Leveraging constraints plus dynamic programming for the large Dollo parsimony problem

Junyan Dai
Department of Computer Science, University of Maryland, College Park, MD, USA

Tobias Rubel
Department of Computer Science, University of Maryland, College Park, MD, USA

Yunheng Han
Department of Computer Science, University of Maryland, College Park, MD, USA

Erin K. Molloy[1]
Department of Computer Science, University of Maryland, College Park, MD, USA

Abstract

The last decade of phylogenetics has seen the development of many methods that leverage constraints plus dynamic programming. The goal of this algorithmic technique is to produce a phylogeny that is optimal with respect to some objective function and that lies within a constrained version of tree space. The popular species tree estimation method ASTRAL, for example, returns a tree that (1) maximizes the quartet score computed with respect to the input gene trees and that (2) draws its branches (bipartitions) from the input constraint set. This technique has yet to be used for classic parsimony problems where the input are binary characters, sometimes with missing values. Here, we introduce the clade-constrained character parsimony problem and present an algorithm that solves this problem in polynomial time for the Dollo criterion score. Dollo parsimony, which requires traits/mutations to be gained at most once but allows them to be lost any number of times, is widely used for tumor phylogenetics as well as species phylogenetics, for example analyses of low-homoplasy retroelement insertions across the vertebrate tree of life. Thus, we implement our algorithm in a software package, called Dollo-CDP, and evaluate its utility in the context of species phylogenetics using both simulated and real data sets. Our results show that Dollo-CDP can improve upon heuristic search from a single starting tree, often recovering a better scoring tree. Moreover, Dollo-CDP scales to data sets with much larger numbers of taxa than branch-and-bound while still having an optimality guarantee, albeit a more restricted one. Lastly, we show that our algorithm for Dollo parsimony can easily be adapted to Camin-Sokal parsimony but not Fitch parsimony.

2012 ACM Subject Classification Applied computing → Molecular evolution

Keywords and phrases phylogenetics, parsimony, Dollo, Camin-Sokal, dynamic programming, constraints

Digital Object Identifier 10.4230/LIPIcs.WABI.2023.6

Funding This work was financially supported by the State of Maryland. All computational experiments were performed on the compute cluster for the Center for Bioinformatics and Computational Biology at the University of Maryland, College Park.

Acknowledgements We thank the anonymous reviewers for constructive feedback.

1 Introduction

The last decade of phylogenetics has seen the development of many methods that leverage constraints plus dynamic programming (CDP). The goal of CDP is to produce a phylogeny that is optimal with respect to some objective function and that lies within a constrained

[1] Corresponding author

version of tree space. To our knowledge, the first method based on CDP was introduced in 2000 by Hallet and Lagergren [20] for gene tree parsimony, which seeks a species tree that minimizes the number of events (e.g., duplications and losses) needed to explain the input gene trees (also see the related results presented at WABI 2017 [1, 2]). Since its introduction, CDP has been leveraged for a variety of optimization problems, including minimizing deep coalescence [42], maximizing quartet support [6, 29] (see [44] for extensions to multi-copy genes), and maximizing bipartition support [37] (see [32] for extensions to multi-copy genes). All of these optimization problems take gene trees as input and seek a species tree that minimizes the dissimilarity between it and the input gene trees or alternatively maximizes the similarity (note that species trees depict the evolutionary history of species, whereas gene trees depict the evolutionary history of orthologous genomic regions, referred to as genes).

That CDP is so widely utilized in phylogenetics is likely due to it being possible to build effective constraints in practice. The constraints are effective if (nearly) optimal solutions lie within the constrained solution space and this space is small enough to enable efficient running times. As an example, the species tree estimation method `ASTRAL` solves the *bipartition-constrained maximum quartet support supertree problem* [29]. The first version of `ASTRAL` [29] formed the constrained solution space from the set of all bipartitions (i.e., branches) found in the input gene trees. The newer versions of `ASTRAL` [30, 43] allow extra bipartitions to be included as constraints with the goal of improving accuracy. `FASTRAL` [10], on the other hand, aggressively limits the number of bipartitions added with the goal of improving runtime. Overall, the popularity of the `ASTRAL` family of methods is likely due to their speed and accuracy on practical inputs.

Given the success of CDP thus far, we explore the use of this technique for traditional parsimony problems where the input are binary characters, sometimes with missing values. The remainder of this paper is organized as follows. We begin with some notation and preliminaries in Section 2. In Section 3, we introduce the *clade-constrained character parsimony problem* and present an algorithm that solves this problem in polynomial time for the Dollo criterion score. We then show how our algorithm can easily be adapted to Camin-Sokal parsimony but not Fitch parsimony. Dollo parsimony, in particular, is widely used for tumor phylogenetics [3, 4, 7, 13] as well as species phylogenetics, for example analyses of low-homoplasy retroelement insertions across the vertebrate tree of life. Prior studies have leveraged Dollo parsimony to analyze higher-level clades of birds (e.g., *Palaeognathe* [8]) and mammals (e.g., *Laurasiatheria* [11, 12]), in addition to clades at the family and genus levels (e.g., rorquals [26], mouse-eared bats [23, 25], and primates [33]).

