less than 0.0001 relative to the baseline without false positives across all evaluated false positive rates. Since the presence of false positives resulted in only negligible changes in different-lineage accuracy, we do not show separate plots for these results.

Overall, these results demonstrate that our method is only marginally affected by the presence of false positives and remains highly accurate under realistic levels of such noise.

6 Discussion and future work

Although the latter lower bound derived in (LP-Extended) is stronger than that of (LP-VC), experiments showed that the lower bound from (LP-VC) solved the perfect phylogeny reconstruction problem faster. In fact, (LP-Extended) is asymptotically smaller using the same number of variables but $O(nm^2)$ constraints compared to the $O(n^2m^2)$ constraints of (LP-VC). This finding is yet another example of an important recurring theme in this work: theory versus practice; or better yet, theory-aided practice.

Nevertheless, our theory-informed algorithms saw speedups upwards of two orders of magnitude in experiments. Stronger theoretical results, such as the bounds of (LP-Extended) in Section 4.4, will likely continue to be an important part of future improvements. This bound may also see more success with even larger datasets or in settings where the SCS data noise is more adversarial; for example, noise with correlations instead of uniformly random.

Our work is a practical advancement for the most likely perfect phylogeny reconstruction problem and sets up groundwork for future improvements by providing the first polynomial-time pruning algorithms with theoretical guarantees. Additionally, we leave room to improve our algorithm via the, still unexploited, polynomial-time pruning algorithms that account for more conflicts than just those initially present in a matrix, as well as with practical improvements in LP solvers such as [13].

References

1. David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *Mathematical Programming*, 201(1-2):133–184, September 2023. URL: https://link.springer.com/10.1007/s10107-022-01901-9, doi:10.1007/s10107-022-01901-9.
2. Sebastian Böcker, Quang Bao Anh Bui, and Anke Truss. Improved Fixed-Parameter Algorithms for Minimum-Flip Consensus Trees. *ACM Trans. Algorithms*, 8(1):7:1–7:17, January 2012. doi:10.1145/2071379.2071386.
3. Duhong Chen, O. Eulenstein, D. Fernandez-Baca, and M. Sanderson. Minimum-flip supertrees: complexity and algorithms. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3(2):165–173, April 2006. URL: https://ieeexplore.ieee.org/document/1631997, doi:10.1109/TCBB.2006.26.
4. Markus Chimani, Sven Rahmann, and Sebastian Böcker. Exact ILP solutions for phylogenetic minimum flip problems. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pages 147–153, Niagara Falls New York, August 2010. ACM. URL: https://dl.acm.org/doi/10.1145/1854776.1854800, doi:10.1145/1854776.1854800.
5. Simone Ciccolella, Mauricio Soto Gomez, Murray D Patterson, Gianluca Della Vedova, Iman Hajirasouliha, and Paola Bonizzoni. gpps: an ilp-based approach for inferring cancer progression with mutation losses from single cell data. *BMC bioinformatics*, 21:1–16, 2020.
6. Mohammadamin Edrisi, Hamim Zafar, and Luay Nakhleh. A combinatorial approach for single-cell variant detection via phylogenetic inference. In *19th International Workshop on Algorithms in Bioinformatics (WABI 2019)*, pages 22–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.
7. Mohammed El-Kebir. Sphyr: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018.
8. Yusi Fu, Chunmei Li, Sijia Lu, Wenxiong Zhou, Fuchou Tang, X Sunney Xie, and Yanyi Huang. Uniform and accurate single-cell sequencing based on emulsion whole-genome amplification. *Proceedings of the National Academy of Sciences*, 112(38):11923–11928, 2015.
9. Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230210104. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230210104, doi:10.1002/net.3230210104.
10. Katharina Jahn, Jack Kuipers, and Niko Beerenwinkel. Tree inference for single-cell data. *Genome Biology*, 17(1), May 2016. doi:10.1186/s13059-016-0936-x.
11. Jack Kuipers, Katharina Jahn, and Niko Beerenwinkel. Advances in understanding tumour evolution through single-cell sequencing. *Biochimica et Biophysica Acta (BBA)-Reviews on Cancer*, 1867(2):127–138, 2017.
12. Can Kızılkale, Farid Rashidi Mehrabadi, Erfan Sadeqi Azer, Eva Pérez-Guijarro, Kerrie L. Marie, Maxwell P. Lee, Chi-Ping Day, Glenn Merlino, Funda Ergün, Aydın Buluç, S. Cenk Sahinalp, and Salem Malikić. Fast intratumor heterogeneity inference from single-cell sequencing data. *Nature Computational Science*, 2(9):577–583, September 2022. Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/s43588-022-00298-x, doi:10.1038/s43588-022-00298-x.
13. Haihao Lu and Jinwen Yang. cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia, June 2024. arXiv:2311.12180 [math]. URL: http://arxiv.org/abs/2311.12180, doi:10.48550/arXiv.2311.12180.
14. Salem Malikic, Katharina Jahn, Jack Kuipers, S. Cenk Sahinalp, and Niko Beerenwinkel. Integrative inference of subclonal tumour evolution from single-cell and bulk sequencing data. *Nature Communications*, 10(1):2750, June 2019. Publisher: Nature Publishing Group. URL: https://www.nature.com/articles/s41467-019-10737-5, doi:10.1038/s41467-019-10737-5.

