TOWARDS CHARACTERIZING THE SOLUTION SPACE OF THE 1-DOLLO PHYLOGENY PROBLEM

BY

SHUNPING XIE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2019

Urbana, Illinois

Adviser:

Assistant Professor Mohammed El-Kebir

ABSTRACT

Cancer cells may mutate multiple times, from a normal state to a mutated state and vice versa. Given our sequenced data, we can model the mutation process with a phylogenetic tree. One representative model is the *k-Dollo parsimony*, where all observed mutations mutate from a single normal cell and each character of a cell is *gained* at most once and *lost* at most k times. We examine the *1-Dollo Phylogeny problem*, does a *1-Dollo phylogeny*, a tree that follows the *1-Dollo parsimony* model, exist for the observations.

Current algorithms to solve the *1-Dollo Phylogeny problem* only tell us whether or not a set of observations has a *1-Dollo phylogeny* by outputting a single solution. We explore the structure of *1-Dollo phylogenies* and use our idea of a *skeleton* to develop an algorithm that enumerates all *1-Dollo phylogenies* for any set of observations. This algorithm runs much faster than the naive brute force enumeration algorithm for random input. The implementation is here: `https://github.com/sxie12/skeleton_solver`.

To my parents, for their love and support.

ACKNOWLEDGMENTS

I would like to thank my advisor, Assistant Professor Mohammed El-Kebir, for proposing and guiding me through this project.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Cancer is a genetic disease characterized by the uncontrolled growth of cells which disrupt the function of normal cells. It is caused by changes in the genes that control how cells function. These cells mutate in a way that cause them to divide more than usual or not go through the normal cell death process. These extra growths are called tumors. When multiple, different mutations occur within the same tumor, this is known as *intra-tumor heterogeneity*. These are often difficult to treat because of the various diverse cancer cells [Heppner(1984)]. Knowing the mutation history of these cells gives us a better understanding of why and how intra-tumor heterogeneity occurs. This may lead to more effective treatment strategies [Tabassum and Polyak(2015)].

Phylogenetic trees are commonly used to model and reason about cancer progression and also have clinical implications [Schwartz and Schaffer(2017)]. We consider a character-based phylogeny tree T. The sequenced data are the leaves of the tree, called *taxa*. Each taxon is the set of characters that we observed have mutated. The edges introduce changes to the characters, which can be gained or lost. A gain of character c means every cell corresponding to the leaves under that edge has a mutation of character c provided they do not *back mutate* later. A loss of character c, known as a *back mutation*, means every cell corresponding to the leaves under that edge does not have a mutation of character c when it was sequenced. Multiple characters can be gained and/or lost on the same edge. We assume all sequenced data start from a single normal cell. As the cell progresses, it mutates and divides creating new cells with these mutations. These cells repeat the process, mutating on their own independent of the others, causing the intra-tumor heterogeneity [Nowell(1976)]. We consider the more restrictive *1-Dollo parsimony* model [Dollo(1893), Farris(1977)]. In this model, each character is gained at most once and lost at most once. This is a special case of the *k-Dollo parsimony* model, where each character is gained at most once and lost at most k times. A *1-Dollo phylogeny* is a tree that follows the 1-Dollo parsimony model. The 1-Dollo Phylogeny problem, abbreviated as 1-DP problem and also known as the *persistent phylogeny problem*, is as follows: given sequenced data, does there exist a 1-Dollo phylogeny for that data and if so, construct one [Bonizzoni et al.(2012)].

Currently, hardness of the 1-DP problem is open. In the first part of this paper, we use exhaustively enumeration to identify patterns among small matrices that do not admit a solution. We expect that these results will provide guidance to resolve the hardness question.

In the latter half, we explore the counting version of the 1-DP problem. How many different 1-Dollo phylogenies can represent a given set of observations? Similar to the 1-

DP problem, hardness of this is also open. Current treatments are normally based on a single tree. However there may be multiple plausible phylogenies for the sequenced data. To avoid drawing incorrect conclusions in downstream analysis, we need to consider all possible trees [Pradhan and El-Kebir(2018)]. No algorithm other than the naive brute force enumeration algorithm has been proposed for this problem before. We introduce the notion of a *skeleton* for every 1-Dollo phylogeny and then propose an algorithm using this idea to enumerate all possible 1-Dollo phylogenies. This algorithm runs in $O(n^{(n+m+2)}m^2)$ whereas the naive brute force enumeration algorithm runs in $O(2^k kmn)$ where k is the number of zeros. It is much faster than the brute force algorithm in practice.

1.1 PRELIMINARIES

Suppose we sequence m different cells within the same tumor where each reading has n characters. Assume the reading is completely accurate. This is not a safe assumption in practice, but there are techniques to estimate the true values given the observations. We represent the reading as an m by n binary matrix, $B \in \{0,1\}^{m\times n}$, for convenience. The rows are the individual readings and the columns are the characters. A 1 indicates the character for that reading is at a mutated state and 0 indicates its at a normal state. We first consider the simpler case of no loss edges. Consider the phylogeny for these observations where edges are only gain edges or unlabeled. The definition of a *perfect phylogeny* for the binary case follows.

Definition 1.1. *[Estabrook et al.(1975), Gusfield(1991)]: A rooted tree T is a* perfect phylogeny *for a matrix $B \in \{0,1\}^{m\times n}$ if*

1. *T has m leaves that uniquely correspond to rows of B.*
2. *The root r of T is labeled by vector $\boldsymbol{b}_r = [0,0,\ldots,0]^T$.*
3. *Nodes labeled with state i for character c form a connected subtree of T*

This definition actually works for all matrices $B \in \{0,1,\ldots,k\}^{m\times n}$, not just binary ones. The second and third conditions ensure each character is gained at most once and never lost. This brings up the *perfect phylogeny problem*. Given an m by n binary matrix, does there exist a perfect phylogeny for that matrix and if so, construct it? If the answer is yes, that matrix is called a *perfect phylogeny matrix*. Gusfield showed the perfect phylogeny problem can be solved in polynomial time. There exists a simple characterization for the perfect phylogeny matrices.

Theorem 1.1. *Perfect Phylogeny Theorem [Gusfield(1991)]: A binary matrix* $B \in \{0,1\}^{m \times n}$ *is a perfect phylogeny matrix if and only if it does not contain the submatrix*

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

or any permutation of its rows.

We call that submatrix a *forbidden matrix*. The naive check runs in $O(m^3n^2)$ time. Gusfield showed this can be checked and the phylogeny, if one exists, can be constructed by a greedy constructive algorithm in $O(mn)$ time [Gusfield(1991)].

Now consider the case where characters can back mutate. The formal definition of 1-Dollo phylogeny follows.

Definition 1.2. *[El-Kebir(2018)]: A rooted tree* T *is a* 1-Dollo phylogeny *for a matrix* $B \in \{0,1\}^{m \times n}$ *provided that*

1. T *has* m *leaves that uniquely correspond to rows of* B*.*
2. *The root* r *of* T *is labeled by vector* $\boldsymbol{b}_r = [0, 0, \ldots, 0]^T$*.*
3. *For each character* $c \in [n]$*, there is at most one gain edge* (u, v) *in* T *such that* $b_{u,c} = 0$ *and* $b_{v,c} = 1$*.*
4. *For each character* $c \in [n]$*, there is at most one loss edge* (u, v) *in* T *such that* $b_{u,c} = 1$ *and* $b_{v,c} = 0$*.*

We impose an ordering of the gains and losses with the last three conditions. A character must be gained before it can be lost. An explicit way to enforce this is with a *state tree* S for each character.