This motivated us to implement our algorithm for the Dollo criterion score in `Dollo-CDP`, an open-source software package available on Github (`https://github.com/molloy-lab/Dollo-CDP`). In Section 4, we describe an experimental study evaluating `Dollo-CDP` on real and synthetic data sets from species phylogenetics in comparison to branch-and-bound and heuristic search. Our results, presented in Section 5, reveal that `Dollo-CDP` can improve upon heuristic search from a single starting tree, often recovering a better scoring tree. Moreover, `Dollo-CDP` scales to data sets with much larger numbers of taxa than branch-and-bound while still having an optimality guarantee, albeit a more restricted one. We conclude in Section 6 by discussing limitations and opportunities for future research.

2 Preliminaries and Background

Before introducing the clade-constrained Dollo parsimony problem, we review some preliminaries on phylogenetic trees, characters, and parsimony approaches.

2.1 Phylogenetic Trees

A *phylogenetic tree* T is an acyclic graph whose leaves (i.e., vertices with degree one) are bijectively labeled by a set S of species (note that in the context of tumor phylogenetics the leaves may be labeled by cells in a tumor). For convenience and simplicity of notation, we treat leaves and species as being interchangeable. We use $L(T)$, $V(T)$, and $E(T)$ to denote the leaf set, vertex set, and edge set of T, respectively.

Phylogenetic trees can be either *unrooted* or *rooted*; for the former the graph is undirected and for the latter the graph is directed, with edges orientated away from the root (a special vertex with in-degree zero). For rooted trees, we say that vertex u is an ancestor of v (or that v is a descendant of u) if u is on a directed path from the root to v. The *lowest common ancestor (LCA)* for a set V of vertices is the vertex that is the ancestor of all vertices in V that is farthest away from the root.

Unless otherwise noted, we will assume that all trees are *binary*. An unrooted tree is binary if all non-leaf vertices (called internal vertices) have degree three, and a rooted tree is binary if all non-leaf vertices have out-degree two (all non-root vertices have in-degree one). For rooted trees, we use $v.parent$ to indicate the parent of vertex v; similarly, we use $v.left$ and $v.right$ to denote the left and right children of vertex v, respectively. Sometimes it will be useful to restrict a tree T to a subset $X \subseteq S$ of leaves, meaning that each leaf in $S \setminus X$ is deleted from T and then all vertices with degree two are suppressed. There are three additional concepts for phylogenetic trees that will also prove useful later.

▶ **Definition 1** (Bipartition). *Each edge e in an unrooted phylogenetic tree T induces a* bipartition*, which splits the leaf set of T into two disjoint subsets whose union is the complete leaf set S. The bipartition $Bip(e) = X|Y$ is formed by deleting edge e but not its endpoints from T and assigning the leaves in one of the resulting subtrees to X and the leaves in the other to Y.*

▶ **Definition 2** (Clade). *Each vertex v in a rooted phylogenetic tree T induces a* clade*, denoted $Clade(v)$, which is simply the set of leaves that are descendants of v. A clade is* trivial *if it contains only a single element (as it must be associated with a leaf vertex) or if it contains all leaves (as it must be associated with the root vertex).*

▶ **Definition 3** (Subtree bipartition [24]). *Each internal vertex v in a rooted binary phylogenetic tree T induces a* subtree bipartition*, which partitions the leaf set of the subtree into two disjoint subsets whose union is $Clade(v)$. A subtree bipartition $SubBip(v) = X|Y$ is formed by setting X to be the leaves that are descendants of $v.left$ and Y to be the set of leaves that are descendants of $v.right$ (or vice versa).*

It is well established that an unrooted phylogenetic tree t is uniquely defined by its bipartition set $Bip(t) = \{Bip(e) : e \in E(t)\}$ and that a rooted phylogenetic tree T is uniquely defined by its clade set $Clade(T) = \{Clade(v) : v \in V(T)\}$ (see Chapters 2 and 3 in [39]). Similarly, we define the subtree bipartition set of T as $SubBip(T) = \{SubBip(v) : v \in V(T)\}$. Note that $X|Y \in SubBip(T)$ if and only if $\{X, Y, X \cup Y\} \subseteq Clade(T)$; thus, we can go back and forth between clades and subtree bipartitions.