15 Salem Malikić, Farid Rashidi Mehrabadi, Erfan Sadeqi Azer, Mohammad Haghir Ebrahimabadi, and Suleyman Cenk Sahinalp. Studying the history of tumor evolution from single-cell sequencing data by exploring the space of binary matrices. *Journal of Computational Biology*, 28(9):857–879, 2021.

16 Salem Malikic, Farid Rashidi Mehrabadi, Simone Ciccolella, Md Khaledur Rahman, Camir Ricketts, Ehsan Haghshenas, Daniel Seidman, Faraz Hach, Iman Hajirasouliha, and S. Cenk Sahinalp. PhISCS: a combinatorial approach for subperfect tumor phylogeny reconstruction via integrative use of single-cell and bulk sequencing data. *Genome Research*, 29(11):1860–1877, November 2019. doi:10.1101/gr.234435.118.

17 Edith M Ross and Florian Markowetz. Onconem: inferring tumor evolution from single-cell sequencing data. *Genome biology*, 17:1–14, 2016.

18 Erfan Sadeqi Azer, Farid Rashidi Mehrabadi, Salem Malikić, Xuan Cindy Li, Osnat Bartok, Kevin Litchfield, Ronen Levy, Yardena Samuels, Alejandro A. Schäffer, E. Michael Gertz, Chi-Ping Day, Eva Pérez-Guijarro, Kerrie Marie, Maxwell P. Lee, Glenn Merlino, Funda Ergun, and S. Cenk Sahinalp. PhISCS-BnB: a fast branch and bound algorithm for the perfect tumor phylogeny reconstruction problem. *Bioinformatics (Oxford, England)*, 36(Suppl_1):i169–i176, July 2020. doi:10.1093/bioinformatics/btaa464.

19 Gryte Satas, Simone Zaccaria, Geoffrey Mon, and Benjamin J Raphael. Scarlet: single-cell tumor phylogeny inference with copy-number constrained mutation losses. *Cell systems*, 10(4):323–332, 2020.

20 A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

21 Jochen Singer, Jack Kuipers, Katharina Jahn, and Niko Beerenwinkel. Single-cell mutation identification via phylogenetic inference. *Nature communications*, 9(1):5144, 2018.

22 Dekel Tsur. Faster parameterized algorithms for Bicluster Editing and Flip Consensus Tree. *Theoretical Computer Science*, 953:113796, April 2023. URL: https://www.sciencedirect.com/science/article/pii/S0304397523001093, doi:10.1016/j.tcs.2023.113796.

23 Leah L Weber, Chuanyi Zhang, Idoia Ochoa, and Mohammed El-Kebir. Phertilizer: Growing a clonal tree from ultra-low coverage single-cell dna sequencing of tumors. *PLoS computational biology*, 19(10):e1011544, 2023.