Definition 1.3. *[Fernàndez-Baca(2001)]: A* state tree S *is a rooted tree whose root is labeled 0 and the other vertices are distinct values in the set* $\{1, \ldots, k{+}1\}$ *where* k *is the number of vertices.*

For the 1-DP problem, the state tree looks like a linked list with three vertices. Vertex 0 is the root, vertex 1 is its only child, and vertex 2 is a child of vertex 1. Every character has the same state tree. A label of 0 indicates the character is not gained or lost. A label of 1 indicates the character is gained and not lost. A label of 2 indicates the character is gained and then lost.

A phylogeny T is *consistent* with a state tree S if for any pair of nodes (u, v) in T, if u is an ancestor of v, then for all characters, the label of u for a character must be an ancestor of the label of v for the same character in that character's state tree. Given matrix $A \in \{0, 1, \ldots k+1\}^{m \times n}$, the problem of finding a perfect phylogeny for A consistent with a state tree S for each character is the *cladistic perfect phylogeny problem.* This problem can be solved in $O(mnk)$ time [Fernàndez-Baca(2001)]. Given matrix A and state trees $S = \{S_1, \ldots, S_n\}$, define the *(k+1) binary factor matrix* B' of (A, S) as

Definition 1.4. *[Fernàndez-Baca(2001)]: Let matrix* $A \in \{0, 1, \ldots k+1\}^{m \times n}$ *and state trees* $S = \{S_1, \ldots, S_n\}$. *The* binary factor matrix $B' \in \{0,1\}^{m \times n(k+1)} = [b'_{i,j}]$ *of* (A, S) *has entries*

$$b'_{i,j} = \begin{cases} 1 \text{ if the vertex with value } l \text{ is an ancestor of the vertex with value } a_{i,q} \text{ for } S_c \\ 0 \text{ otherwise} \end{cases}$$

where $q = \lfloor \frac{j}{k+1} \rfloor + 1$ *and* $l = (j \mod (k+1)) + 1$.

The expansion of each character enforces the state tree ordering constraint. After the expansion, the problem becomes the perfect phylogeny problem on an $m \times n(k+1)$ size matrix which can be solved with Gusfield's perfect phylogeny algorithm in $O(mnk)$ time.

For the 1-DP problem with $A \in \{0, 1, 2\}^{m \times n}$ and the state tree described above, the binary factor matrix $B' \in \{0,1\}^{m \times 2n}$ has entries

$$b'_{i,2j}, b'_{i,2j+1} = \begin{cases} 0,0 \text{ if } a_{i,j} = 0 \\ 1,0 \text{ if } a_{i,j} = 1 \\ 1,1 \text{ if } a_{i,j} = 2 \end{cases}$$

We relate A with B.

Definition 1.5. *[El-Kebir(2018)]: Let* $B \in \{0,1\}^{m \times n}$. *A matrix* $A \in \{0,1,2\}^{m \times n}$ *is a* 1-completion *of* B *if* $a_{i,j} \in \{0, 2\}$ *when* $b_{i,j} = 0$ *and* $a_{i,j} = 1$ *when* $b_{i,j} = 1$.

Definition 1.6. *[El-Kebir(2018)]: Let* $B \in \{0,1\}^{m \times n}$. *A matrix* $A \in \{0,1,2\}^{m \times n}$ *is a* 1-Dollo completion *of* B *if it does not contain any of the following submatrices*

$$\begin{pmatrix} i_1 & 0 \\ 0 & j_1 \\ i_2 & j_2 \end{pmatrix} \text{ or } \begin{pmatrix} i_1 & 1 \\ 0 & 2 \\ i_2 & 2 \end{pmatrix} \text{ or } \begin{pmatrix} 2 & 0 \\ 1 & j_1 \\ 2 & j_2 \end{pmatrix} \text{ or } \begin{pmatrix} 2 & 1 \\ 1 & 2 \\ 2 & 2 \end{pmatrix}$$

where $i_1, i_2, j_1, j_2 \in \{1, 2\}$

Corollary 1.1. *If matrix* $A \in \{0, 1, 2\}^{m \times n}$ *is a* 1*-Dollo completion, then any submatrix of* A *is also a* 1*-Dollo completion. Likewise, if* A *is not a* 1*-Dollo completion, any matrix with* A *as a submatrix also is not a* 1*-Dollo completion.*

All these definitions relate together.

Theorem 1.2. *[El-Kebir(2018)]: Let* $B \in \{0, 1\}^{m \times n}$*. The following statements are equivalent.*

1. *There exists a* 1*-Dollo phylogeny* T *for* B*.*
2. *There exists a* 1*-Dollo completion* A *or* B*.*
3. *There exists a* 1*-completion* C *of* B *such that the binary factor matrix* B' *of* (C, S) *is a perfect phylogeny matrix where* S *is a set of state trees with three vertices labeled with values in* $\{0, 1, 2\}$*.*
4. *There exists a* 1*-completion* C *of* B *and perfect phylogeny* T *for* C *whose characters are consistent with the state tree described above.*

Due to this relationship, it suffices to look at any of these problems. This also allows us to check if a 1-completion is a 1-Dollo completion and construct the 1-Dollo phylogeny if so in $O(mn)$ time. Therefore, the 1-DP problem is in NP, but it is open whether it is NP-complete.

We mainly jump between the first two problems and exploit the relationship between them for this paper. **We will leave out "1-Dollo" from completion and phylogeny when it is obvious from context.**

CHAPTER 2: CHARACTERIZING INFEASIBLE INSTANCES

To gain insight on the hardness of the 1-Dollo phylogeny problem, we explore the submatrices of binary matrices without a completion for the 1-DP problem to see if simple forbidden matrix rules exist like in the perfect phylogeny case. Using the k-DP solver from SPhyR [El-Kebir(2018)], we check all matrices up to size 5×5. Since the number of matrices is the power set of the size of the matrix and the order of the rows do not matter, we remove matrices that are the same up to permutation of its rows to save time. These matrices form an equivalence class. As an example, the following matrices are in the same equivalence class under row permutation.

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \equiv \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \equiv \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

We only need to run SPhyR on one of them so we pick the rightmost matrix because the bitset representation of its rows, 011, 101, and 110, are in non decreasing order. This check is executed during the generation of all matrices. Note we only removed row symmetry. There may still be matrices which are the same up to permutation of its columns. For ideal results, we want to ensure distinct rows and columns and remove all row and column symmetry. This can be achieved using the Pólya enumeration theorem [Kerber(1988)].

A completion with a loss is a matrix that has a completion, but is not a perfect phylogeny. We call a matrix without a completion *trivial* if it contains a submatrix that also does not have a completion. Therefore, the *non-trivial* matrices are those where every submatrix has a completion. We extend the definition of *forbidden matrix* from the perfect phylogeny case to the 1-Dollo case. A matrix is a *1-Dollo forbidden matrix* if it is a non-trivial matrix. By corollary 1.1, any matrix that contains a 1-Dollo forbidden matrix does not have a completion. Since we calculated the smaller matrices first, we were able to find the forbidden matrices recursively using the smaller instances that do not admit a completion. However, one has to be careful and sort the submatrices so the rows are in non decreasing order. Our brute force results are in table 2.1.

The 4×4 case is the smallest case where there exist matrices without a completion. The

Size	Computed	Completion w/ loss	No completion	No completion w/o trivial (1-Dollo forbidden matrices)
2x2	10	0	0	0
2x3	36	0	0	0
3x2	20	1	0	0
3x3	120	19	0	0
3x4	816	243	0	0
4x3	330	111	0	0
4x4	3876	2209	3	3
4x5	52360	38985	210	0
5x4	15504	11554	79	31
5x5	376992	326626	11814	1704

Table 2.1: Brute force results with SPhyR

three matrices we found without a completion for 4×4 are

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

There is a pattern for these matrices. Every row and column contains exactly two ones. They are symmetric with respect to column permutation. They are all the same in this regard.