2.2 Characters and Parsimony

A *character* c on species set S is a function mapping species in S to a state set, which is $\{0, 1\}$ for binary characters. For biological data, the `0` and `1` might refer to some feature of the genomic data with all species assigned the same state having the same feature. If `0`

indicates the ancestral state and `1` indicates the derived (i.e., mutated) state, we say the characters are *ordered*; otherwise, we say they are *unordered*. We describe biological data that are encoded as ordered binary characters in Section 4. These character matrices also include a third state `?` to indicate the state assignment is ambiguous or missing (in other words it could not be called as `0` or `1` for whatever reason).

For now, we assume that we are given a set $\mathcal{C}$ of binary characters with no missing values, and our goal is to find a phylogenetic tree T that best explains our data. To explain how character c evolves on a tree T on the same species set as c, we must assign states to the internal vertices of T. The quality of our explanation is determined by the number of substitutions, where a substitution is implied by any edge $e = (u, v) \in E(T)$ such that $c[u] \neq c[v]$ (and $c[u], c[v]$ are not the ambiguous state `?`). This brings us to the small and large parsimony problems.

▶ **Definition 4** (Fitch Parsimony Score). *Given an unrooted binary tree T and an unordered binary character c, both on species set S, the* Fitch parsimony score*, denoted $Fitch(T, c)$, is the minimum number of substitutions needed to explain the evolution of c on T. A* Fitch-labeling *for (T, c) is a function $\hat{c}$ mapping vertices in $V(T)$ to states in $\{0, 1\}$ so that $\hat{c}[l] = c[l]$ for all $l \in L(T)$ and the number of substitutions equals $Fitch(T, c)$.*

▶ **Problem 5** (Large Fitch Parsimony Problem). *The* large Fitch parsimony problem *takes as input a set $\mathcal{C}$ of unordered binary characters, each on species set S; the output is an unrooted binary tree T on S that minimizes $\sum_{c \in \mathcal{C}} Fitch(T, c)$.*

Although the Fitch parsimony score can be computed in polynomial-time [18], the large Fitch parsimony problem is NP-hard [19]. We now consider ordered characters and rooted phylogenetic trees, which enables us to distinguish between the $\texttt{0} \to \texttt{1}$ substitution (indicating a mutation is gained) and the $\texttt{1} \to \texttt{0}$ substitution (indicating a mutation is lost).

▶ **Definition 6** (Dollo and Camin-Sokal Parsimony Score). *Given a rooted binary tree T and an ordered binary character c, both on species set S, the* Dollo parsimony score*, denoted $Dollo(T, c)$, is the minimum number of losses needed to explain the evolution of c on T when at most one gain is allowed (note that sometimes the gains are also counted as part of the score). The* Camin-Sokal parsimony score*, denoted $CamSok(T, c)$, is the minimum number of gains needed to explain the evolution of c on T when losses are prohibited.*

Just as the Fitch parsimony score was used to define the Fitch-labeling and the large Fitch parsimony problem, we can define similar concepts for the Dollo and Camin-Sokal parsimony scores. The following result about Dollo-labelings is the basis of a linear-time algorithm presented by Bouckaert *et al.* [5] for computing $Dollo(T, c)$.

▶ **Theorem 7** (Theorem 1 in [5]). *Let T be a rooted, binary tree and let c be an ordered, binary character with no missing values, both on the same species set. A Dollo-labeling for (T, c) must assign state `1` to every internal vertex on a path from the LCA of all leaves assigned state `1` (including the LCA) to any leaf assigned state `1` and assign state `0` to all remaining vertices.*

It follows that a Dollo-labeling for (T, c) is unique and always exists if we allow the root to be assigned state `1`, in which case the gain must have occurred above the root so no gains are allowed on T (see Figure 1A for an example).

Lastly, the large Dollo and Camin-Sokal parsimony problems are NP-hard [9]. The existing methods for these problems, including those implemented in `PAUP*` [36] and `Phylip` [16], are based on either heuristic searches of tree space (which have no optimality guarantees)

or branch-and-bound (which is guaranteed to find an optimal solution). We describe these approaches further in Section 4 and refer the interested reader to [14, 15] for more information about parsimony problems and the related methodologies.

3 The clade-constrained large Dollo parsimony problem and a polynomial-time algorithm

We now introduce the *clade-constrained large Dollo parsimony* (CC-LDP) problem.

▶ **Problem 8** (Clade-constrained large Dollo parsimony problem). *The CC-LDP problem is defined by the following input and output.*
Input: *A set S of species, a set $\mathcal{C}$ of ordered binary characters (each on species set S), and a set Σ of clades (subsets of S)*
Output: *A rooted binary tree on species set S such that $\sum_{c\in\mathcal{C}} Dollo(T,c)$ is minimized and $Clade(T) \subseteq \Sigma$, if such a tree exists*

The CC-LDP problem has a solution provided there exists at least one binary tree T on S with $Clade(T) \subseteq \Sigma$. We present approaches for constructing Σ in Section 4 and evaluate their empirical performance in Section 5. In the remainder of this section, we assume that Σ is constructed from $\mathcal{C}$ in such a way that a solution to CC-LDP always exists. To present a polynomial-time algorithm for CC-LDP, a corollary of Theorem 7 will be useful.