24 Yochai Wolf, Osnat Bartok, Sushant Patkar, Gitit Bar Eli, Sapir Cohen, Kevin Litchfield, Ronen Levy, Alejandro Jiménez-Sánchez, Sophie Trabish, Joo Sang Lee, Hiren Karathia, Eilon Barnea, Chi-Ping Day, Einat Cinnamon, Ilan Stein, Adam Solomon, Lital Bitton, Eva Pérez-Guijarro, Tania Dubovik, Shai S. Shen-Orr, Martin L. Miller, Glenn Merlino, Yishai Levin, Eli Pikarsky, Lea Eisenbach, Arie Admon, Charles Swanton, Eytan Ruppin, and Yardena Samuels. UVB-Induced Tumor Heterogeneity Diminishes Immune Response in Melanoma. *Cell*, 179(1):219–235.e21, September 2019. doi:10.1016/j.cell.2019.08.032.

25 Yufeng Wu. Accurate and efficient cell lineage tree inference from noisy single cell data: the maximum likelihood perfect phylogeny approach. *Bioinformatics*, 36(3):742–750, 2020.

26 Hamim Zafar, Anthony Tzen, Nicholas Navin, Ken Chen, and Luay Nakhleh. Sifit: inferring tumor trees from single-cell sequencing data under finite-sites models. *Genome biology*, 18:1–20, 2017.

7 Appendix

7.1 Algorithms

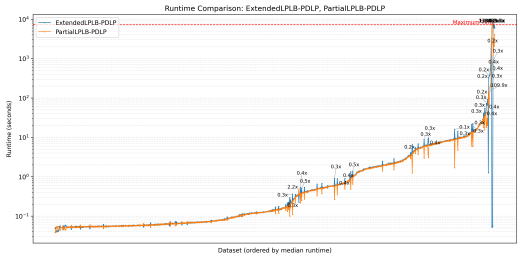
Input: H is the matrix at the current branch-and-bound node, I is the initial matrix, Π is an LP-VC written using $(V, E) = (V_I, E_I)$, and `solver` is an LP solver.
function *PartialLPLowerBound*$(I, H, \Pi,$ *solver*$)$
  for $H_{ip} = 1$ *and* $I_{ip} = 0$ **do**
    Set $x_{ip} \geq 1$ in Π;
  end
  $x^*_{ip} \leftarrow$ optimal solution to Π using `solver`;
  for $H_{ip} = 1$ *and* $I_{ip} = 0$ **do**
    Set $x_{ip} \geq 0$ in Π (i.e., unset the previous $x_{ip} \geq 1$);
  end
  for $H_{ip} = 0$ **do**
    $X^*_{ip} \leftarrow 1$ if $x^*_{ip} \geq \frac{1}{2}$ else 0;
  end
  $C \leftarrow \{(i, p) : X^*_{ip} = 1\}$;
  return $\left\lceil \sum_{i,p} x^*_{ip} \right\rceil$, *CandidateBestNode*$(H, C)$
end

Algorithm 4 Lower bound while only partially rewriting (LP-VC)

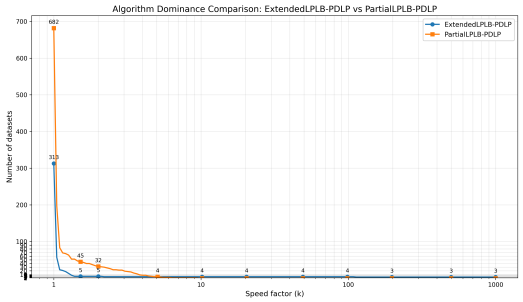
Input: H is the matrix at the current branch-and-bound node.
function *VCLowerBound*(H)
  $(V_H, E_H) \leftarrow$ VC instance from H as described in Section 4.2;
  $C \leftarrow \emptyset$;
  for $((i, p), (j, q)) \in E_H$ **do**
    if $(i, p) \notin C$ *and* $(j, q) \notin C$ **then**
      $C \leftarrow C \cup \{(i, p), (j, q)\}$;
    end
  end
  return $\left\lceil \frac{|C|}{2} \right\rceil$, *CandidateBestNode*$(H, C)$
end

Algorithm 5 Lower bound using a VC instance

652 ## 7.2 Additional Figures

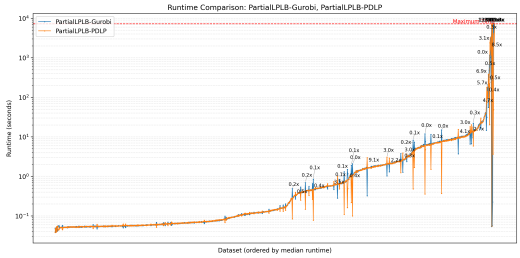


(a) Speed comparison of ExtendedLPLB(PDLP) and PartialLPLB(PDLP).