We look at them in a different representation. Let T be the set of taxa and C be the set of characters. Consider the bipartite graph $G(B) = (T, C, E)$ where B is a binary matrix and $E = \{(t_i, cj) \mid b_{ij} = 1\}$. An example of this graph for the leftmost 4×4 matrix listed above is figure 2.1. The taxa are labeled in order from 1 to 4 and the characters are labeled in order from A to D.

The resulting graphs for the above three matrices are all cycle graphs with eight vertices in this representation. These matrices are also the same here. We redraw the above bipartite graph as a cycle graph shown in figure 2.2.

The row and column symmetric representation may be related to this bipartite graph representation, since they both have only one unique matrix for the 4×4 case. Our observation in this representation led to two lemmas.

Lemma 2.1. *If the bipartite graph $G(B)$ is the cycle graph C_{2n} with $n \geq 4$ vertices, then B does not have a completion.*

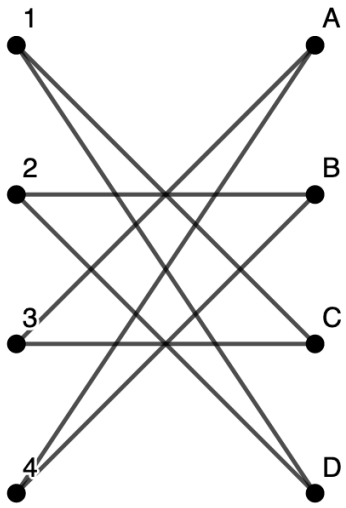

Figure 2.1: Bipartite graph $G(B)$

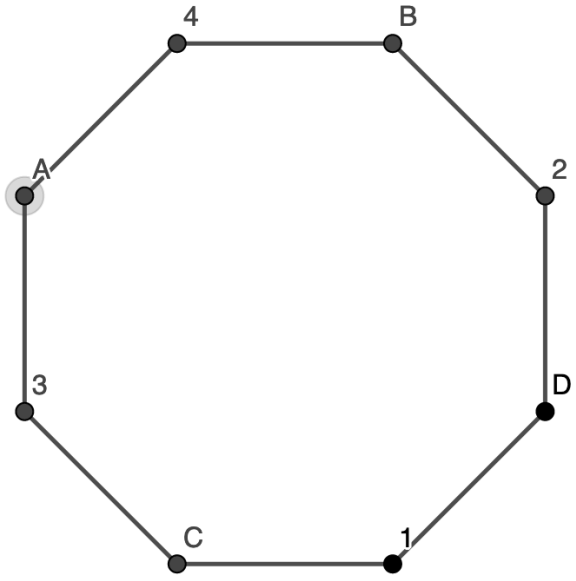

Figure 2.2: $G(B)$ redrawn as a cycle graph

Proof. We show B does not have a completion by showing no phylogeny can be constructed and applying theorem 1.2. Observe at most two taxa can have no losses because each character is gained by two taxa. We have two cases.

1. If there are two taxa without losses, they share a common character c. Then the phylogeny must branch out to gain the different character. Let those be a and b. Each of those subtrees contain at least one child because there is one other taxon with character a and same for b. However, both of these are on different branches and must lose c. We are only allowed to lose c on one of these branches.

2. If there is one taxon without losses. Let the two characters it has be a and b. Consider the tree. It starts with gain a and b. All other nodes must be descendants of this internal node due to our restriction. There are two different taxa that have characters a and b. Let those taxa have $\{a, c\}$ and $\{b, d\}$ respectively. These two must be on separate branches because the first one has to lose b and second has to lose a. However there exists a taxon which has to lose both a and b. Clearly this is not possible because the losses occur on different branches.

In both cases, no phylogeny can be constructed.

Lemma 2.2. *Let $B \in \{0,1\}^{n\times n}$. If $G(B) = C_{2n}$ for $n \geq 4$, then B is a 1-Dollo forbidden matrix.*

Proof. By above, B does not have a completion. We show all submatrices of B have a completion by greedily constructing the equivalent phylogeny for them. Consider submatrices formed by removing rows and/or columns. Remember any submatrix of a matrix with a completion also has a completion. We only need to consider the case when we remove one row or one column of B because any submatrix of B will be a submatrix of either of these.

1. Remove one row. Assume without loss of generality that we removed the n^{th} row. Look at the underlying bipartite graph $G(B)$. It is a linked list with character nodes at its endpoints. Consider the linked list formed by removing all the character nodes and adding an edge between two taxa nodes if they had an edge to the same character. The resulting linked list is of the form $\{c_1, c_2\}, \{c_2, c_3\}, \ldots, \{c_{n-1}, c_n\}$ where the c_i are characters and each taxon node is represented by the set of characters in the taxon. Consider the endpoints. Those taxa each have a character no other taxa have. Start at either one of them. We start at the $\{c_1, c_2\}$ taxon. Gain c_1 and c_2 in the tree. The first taxon is set. Then lose c_1 and gain c_3 so the second taxon is set. Repeat this process of lose c_k and gain c_{k+2} for all $k \geq 2$ until we finish with all the taxa. Each

taxon is satisfied in the order they are in the linked list. Note the internal vertices in the phylogeny form a linked list as well.

2. Remove one column. Assume without loss of generality that we removed the n^{th} column. This construction is similar to above. Again look at the bipartite graph $G(B)$ and apply the transformation above. The result is also a linked list, but of the form $\{c_1\}, \{c_1, c_2\}, \ldots, \{c_{n-2}, c_{n-1}\}, \{c_{n-1}\}$. In this case, the endpoints are taxa with exactly one character. We start at $\{c_1\}$. Gain that first so the first taxon is set. Then gain c_2 so the second taxon is set. Then lose c_1 and gain c_3 to complete the third taxon. Repeat this process of lose c_k and gain c_{k+2} for all $k \geq 2$ until we finish with all the taxa. Each taxon is satisfied in the order they are in the linked list. The phylogeny looks very similar to the case above.

Hence, there exists a phylogeny in both cases.

We look at the 5×4 case next to see if those forbidden matrices follow a similar pattern. However no obvious pattern is recognizable in either representation. Three of the 31 forbidden matrices for 5×4 are

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Three of the 1704 forbidden matrices for 5×5 are

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

While the two state perfect phylogeny has a well characterized forbidden matrix of constant size, the number of 1-Dollo forbidden matrices seems to grow as a function of m and n. There does not appear to be a way to characterize all of them. These small forbidden matrices may be useful in constructing big examples that do not admit a completion.

CHAPTER 3: CHARACTERIZING SOLUTION SPACE

3.1 SKELETON

To create an enumeration algorithm for the counting version of 1-DP, we introduce the notion of a backbone for all 1-Dollo phylogenies. We refer to it as a *skeleton*.

Define the following functions where v is any leaf of a 1-Dollo phylogeny.

- $H(v)$ is the set of characters where there is a gain edge on the path from v to the root of the tree.
- $I(v)$ is the set of characters where there is a loss edge on the path from v to the root of the tree.
- $J(v)$ is the set of characters where there is a gain edge and no loss edge on the path from v to the root of the tree.

Observation 3.1. *For all leaves v in a 1-Dollo phylogeny T, $H(v) \setminus I(v) = J(v)$.*

A *skeleton* is similar to a *1-Dollo phylogeny* except it does not contain loss edges. Therefore, the definition is the same as the *1-Dollo phylogeny* except for loss edges and the last condition.

Definition 3.1. *A* skeleton *of a tree T is a tree for a matrix $B \in \{0,1\}^{m \times n}$ that satisfies the following conditions*

1. *T has m leaves that uniquely correspond to rows of B.*
2. *The root r of T is labeled by vector $\boldsymbol{b}_r = [0, 0, \ldots, 0]^T$.*
3. *For each character $c \in [n]$, there is at most one gain edge (u, v) in T such that $b_{u,c} = 0$ and $b_{v,c} = 1$.*

Observation 3.2. *For all leaves v in T, $J(v) \subseteq H(v)$.*

Define the following functions on skeletons, T, where v is any leaf of the skeleton and B is the corresponding binary matrix.