▶ **Corollary 9.** *Let c be an ordered binary character on species set S (with no missing values), let T be a rooted binary tree on S, and let $\hat{c}$ be the unique Dollo-labeling for (T,c). Then for any internal vertex $v \in V(T)$, $\hat{c}[v]$ can be determined just by having knowledge of its subtree bipartition $SubBip(v)$; no other information about T is needed.*

Proof. Consider an arbitrary internal vertex $v \in V(T)$ that induces subtree bipartition $X|Y$. Without loss of generality, let X contain the leaves that are descendants of $v.left$, let Y contain the leaves that are descendants of $v.right$, and let Z contain all other leaves so $X|Y|Z$ is a partition of S. When c has no missing values, by Theorem 7, there exists a unique Dollo-labeling $\hat{c}$ for (T,c), where $\hat{c}[v] = 1$ if at least one of the following two cases holds (otherwise $\hat{c}[v] = 0$).

- **Case A:** Vertex v is the LCA of two leaves assigned state 1.
- **Case B:** Vertex v is on the path from the LCA of two leaves assigned state 1 to one of those two leaves.

Looking at subtree bipartition $SubBip(v) = X|Y$, case A holds if condition 1 below is true, and case B holds if at least one of conditions 2 and 3 below is true.

- **Condition 1:** There exists a leaf $x \in X$ and a leaf $y \in Y$ such that $c[x] = c[y] = 1$.
- **Condition 2:** There exists a leaf $x \in X$ and a leaf $z \in Z$ such that $c[x] = c[z] = 1$.
- **Condition 3:** There exists a leaf $y \in Y$ and a leaf $z \in Z$ such that $c[y] = c[z] = 1$.

Therefore, $\hat{c}[v] = 1$ if at least one of conditions 1–3 is true; otherwise $\hat{c}[v] = 0$. Conditions 1–3 can be evaluated so long as we know the subtree bipartition induced by v (given the character c and its complete leaf set S). ◀

▶ **Theorem 10.** *Let c be an ordered binary character on species set S, and let T be a rooted binary tree on S. If c has missing values, the Dollo-labeling for (T,c) may not be unique. However, we can define a unique Dollo-labeling $\hat{c}_*$ for (T,c) with the property that $\hat{c}_*[v]$ can be determined for any internal vertex $v \in V(T)$ just by having knowledge of its subtree bipartition $SubBip(v)$; no other knowledge of T is needed.*