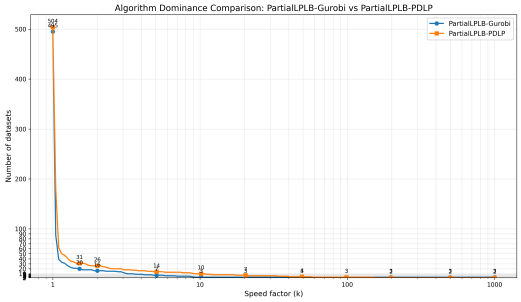


(b) Speedup factor comparison between PartialLPLB(PDLP) and PhISCS-BnB-MWS.

Figure 4 PartialLPLB(PDLP) slightly outperforms its Extended LP counterpart in more instances throughout, but ExtendedLPLB(PDLP) comes out on top in some of the harder instances of Figure 4a.



(a) Speed comparison of PartialLPLB(Gurobi) and PartialLPLB(PDLP).



(b) Speedup factor comparison of PartialLPLB(Gurobi) & PartialLPLB(PDLP).

Figure 5 These plots compare the runtimes of PartialLPLB using PDLP and Gurobi as the underlying solvers. Generally, PDLP was most performant, especially as the instances take longer to solve. Figure 5b shows each speedup factor k, as well as a count of the number of datasets in which an algorithm finishes k times faster than the other.

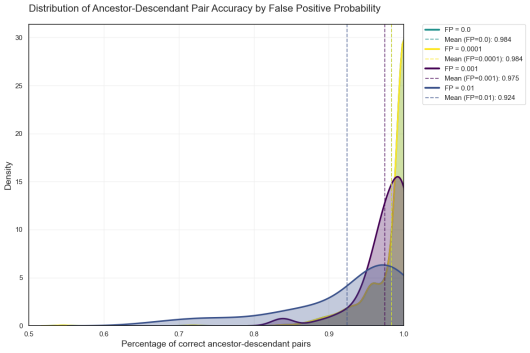


Figure 6 Assessment of robustness in the presence of false positive noise. Ancestor–descendant accuracy is shown for varying false positive rates (for the definition of this measure, please refer to [14]).

7.3 Proofs

Proof of Lemma 8. It suffices to show that the feasible region of (LP-Extended) is a subset of (LP-VC). Let y^* be a feasible solution of (LP-Extended) and consider a pair of zeros in an initial conflict, say, at coordinates (i, p) and (j, q). Then,

$$B(p, q, 0, 1) \geq -y^*_{ip} + y^*_{iq} = -y^*_{ip} + 1. \tag{9}$$

Similarly,

$$B(p, q, 1, 0) \geq y^*_{jp} - y^*_{jq} = 1 - y^*_{jq}. \tag{10}$$

As there is an in initial conflict between p and q, it must be that $|R_{11}(p, q)| > 0$ so

$$B(p, q, 1, 1) = 1. \tag{11}$$

Applying (6) alongside (9), (10), and (11), gives

$$(-y^*_{ip} + 1) + (1 - y^*_{jq}) + 1 \leq 2,$$

which simplifies to $y^*_{ip} + y^*_{jq} \geq 1$. In other words, y^* satisfies (1) and thus lies in the feasible region of (LP-VC). ◀

Proof of Lemma 10. Consider the matrix

$$U = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

We can easily verify by hand that $\mathrm{OPT}_{\mathrm{init}}(U) = 2$, corresponding to flipping the two zeros in the last two rows. With the aid of an LP solver, however, we can verify that $L_2(U) = 5$, corresponding to flipping either the first or second column, in addition to the bottom two rows. We can derive more matrices, with larger gaps between L_2 and $\mathrm{OPT}_{\mathrm{init}}$, by increasing the number of $(0, 1, 0)$ and $(1, 0, 0)$ rows. ◀