- $\hat{H}(B, v)$ is the set of characters of v with value 0 in B.
- $\hat{I}(B, v)$ is the set of characters of v with value 0 in B but is in gained on the path from v to the root of the tree.

- $\hat{J}(B, v)$ is the set of characters of v with value 1 in B.

Observe $\hat{H} = H$, $\hat{I} = I$, and $\hat{J} = J$.

Every taxon in our skeleton gained a superset of the characters required. Now, we want to solve the "inverse" problem. Given the gains, we want to add loss edges so that the resulting tree is a 1-Dollo phylogeny. To enforce the constraints, we use the state tree described in the beginning. Let B' is the binary factor matrix associated with this problem. We formally define it below.

Definition 3.2. *Let $B \in \{0,1\}^{m \times n}$. $B' = \sigma(B, T)$ is a $m \times 2 \cdot n$ binary matrix where each character in B correspond with two characters in B'. For each leaf v in T, each character c in B maps to the following two in B'*

- *00 if c is never gained, $c \notin \hat{H}(B, v)$.*
- *10 if c is gained and cannot be lost, $c \in \hat{J}(B, v)$.*
- *11 if c is gained and must be lost, $c \in \hat{I}(B, v)$.*

Definition 3.3. *A rooted tree T' is a* valid skeleton *for $B \in \{0,1\}^{m \times n}$ if $B' = \sigma(B, T')$ is a perfect phylogeny matrix.*

Edges can have any number of labels. Because of this, we define a new condition to compress the tree and establish uniqueness.

Definition 3.4. *We call an edge (u, v) an* internal edge *if both u and v are internal vertices. We say a tree T is $\phi(T)$ if there are no internal edges with in-degree one.*

Given these definitions, we make two observations. The proofs are in our main result.

Observation 3.3. *A 1-Dollo phylogeny T has a unique valid skeleton T'.*

Note that there are equivalence classes of 1-Dollo phylogenies where each class consists of phylogenies of branches of out-degree one.

Observation 3.4. *A valid skeleton T' has a unique 1-Dollo phylogeny T provided $\phi(T)$.*

Note none of these observations depend on the matrix B. There may be multiple 1-Dollo phylogenies for B.

We show a corollary that we will use in our main result.

Corollary 3.1. *If a binary matrix A has a perfect phylogeny, then any submatrix of A also has a perfect phylogeny.*

This follows directly from the fact that any submatrix of a matrix with a completion also has a completion and the relationship established in Theorem 1.2.

Theorem 3.1. *There exists a 1-Dollo phylogeny T for B if and only if there exists a valid skeleton T' for B.*

Proof. We first prove the ($\Rightarrow$) direction. Suppose we know there exists a 1-Dollo phylogeny, T, for B. Then, we can create T' from T using the following algorithm.

1. Remove all labels of edges that introduce a loss edge
2. Collapse edges (u, v) if it does not have a label and v is not a leaf. This means remove v and attach all the children of v to u.

The no loss edges condition is satisfied because we removed them in step 1. For all leaves v, $J(v) \subseteq H(v)$ because we did not modify the gain edges nor the position of the leaves relative to the gain edges. All moves are fixed so the result is unique. All there is left to show is the skeleton is valid. We show this by contradiction. Assume the skeleton is not valid. Thus $B' = \sigma(B, T')$ has the submatrix

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$

We choose two columns and three rows. Observe the two columns in B' cannot come from the same character in B because no character maps to 01. Let c, d be characters where $c \neq d$ and c_l, c_r be the left and right values of character c in B'. Observe if $c_l = 1$, then it must have gained c (and maybe lose) at some point. If $c_r = 1$, it must have gained and lost c. To simplify our proof, let 0 denote character was never gained, 1 denote the character was gained and never lost, and 2 denote the character was gained and lost, like the state tree. There are four cases for the columns we pick.

- c_l, d_l. The corresponding completed submatrix in B looks like

$$\begin{pmatrix} 1/2 & 0 \\ 0 & 1/2 \\ 1/2 & 1/2 \end{pmatrix}$$

  where 1/2 means it could either be a 1 or a 2. These characters are gained and potentially lost. We'll show there is no 1-Dollo phylogeny for even gaining the edges. There are two cases. Assume c is gained at or before d. Then the leaves corresponding

to the first and third rows are on a separate branch from the leaf corresponding to the second row. Then d is gained, but since the second and third are on different branches, it must be gained twice. This is not a valid phylogeny. Assume d is gained before c. Then the leaves corresponding to the second and third rows are on a separate branch from the leaf corresponding to the first row. Then c must be gained twice because the first and third are on separate branches. In both cases, there is no 1-Dollo phylogeny for B.

- c_l, d_r. The corresponding completed submatrix in B looks like

$$\begin{pmatrix} 1/2 & 0/1 \\ 0 & 2 \\ 1/2 & 2 \end{pmatrix}$$

  We have two cases for the 0/1 entry.

  - It is 0. Then there is no 1-Dollo phylogeny by the same reasoning as above
  - It is 1. There are two cases. Assume c is gained at or before d. Then the first and third are on a separate branch from the second. All of them have to gain d, which means it must be gained twice. Now assume d is gained before c. There are two subcases here. First is c is gained at or before d is lost. This means first and third are on a separate branch from second. Then we require two losses of d because second and third are on difference branches. If d is lost before c is gained, then c must be gained twice because first and third are on different branches.

  Therefore, there is no 1-Dollo phylogeny for B.

- c_r, d_l. The corresponding completed submatrix in B looks like

$$\begin{pmatrix} 2 & 0 \\ 0/1 & 1/2 \\ 2 & 1/2 \end{pmatrix}$$

  There is no 1-Dollo phylogeny for B using the same reasoning as above.

- c_r, d_r. The corresponding completed submatrix in B looks like

$$\begin{pmatrix} 2 & 0/1 \\ 0/1 & 2 \\ 2 & 2 \end{pmatrix}$$

Observe the losses. Assume we can gain all the characters we need. Now we need to lose those so they match the submatrix. This is not possible following the logic from the first case. Losing them is the same as gaining them. Therefore, these is no 1-Dollo phylogeny for B.

In all four cases, we show there is no 1-Dollo phylogeny for B. However, we know there is one, namely T. This contradicts our assumption so B' is conflict free and the skeleton, T', is valid.

Now, the ($\Leftarrow$) direction. We propose another algorithm to transform the skeleton to a 1-Dollo phylogeny.

1. Find all the leaves that require losses. Observe that leaf v requires losses if and only if $L(v) \neq \emptyset$.

2. Construct an indicator matrix, C, where each row corresponds to a leaf, v, that require losses and each character in the row is 1 if it is in the set $I(v)$ and 0 otherwise.

3. Construct a perfect phylogeny T_C for our indicator matrix C. We know one exists because C is a submatrix of the conflict free matrix $B' = \sigma(B, T')$ along with the corollary above.

4. Merge T_C with our skeleton to get the 1-Dollo phylogeny. The loss edges we need to add to the skeleton are exactly the edges in T_C. Consider an edge (u, v) in T_C where u is the parent of v. All the leaves in the subtree of v in T_C must be descendants of v in the skeleton. Therefore, we can add vertex V' and edge (u', v') to our skeleton where u' is the lowest common ancestor of all the leaves in the subtree rooted at v in T_C. Then attach all the children of u' to v' so that v' is now the lowest common ancestor of those leaves. Process the vertices in topological order in T_C so an edge is added before any of the edges in the subtree rooted at the incoming vertex.

This gives us a tree that satisfies the condition to be a 1-Dollo phylogeny. It unique provided that edges are allowed to have multiple labels because the perfect phylogeny is unique.