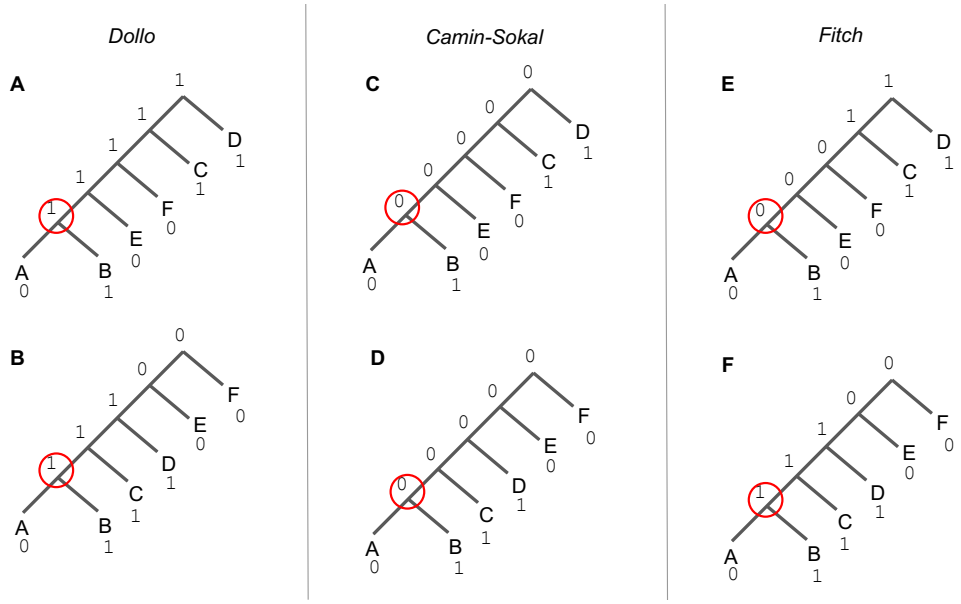

Figure 1 Let v be the internal vertex associated with the subtree bipartition $X|Y$, where $X = \{A\}$ and $Y = \{B\}$ (note that these vertices are circled in the trees above). Subfigures A and B show two different trees on the same species set with Dollo-labelings for the same character. The state assignment at v only requires us to know the subtree bipartition associated with v (Corollary 9). Vertex v is assigned state `1` because there is a leaf in Y assigned state `1` and a leaf in $S \setminus X \cup Y$ assigned state `1`. Subfigures C and D show two different trees with Camin-Sokal-labelings for the same character. The state assignment at v only requires us to know the clade associated with v (Corollary 16). Vertex v is assigned state `0` because there is a leaf in clade $X \cup Y$ assigned state `0`. Lastly, subfigures E and F show two different trees with Fitch-labelings for the same character. In subfigure E, the assignment of state `0` or `1` to v results in a score of two or three, respectively (so `0` is better). In subfigure F, the assignment of state `0` or `1` to v results in a score of three or two, respectively (so `1` is better). Thus, for the Fitch criterion score, the state assignment at v depends on more than the bipartition induced by the edge incident to v.

Proof. If c is allowed to have missing values, we compute the Dollo parsimony score *after* restricting T and c to the subset R of leaves that are *not* assigned the ambiguous state `?`. Letting $T|_R$ and $c|_R$ denote the restriction of T and c to R, respectively, we say that $\hat{c}$ is a Dollo-labeling for (T, c) if $Dollo(T, c) = Dollo(T|_R, c|_R)$.

We define a unique Dollo-labeling $\hat{c}_*$ for (T, c) with the property that it can be determined for any vertex $v \in V(T)$ with $SubBip(v) = X|Y$ by applying conditions 1–3 (see proof of Corollary 9) plus one additional condition:

Condition 0: For all leaves $l \in X \cup Y$, $c[l] =$ `?`.

If condition 0 is true, we set $\hat{c}_*[v] =$ `?`. Otherwise, we proceed in the usual fashion, setting $\hat{c}_*[v] =$ `1` if at least one of conditions 1–3 is true and setting $\hat{c}_*[v] =$ `0` otherwise.

We need to show that this procedure produces a valid Dollo-labeling for (T, c) for all $v \in V(T)$. We begin by classifying the vertices of T into three groups based on the formation of $T|_R$ described next.

1. First, we identify every edge incident to the root of a maximally-sized subtree of T whose leaves are all assigned state `?`. That is, we identify every edge $a \mapsto b \in E(T)$ such that all leaves in $Clade(b)$ are assigned state `?` *and* at least one leaf in $Clade(a)$ is assigned a non-ambiguous state. Let $\mathcal{E}$ denote the set of edges with this property.

2. Second, we delete each edge in $\mathcal{E}$ but not its endpoints from T. This produces a tree T' with no leaves assigned state `?` plus a collection $\mathcal{P}$ of subtrees of T whose leaves are all assigned state `?`.
3. Lastly, we form $T|_R$ by suppressing all vertices in T' with out-degree one.

The above procedure can be used to classify vertices in $V(T)$ into three groups by which we can build a valid Dollo-labeling $\hat{c}$ for (T, c) as follows.

- **Group 1** contains every vertex $v \in V(T)$ that maps to a vertex $w \in V(T|_R)$. For each vertex v in this group, we assign $\hat{c}[v] = \hat{c}|_R[w]$, where $\hat{c}|_R$ is the Dollo-labeling for $(T|_R, c|_R)$. The idea is to propagate the state assignments for $T|_R$ back to T. Now we need to assign states to the remaining vertices in T without changing the Dollo score.
- **Group 2** contains every vertex $v \in V(T)$ that maps to a vertex $w \in \cup_{t\in\mathcal{P}} V(t)$, meaning that v is in a maximally-sized subtree of T whose leaves are all assigned state `?`. For each vertex v in this group, we set $\hat{c}[v] =$ `?`. This state assignment does not change the Dollo score because substitutions are not counted on edges with at least one of their two endpoints assigned state `?`.
- **Group 3** contains every vertex $v \in V(T)$ that does not map to any vertex in $V(T|_R) \cup \cup_{t\in\mathcal{P}} V(t)$, meaning that the corresponding vertex must have been suppressed in step 3. Thus, v maps to an edge $e \in E(T|_R)$, and exactly one of v's children is the root of a maximally-sized subtree of T whose leaves are all assigned state `?`. A state assignment that does not change the Dollo score can be achieved by assigning every vertex v in this group that maps back to the same edge $e = s \mapsto t \in E(T|_R)$ the same state: either the state assigned to s or the state assigned to t.

Because there can be multiple ways to assign states to vertices in group 3, a unique Dollo labeling for (T, c) does not necessarily exist.