3.1.1 Additional optimizations

We can further optimize the algorithm by exploiting structure in our input B. Two of them are below.

Definition 3.5. *Let $B \in \{0,1\}^{m\times n}$ be a binary matrix and c be a column of that matrix. Define $I_c(c)$ to be the set of taxa that have a 1 for column c in B.*

A column in a binary matrix $B \in \{0,1\}^{m\times n}$ is *nonzero* if it has at least one 1.

Lemma 3.1. *Let c and d be two nonzero columns in the binary matrix $B \in \{0,1\}^{m\times n}$. If $I_c(d) \subset I_c(c)$ and c is not all 1, then the character corresponding to c is gained at or before the character corresponding to d in all 1-Dollo phylogenies of B.*

Proof. We prove this by contradiction. Let $B \in \{0,1\}^{m\times n}$ be a binary matrix with a phylogeny T. Let c and d be two columns in B with $I_c(d) \subset I_c(c)$ and c is not a column of all 1s. Assume the character corresponding to d is strictly gained before the character corresponding to c in T. Then consider the two columns (c d) in B and in the completion of B. We have three cases for the two columns in B. (0 1) is not possible because $I_c(d) \subset I_c(c)$.

1. They were (1 1). Then it will be the same in the completion of B. Observe there is at least one taxon with this because every column has at least one 1.

2. They were (1 0). Then it must be (1 2) in the completion of B because d is strictly gained before c. There is also at least one taxon with this because $I_c(d) \subset I_c(c)$.

3. They were (0 0). There is at least one taxon in B with this because c is not the column of all 1s. Then there is at least one taxon with (0 2) in the completion of B because d is gained strictly before c.

Therefore, the completion of B has the submatrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 0 & 2 \end{pmatrix}.$$

However, this is one of the forbidden submatrices. Therefore, there is no 1-Dollo phylogeny of B where character d is strictly gained before character c.

Lemma 3.2. *Let c and d be two nonzero columns in the binary matrix $B \in \{0,1\}^{m\times n}$. If $I_c(d) \cap I_c(c) = \emptyset$ and there exists a row where there is a zero in both c and d. That row cannot have a loss of c and d.*

Proof. We prove this by contradiction. Let $B \in \{0,1\}^{m \times n}$ be a binary matrix with a 1-Dollo phylogeny. Let c and d be two columns in B with $I_c(d) \cap I_c(c) = \emptyset$ and there is a row where there is a zero in both c and d. Assume characters c and d are in the same branch there exists a leaf that loses both. Then the completion of B must contain the submatrix where $x_1, x_2 \in \{0,2\}$

$$\begin{pmatrix} x_1 & 1 \\ 2 & 2 \\ 1 & x_2 \end{pmatrix}$$

However, all of these are forbidden matrices so there is no completion where that row loses both c and d.

Using these two, we can safely ignore skeletons that do not correspond to a 1-Dollo phylogeny.

3.2 ENUMERATION ALGORITHM

3.2.1 Description

We propose the *skeleton solver*, an algorithm that outputs all completions to an instance. Since there is a valid skeleton for a matrix if and only if there is a 1-Dollo phylogeny for that matrix and there is an algorithm to construct the phylogeny from the valid skeleton, we generate all valid skeletons to get all phylogenies for an input. The input is an m by n binary matrix. The output is a list of all completions in increasing order. A matrix X is less than another Y if the concatenated bitset representation of each row of X is less than that of Y. There are three main steps in the algorithm.

1. We first generate a skeleton. Skeletons only have gain edges so we try all trees with only gain edges. We can efficiently construct all of them because there is a bijection between these trees and Prufer sequences [Prüfer(1918)]. The Prufer sequence generates trees with labels on its vertices. Since we want to label the edges, the tree will have one more vertex than the number of characters. That "extra" vertex will be the root of the tree. The directed edges are labeled as the value of the vertex its going to. We did not use any optimizations that was listed in the previous section in this step.

2. To complete the skeleton, we have to decide the ancestor of each of the leaves given a tree of only gain edges. There are multiple possibilities for each leaf so we try them

all. Observe the condition for all leaves v is $J(v) \subseteq H(v)$. Therefore, u is an ancestor of leaf v if $J(v) \subseteq H(u)$.

3. Now we check if the skeleton is a valid skeleton. We generate the matrix in definition 8 above and check if it has a perfect phylogeny.

3.2.2 Analysis

Theoretical runtime. We compute the big-O runtime for each step of the algorithm and combine them in the end.

1. First, we compute all possible trees with the characters as gain edges. There are n characters so we need a tree with n edges. This tree has $n+1$ vertices. By Cayley's formula, there are $(n+1)^{(n-1)}$ of these trees, which correspond to Prufer sequences of length $n-1$. Constructing the tree from the Prufer sequence takes $O((n-1)^2)$ time. Hence, generating all trees takes $O((n-1)^2(n+1)^{(n-1)})$ time.

2. Now we have to create a skeleton. There are m leaves and at most $n+1$ possible ancestors for each of them. In order to enumerate them, we create a sequence of numbers where each position corresponds to a leaf and the value at that position is the ancestor for that leaf. Reading this sequence takes $O(m)$ time. We use bitset representation for the labels of the vertices so checking if a leaf can be a descendant of a vertex in the tree takes constant time. Therefore, the skeleton creation step requires $O(m(n+1)^m)$ time. This part runs much faster in practice because the number of possible ancestors is usually small.

3. The last step is to determine if the given skeleton is valid. Use the $O(mn)$ perfect phylogeny algorithm proposed by Gusfield.

Therefore, the total runtime is $O((n-1)^2(n+1)^{(n+m-1)}m^2n) = O(n^{(n+m+2)}m^2)$. Only the first step is deterministic. That gives this a lower bound runtime for the algorithm, which is the runtime for enumerating the Prufer sequence. The brute force algorithm runs in $O(2^k kmn)$ time where k is the number of zeros in B. Observe the extra k is there because it requires k time to read the k digits in the binary string.

Real world runtime. We test how fast this algorithm is in practice and how it scales based on the size. The input is grouped by the size of the matrix, each one containing fifty

Size	Total time (sec)	Average time of each	Total number of solutions
4×4	0.052877	0.00105754	11342
4×5	0.279887	0.00559774	20530
5×5	2.13644	0.0427288	207018
5×6	57.8506	1.157012	1403137
6×6	80.7195	1.61439	760135

Table 3.1: Runtime results for fifty random matrices

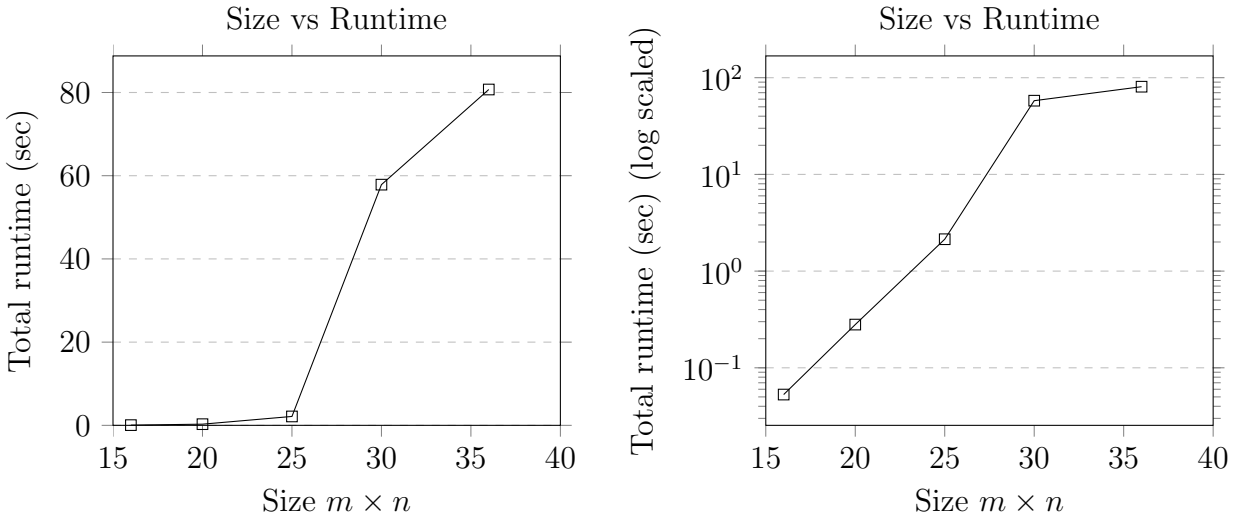

Figure 3.1: Graphs of runtime for fifty random matrices

random matrices. We randomly choose the number of zeroes uniformly from $[\lfloor \frac{n+m}{3} \rfloor, n \cdot m)$ and then randomly generate a matrix with that number of zeroes. We limited each input to fifty matrices and the size up to 6×6 because we noticed that the number of zeroes significantly impacts the runtime. Later, we test random matrices up to 8×8 where around half the entries are zero. We ran everything on a 3.3GHz quad core i5 desktop processor with 16GB of ram although the program only uses one thread. The results and graphs are in table 3.1 and figure 3.1.

The algorithm runs fast on the small instances and slows down on the bigger ones as expected. From the log scaled graph, it appears that the growth is exponential in the size. The theoretical runtime we computed is exponential in n, but our values of n are close which may be why we do not observe that. Nevertheless, our findings are consistent with the exponential growth in the theoretical runtime. There is a minor discrepancy for the 5×6 case. That may be due to the usually large number of solutions. We explore that further in the next section.

Now we test larger matrices up to 8×8. The entries of each matrix are generated uniformly

Size	Total time (sec)	Average time of each	Total number of solutions
6×6	2.03385	0.040677	2772
6×7	26.7019	0.534038	6792
7×7	49.3069	0.986138	856
7×8	423.769	8.47538	31
8×8	1249.04	24.9808	0

Table 3.2: Runtime results for fifty bigger random matrices

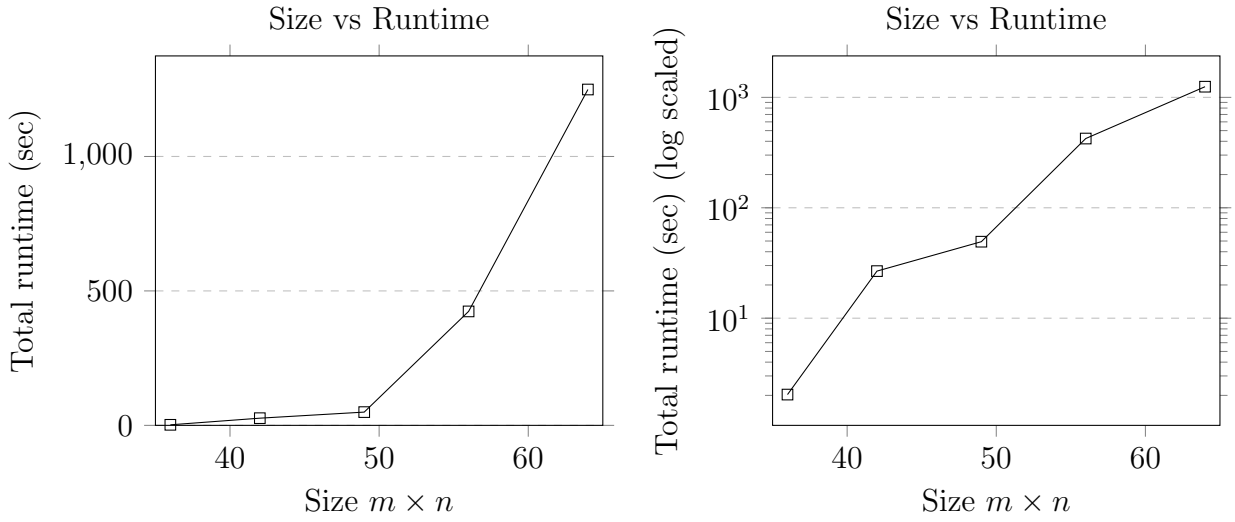

Figure 3.2: Graphs of runtime for fifty bigger random matrices

from $\{0, 1\}$ so the expected number of zeroes is half the size of the matrix. Like before, each input contains fifty random matrices. Our results are in table 3.2 and figure 3.2.

Again, the runtime is exponential on the size. It is tolerable for matrices up to size 7×7, but blows up after that. We do not recommend running instances larger than 8×8 based on this data. The number of solutions should not be very high so using existing solvers for a single solution is sufficient.

Real world runtime dependence on other factors Given the observations above, we test how the number of zeroes and number of solutions affect the runtime in practice. We first check how the runtime changes as the number of zeroes in the input changes since that is easier to isolate and test. Observe the big-O runtime does not directly depend on the number of zeroes in the matrix, unlike the brute force enumeration algorithm. We separate the matrices by the proportion of zeroes and run one hundred randomly generated instances of each. For example, 4×4 (4) means 4 by 4 matrices where 4 of the entries are zeroes. The results are in table 3.3.

Size (zeroes)	Total time (sec)	Average time of each	Total number of solutions
4×4 (4)	0.007765	0.00007765	783
4×4 (6)	0.013361	0.00013361	1268
4×4 (8)	0.022636	0.00022636	2983
4×4 (10)	0.054898	0.00054898	8296
4×4 (12)	0.190477	0.001904775	37822
5×5 (5)	0.046834	0.00046834	858
5×5 (10)	0.115808	0.00115808	2213
5×5 (15)	0.556503	0.00556503	8713
5×5 (20)	10.6249	0.106249	469942
6×6 (6)	0.497128	0.00497128	1202
6×6 (12)	0.896287	0.00896287	1366
6×6 (18)	3.40114	0.0340114	1359
6×6 (24)	30.1174	0.301174	37140
6×6 (30)	864.675	8.64675	6455296
7×7 (7)	6.84749	0.0684749	1269
7×7 (14)	10.3064	0.103064	1087
7×7 (21)	27.0798	0.270798	147
7×7 (28)	157.381	1.57381	878
7×7 (35)	2168.46	21.6846	65032

Table 3.3: Runtime and number of zeroes

It appears that the number of zeroes does affect the runtime of the algorithm. This may be because there are more possible ancestors for each taxon in the second step of our algorithm and so, the solution space is much bigger.

If we look at the small instances, it appears that the number of solutions affect the running time as well. However as we go to the bigger matrices, we see that is not the case. This makes sense if our hypothesis above is correct. The solution space may be very large, but the set of actual solutions is really small. It also makes sense intuitively because the number of solutions should go down as we increase the size of the matrix.

We plot these results below. The left graphs are normal scaled and the right ones are log scaled on the y-axis.