We previously proposed how to define a unique Dollo-labeling $\hat{c}_*$ for (T, c) with the property that it could be determined for any internal vertex $v \in V(T)$ with $SubBip(v) = X|Y$. We now verify that our procedure produces a valid Dollo-labeling by examining the outcomes of applying conditions 0–3 to vertices in groups 1–3.

- **Group 2:** Condition 0 will be true for vertex v if and only if v is in group 2; thus, applying condition 0 assigns state `?` correctly.
- **Group 1:** The outcomes of applying conditions 1–3 to any vertex v in group 1 will be the same as applying conditions 1–3 to the corresponding vertex w in $V(T|_R)$ because $SubBip(v) = SubBip(w)$ after we remove all leaves assigned state `?` from the $SubBip(v)$.
- **Group 3:** Each vertex v in group 3 maps to an edge $e = s \mapsto t \in E(T|_R)$. For each of these vertices v, the subtree bipartition $SubBip(v) = X|Y$ will have *either* all leaves in X assigned state `?` *or* all leaves in Y assigned state `?`. Thus, for any two vertices v_1 and v_2 in group 3 that map to the same edge $e = s \mapsto t$, $SubBip(v_1) = SubBip(v_2)$ after all leaves assigned `?` are removed. It follows that v_1 and v_2 will be assigned the state when applying conditions 1–3. Lastly, we need to show that the state assigned v_1 (call v) will be the same as either the state assigned to s or the state assigned to t. For simplicity, assume v induces $SubBip(v) = X|Y$ and that all leaves in X are assigned state `?`; we can now check the outcomes of applying conditions 1–3 to v.
  - Condition 1 is false because there does not exist a leaf $x \in X$ such that $c[x] =$ `1`.
  - Condition 2 is false for the same reason.
  - Condition 3 may be either true or false.
    - If condition 3 is true for v, then v is assigned state `1`. Because condition 3 is true for v, there exists a leaf $y \in Y$ and a leaf $z \in Z$ such that $c[x] = c[z] =$ `1`. Thus, t is on the path from the $LCA(y, z)$ to leaf y. It follows that t is also assigned state `1`.

- If condition 3 is false for v, then v is assigned state `0`. Because condition 3 is false for v, at least one of the following statements is true.
  - For all $y \in Y$, $c[y] \neq$ `1`. In this case, t is also assigned state `0`.
  - For all $z \in Z$, $c[z] \neq$ `1`. In this case, s is also assigned state `0`.

  The above logic can be applied if all leaves in Y are assigned state `?` (we just swap our arguments for conditions 2 and 3, replacing set Y with X).

This proves our claim. Note that our procedure still works when c has no missing values because all vertices in T will be in group 1. ◀

For a set $\mathcal{C}$ of k characters, each on a set S of n species, the proof of Theorem 10 gives an $O(nk)$ algorithm for determining a unique Dollo-labeling $\hat{c}$ for *all* characters in $\mathcal{C}$ at vertex v provided we know $SubBip(v)$. We refer to this procedure as `GetState` (see Algorithm 1 for details). This relationship between the subtree bipartitions of a tree and its Dollo-labeling brings us back to the CC-LDP problem, where the solution space is constrained by a set of clades, which in turn constrains the subtree bipartitions of any solution and thus its Dollo-labeling.

▶ **Corollary 11.** *Consider the set $\mathcal{T}$ of all solutions to CC-LDP given species set S, character set $\mathcal{C}$, and clade set Σ. Let $\mathcal{STB} = \{X|Y : X, Y, X \cup Y \in \Sigma\}$ be the set of all subtree bipartitions that can be formed from Σ, and let $\mathcal{STB}(A) = \{X|Y \in \mathcal{STB} : X \cup Y = A\}$ be the subset of subtree bipartitions in $\mathcal{STB}$ that are associated with clade A. Then, the unique Dollo-labeling at internal vertex v in an arbitrary phylogenetic tree $T \in \mathcal{T}$ that induces clade A must be in the set $Lab(A) = \{\texttt{GetState}(X|Y, S, \mathcal{C}) : X|Y \in \mathcal{STB}(A)\}$. Note that if v is a leaf, $Lab(A)$ is simply given by the input character set $\mathcal{C}$.*

This easily follows from Theorem 10. We refer to $Lab(A)$ as the *allowed* state assignments for clade A and $\mathcal{STB}$ as the *allowed* subtree bipartitions for clade A. These quantities can be precomputed or computed on the fly. Whenever this is done, we also compute the set $\mathcal{STB}(A, st)$ of subtree bipartitions $X|Y$ such that $X \cup Y = A$ and $\texttt{GetState}(X|Y, S, \mathcal{C}) = st$.

Lastly, if we know the state assignments for some vertex v as well as its children $v.left$ and $v.right$, it is possible for us to compute the number of losses that occur on the outgoing edges of v. We simply need to count the number of times v is assigned state `1` and $v.left$ is assigned state `0`, repeating for $v.right$. This can be done if $O(k)$ time. We refer to this procedure as `CountLosses` (see Algorithm 2 for details). Now we are ready to present the dynamic programming algorithm for CC-LDP.