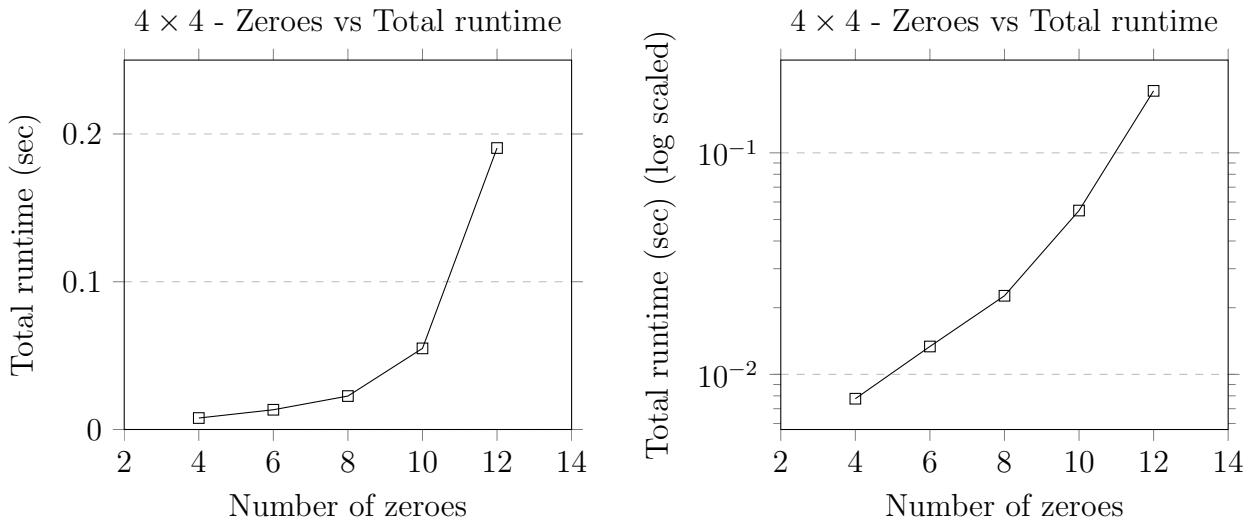

Figure 3.3: 4×4 runtime and number of zeroes

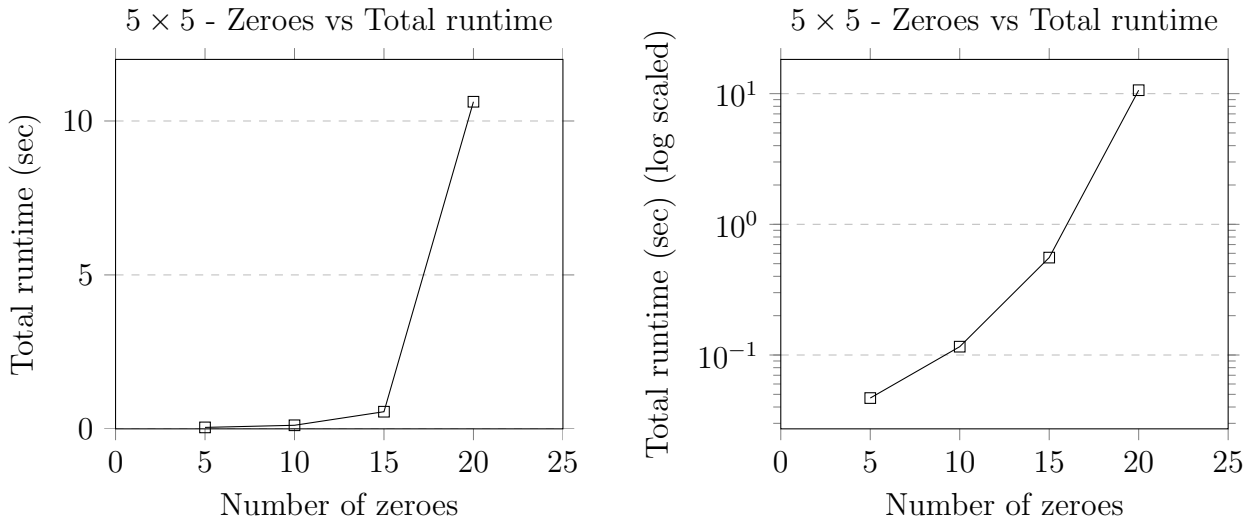

Figure 3.4: 5×5 runtime and number of zeroes

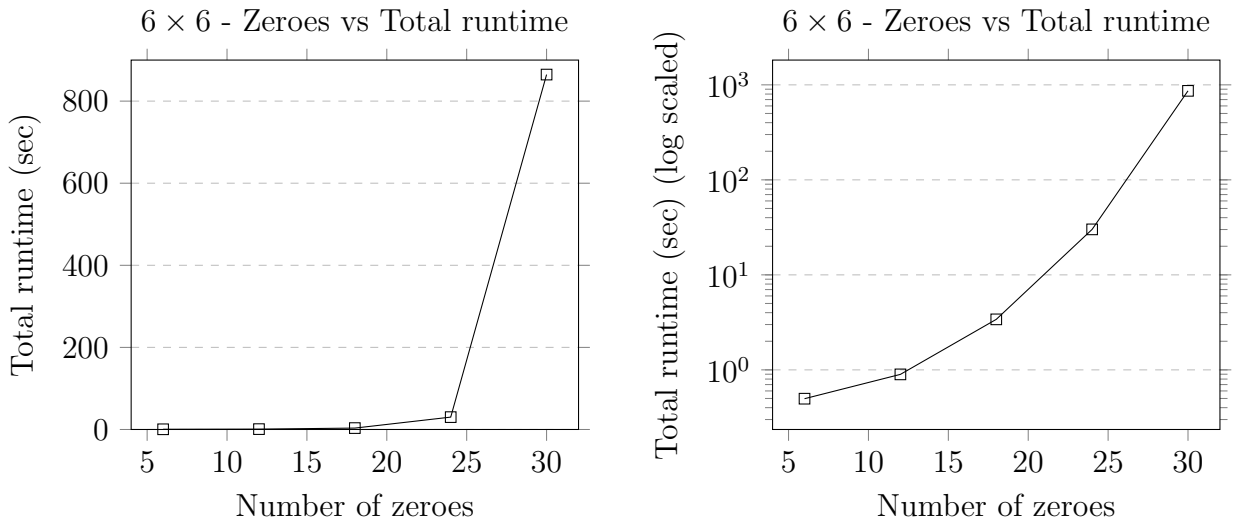

Figure 3.5: 6×6 runtime and number of zeroes

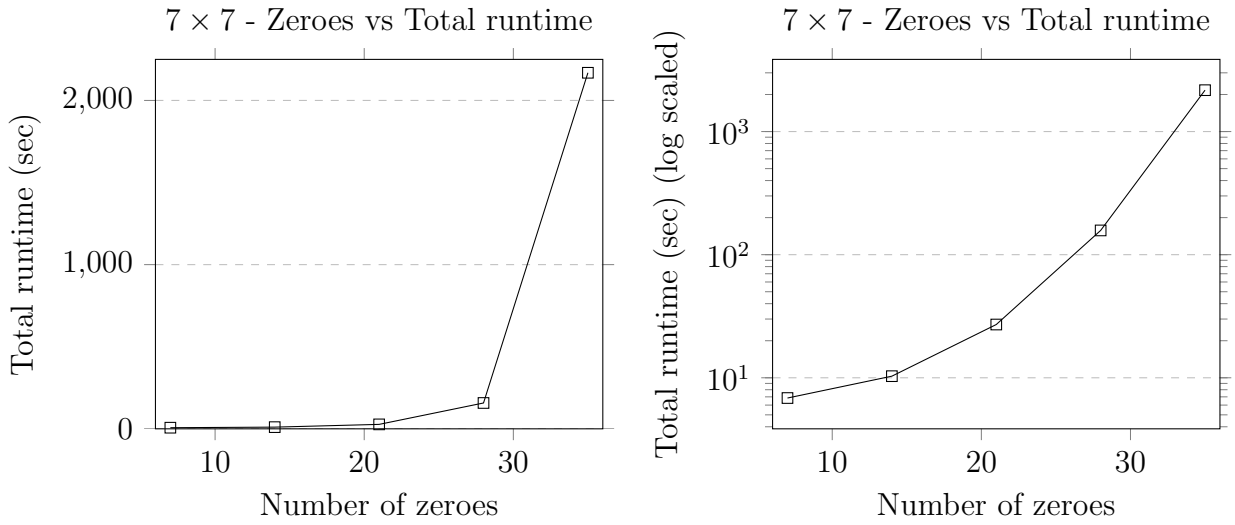

Figure 3.6: 7×7 runtime and number of zeroes

It appears that the runtime scales faster than exponential in the number of zeroes. It looks quadratic in all four log scaled graphs. We suspected it was exponential in the number of zeroes due to the runtime of the second step of our algorithm. However, we cannot explain why all our tests conclude it is more than exponential.

Comparison with naive brute force We run the brute force algorithm on the small input from the real world runtime test (with fifty matrices per input). Our results are in table 3.4.

Size	Total time of skeleton solver (sec)	Total time of brute force (sec)
4×4	0.052877	0.053791
4×5	0.279887	0.269775
5×5	2.13644	8.60843
5×6	57.8506	379.877
6×6	80.7195	$>$ 3600 (terminated)

Table 3.4: Brute force runtime comparison

Size (zeroes)	Total time of skeleton solver (sec)	Total time of brute force (sec)
4×4 (4)	0.007765	0.002092
4×4 (8)	0.022636	0.011281
4×4 (12)	0.190477	0.170938
5×5 (5)	0.046834	0.002805
5×5 (10)	0.115808	0.046162
5×5 (15)	0.556503	1.45705
5×5 (20)	10.6249	47.4835
6×6 (6)	0.497128	0.0057
6×6 (12)	0.896287	0.207622
6×6 (18)	3.40114	13.458
6×6 (24)	30.1174	887.089
7×7 (7)	6.84749	0.009881
7×7 (14)	10.3064	0.938688
7×7 (21)	27.0798	124.533

Table 3.5: Brute force runtime on number of zeroes comparison

They are similar in the small instances. However, the skeleton solver easily outdoes the brute force algorithm in the larger ones. We terminated the brute force algorithm for the 6×6 case after an hour. The file was an eighth complete so it may have taken eight hours if we did not. These tests are consistent with our claim that the skeleton solver performs much better than the brute force enumeration algorithm for random input. We show this difference is because the skeleton solver beats the brute force algorithm on instances with many zeros.

We run the brute force algorithm on the input from the zeroes test, skipping the ones with too many zeroes. Table 3.5 summarizes the results.

The brute force algorithm is faster for the instances with fewer than 15 zeroes. It is only faster by a negligible amount for the small instances. However, it is much faster for the 7×7 case although the skeleton solver's runtime is still acceptable. This is due to the lower bound of the skeleton solver. We recommend using the brute force algorithm if the input is big and

contains few zeroes. As the number of zeroes increase, the brute force algorithm's runtime balloons as expected. The skeleton solver vastly outperforms the brute force algorithm for those instances.

In addition to this runtime comparison, we did a check for correctness. All outputs match which indicates the skeleton solver is likely correct.

CHAPTER 4: CONCLUSION

We attempted to characterize the infeasible instances of the 1-Dollo phylogeny problem and proposed the skeleton solver as a algorithm for the counting version of the 1-Dollo phylogeny problem. Unfortunately we were not able to resolve hardness for 1-DP, but we found potential avenues during our experiment.

Our analysis of the infeasible instances showed that unlike the perfect phylogeny problem, the forbidden matrices in the 1-Dollo case does not appear to have a simple characterization. Moreover, we suspect the number of forbidden matrices that is not a cycle graph in the bipartite graph representation. is unbounded as well. There may be a simple characterization of these graphs in the bipartite graph representation, but we were unable to establish one. This appears to be a promising direction to search for those with a solid background in graph theory. Another follow up is to redo the computation, but remove all symmetries this time. A potential way to do this is by using the Pólya enumeration theorem.

Our skeleton solver algorithm is the first nontrivial algorithm proposed for the counting version of the 1-Dollo phylogeny problem. It uses our concept of a skeleton to efficiently enumerate all phylogenies. We showed it runs fast and beats the naive brute force enumeration algorithm on practical instances. The need for such algorithm arises from how unlikely a single phylogeny is the correct mutation history for a set of observations. With the set of all possible phylogenies, one can analyze the structure and patterns of those to find the most likely ones to model the observations. This can lead to better downstream inferences and therefore, more effective treatments.

The skeleton solver can be used to gain insight on the structure of feasible instances for 1-DP. For example, one can find instances which only have a few phylogenies. Then see if any patterns exist in those and check if they persist on random instances. We investigated a few instances and found the two optimizations for the skeleton, but did not find any conclusive patterns that did not depend on the input.

We end with questions that may speed up the skeleton algorithm. The big one we encountered and could not explain is for a fixed size, why does the runtime grow faster than exponential on the number of zeroes? Answering this may help us identify and minimize the bottleneck. Another question is can we reduce the number of trees we need to create in the Prufer sequence? This is a lower bound on the runtime on the algorithm, which caused it to run slower than the brute force algorithm on big instances with few zeroes. Optimizing this step can lessen the gap. Finally, can the taxon tell us anything? Our optimizations were based on the characters. Perhaps there are constraints between taxa in the phylogeny. We

hope resolving any of these will make the algorithm acceptable for bigger instances.

REFERENCES

[Bonizzoni et al.(2012)] Paola Bonizzoni et al. The binary perfect phylogeny with persistent characters. *Theoretical Computer Science*, 454:51 – 63, 2012. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2012.05.035. URL `http://www.sciencedirect.com/science/article/pii/S0304397512005294`. Formal and Natural Computing.

[Dollo(1893)] Louis Dollo. Les lois de l'évolution. In *Bull. Soc. Belge Geol. Pal. Hydr.*, volume 7, pages 164–166. 1893.

[El-Kebir(2018)] Mohammed El-Kebir. SPhyR: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 09 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty589. URL `https://doi.org/10.1093/bioinformatics/bty589`.

[Estabrook et al.(1975)] G.F. Estabrook et al. An idealized concept of the true cladistic character. *Mathematical Biosciences*, 23:263272, 04 1975. doi: 10.1016/0025-5564(75)90040-1.

[Farris(1977)] James S. Farris. Phylogenetic Analysis Under Dollo's Law. *Systematic Biology*, 26(1):77–88, 03 1977. ISSN 1063-5157. doi: 10.1093/sysbio/26.1.77. URL `https://doi.org/10.1093/sysbio/26.1.77`.

[Fernàndez-Baca(2001)] David Fernàndez-Baca. The perfect phylogeny problem. pages 203–234, 2001.

[Gusfield(1991)] Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991. doi: 10.1002/net.3230210104. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230210104`.

[Heppner(1984)] G. H. Heppner. Tumor heterogeneity. *Cancer Res.*, 44(6):2259–2265, Jun 1984.

[Kerber(1988)] Adalbert Kerber. Experimentelle mathematik. *Séminaire Lotharingien de Combinatoire*, 19:77–83, 1988.

[Nowell(1976)] Peter C. Nowell. The clonal evolution of tumor cell populations. *Science*, 194(4260):23–28, 1976. ISSN 00368075, 10959203. URL `http://www.jstor.org/stable/1742535`.

[Pradhan and El-Kebir(2018)] Dikshant Pradhan and Mohammed El-Kebir. On the non-uniqueness of solutions to the perfect phylogeny mixture problem. In Mathieu Blanchette and Aïda Ouangraoua, editors, *Comparative Genomics*, pages 277–293, Cham, 2018. Springer International Publishing. ISBN 978-3-030-00834-5.

[Prüfer(1918)] Heinz Prüfer. Neuer beweis eines satzes ber permutationen. In *Arch d. Math. u. Phys.*, volume 27, pages 742–744. 1918.

[Schwartz and Schaffer(2017)] Russell Schwartz and Alejandro Schaffer. The evolution of tumour phylogenetics: principles and practice. *Nature Reviews Genetics*, 18, 02 2017. doi: 10.1038/nrg.2016.170.

[Tabassum and Polyak(2015)] D.P. Tabassum and K Polyak. Tumorigenesis: it takes a village. *Nature Reviews Cancer*, 15:473–483, 2015.