▶ **Dynamic Programming 12.** *Let $Dollo[A, st]$ be the smallest number of losses for any pair $(t_A, \hat{c}_A)$ such that t_A is a rooted binary tree on leaf set A that draws all its clades from Σ and $\hat{c}_A$ is an assignment of states to all vertices of t_A constrained by Corollary 11 and returning st for the root of t_A (note that these requirements imply $A \in \Sigma$ and $st \in Lab(A)$). The quantity $Dollo[A, st]$ can be computed with dynamic programming as follows.*
Base Case: *Clade A contains a single species, i.e., $|A| = 1$.*

$$Dollo[A, st] := 0$$

Recurrence: *Clade A contains multiple species, i.e., $|A| > 1$.*

$$Dollo[A, st] := \min_{X|Y \in \mathcal{STB}(A,st), St_X \in Lab(X), St_Y \in Lab(Y)} Dollo[X, St_X] + Dollo[Y, St_Y] + \texttt{CountLosses}(st, St_X, St_Y)$$

The Dollo score of any solution to CC-LDP equals $\min_{st \in Lab(S)} Dollo[S, st]$*, and a solution can be recovered by backtracking.*

▶ **Theorem 13.** *The dynamic programming algorithm above correctly solves CC-LDP.*

Proof. **Base case:** The base case for $Dollo[A, st]$ is trivial because when $|A| = 1$ there is only one rooted, binary tree possible: a single leaf assigned the states given by the input character set $\mathcal{C}$. There are no edges and thus no losses, so $Dollo[A, st] = 0$.

Induction step: Now we consider the case where $|A| > 1$. By the induction hypothesis, we assume that we have correctly solved $Dollo[X, St_X]$ and $Dollo[Y, St_Y]$. Let $(t_X, \hat{c}_X)$ be an arbitrary solution to subproblem $Dollo[X, St_X]$, implying that (1) t_X is a rooted binary tree on leaf set X with $Clade(t_X) \subseteq \Sigma$, (2) $\hat{c}_X$ is an assignment of states to all vertices of t_X constrained by Corollary 11 and returning St_X for the root of t_X, and (3) the number of losses for $(t_X, \hat{c}_X)$ equals $Dollo[X, St_X]$, which is the minimum number of losses for any pair (tree and state assignment) that satisfies both (1) and (2). Similarly, let $(t_Y, \hat{c}_Y)$ be an arbitrary solution to subproblem $Dollo[Y, St_Y]$. Note that by (1), $X, Y \in \Sigma$ and that by (2), $St_X \in Lab(X)$ and $St_Y \in Lab(Y)$.

Let $(t, \hat{c})$ be formed by connecting $(t_X, \hat{c}_X)$ and $(t_Y, \hat{c}_Y)$ at their roots and assigning state st to the new root. The pair $(t, \hat{c})$ is a candidate solution for subproblem $Dollo[A, st]$ provided that $X|Y \in \mathcal{STB}(A)$ (so requirement 1 is satisfied) and $st = \texttt{State}(X|Y, S, \mathcal{C}) \in Lab(A)$ (so requirement 2 is satisfied). These two requirements can be summarized as $X|Y \in \mathcal{STB}(A, st)$. For this candidate solution, the number of losses equals $Dollo[X, St_X] + Dollo[Y, St_Y] + \texttt{CountLosses}(st, St_X, St_Y)$, where the last term gives the number of losses for the new edges.

Now we need to consider requirement 3. Specifically, we need to check whether a better score (i.e., lower number of losses) can be obtained from any other candidate solution. Keeping clades X and Y, other candidate solutions can be formed by all other ways of selecting $St_X \in Lab(X)$ and $St_Y \in Lab(Y)$. Moreover, this process can be repeated for the other allowed subtree bipartitions for clade A that produce state assignment st (i.e., all other ways of selecting $X|Y \in \mathcal{STB}(A, st)$). Any other possibilities will violate (1) and/or (2), and thus will not produce valid candidate solutions. Thus, our recurrence is correct.

Recurrence is solvable by dynamic programming: In our recurrence, $Dollo[A, st]$ only depends on subproblems: $Dollo[X, St_X]$ and $Dollo[Y, St_Y]$ for all $X|Y \in STB(A, st)$, $St_X \in Lab(X)$, and $St_Y \in Lab(Y)$. Since $|X|$ and $|Y|$ must be less than $|A|$, solving subproblems in order of clade cardinality will guarantee that all trivial subproblems are solved first (hitting the base cases) and that all subproblem dependencies are satisfied moving forward (because there are no dependencies between subproblems corresponding to the same clade but different state assignments at the root). Thus, we can use dynamic programming to solve this recurrence.

Putting it all together: We compute $Dollo[S, st]$ for all $st \in Lab(S)$, recording a subproblem for which the number of losses is minimized. Backtracking gives an arbitrary solution to this subproblem, which is also a solution to CC-LDP by (1), (2), and (3). ◀

▶ **Theorem 14.** *The runtime of the dynamic programming algorithm is polynomial in the number* n *of species, the number* k *of characters, and the number of clades in* Σ*.*

Proof. We need to show that the number of subproblems is polynomial and that each subproblem can be solved in polynomial time.

The dynamic programming matrix has two dimensions. The first corresponds to clade $A \in \Sigma$, and the second corresponds to an allowed state assignment $st \in Lab(A)$. The former is clearly $O(|\Sigma|)$. The latter is also $O(|\Sigma|)$ because $|Lab(A)|$ has an upper bound of $|\mathcal{STB}(A)|$,

which in turn has an upper bound of $|\Sigma|$. This can be seen by noting that all allowed subtree bipartitions for A can be identified by looping over $X \in \Sigma$ and checking whether $X \subset A$ and $A \setminus X \in \Sigma$ (see Algorithm 3 for details). Thus, the number of subproblems is $O(|\Sigma|^2)$. Note that to perform the traceback, we also need to store pointers back to the two child subproblems for every subproblem.

Subproblem $Dollo[A, st]$ can be solved by considering all candidate solutions taking $X|Y \in \mathcal{STB}(A, st)$, $St_X \in Lab(X)$, and $St_Y \in Lab(Y)$. All of these sets are bounded above by $|\Sigma|$, so their product is bounded. For each candidate solution, we must execute `CountLosses` and sum three terms together; this can be done in $O(k)$ time. We also do some related bookkeeping that is polynomial time. As an example, for each subtree bipartition $X|Y \in \mathcal{STB}(A)$, we need to compute its state assignment with `GetStates` and then add it to the set $\mathcal{STB}(A, St_A)$; this can be done in $O(nk)$ time. It follows that the worst-case runtime of our algorithm is polynomial, albeit a high degree one (see Algorithm 4 for details). ◀

3.1 Extension to Camin-Sokal Parsimony

Our results for Dollo can be extended to Camin-Sokal parsimony. Specifically, we can define the *clade-constrained large Camin-Sokal parsimony* (CC-LCSP) problem, in the natural way. We can also extend our algorithm by replacing the `CountLosses` function (Algorithm 2) to count gains $0 \to 1$ instead and by redefining the `GetStates` function (Algorithm 1) based on the following result.

▶ **Theorem 15.** *Let T be a rooted binary tree, and let c be an ordered binary character (with no missing values), both on the same species set. A Camin-Sokal-labeling for (T, c) must assign state `0` to every internal vertex that is an ancestor of a leaf assigned state `0` and assign state `1` to all remaining vertices.*

The assignment of state `0` can be shown to be correct by contradiction. The assignment of state `1` to vertex v only occurs when all leaves below v are assigned state `1`; thus, labeling v with state `1` does not increase the number of gains needed to explain the data. It follows that a Camin-Sokal-labeling for (T, c) always exists and is unique.

▶ **Corollary 16.** *Let c be an ordered binary character on species set S, let T be an arbitrary rooted binary tree on S, and let $\hat{c}$ be the unique Camin-Sokal-labeling for (T, c). Then for any internal vertex v in T, $\hat{c}[v]$ can be determined just by having knowledge of $Clade(v)$; no other knowledge of T is needed. Moreover, if c is allowed to have missing values, a unique Camin-Sokal-labeling can be found in a similar fashion.*

Proof. Consider an internal vertex v in T that induces clade A. By Theorem 15, $\hat{c}[v] =$ `0` if the following case holds (otherwise $\hat{c}[v] =$ `1`)

- **Case C:** Vertex v is an ancestor of at least one leaf assigned state `0`.

Looking at clade $Clade(v) = A$, case C holds if condition 4 below is true.

- **Condition 4**: There exists a leaf $a \in A$ such that $c[a] =$ `0`.

To summarize, $\hat{c}[v] =$ `0` if condition 4 is true; otherwise $\hat{c}[v] =$ `1`. Condition 4 can be evaluated so long as we know the clade induced by v (given the character c and its complete leaf set S). This proves our first claim.

We now allow for missing values as discussed in Theorem 10, except applying condition 4 instead of conditions 1–3 to vertices in groups 1 and 3 (recall that condition 0 will be true if and only if a vertex is in group 2). Applying condition 4 to vertices in group 1 will simply propagate state assignments from $T|_R$ back to T. For vertices in group 3, we need to show that every vertex v that maps to the same edge $e = s \mapsto t \in E(T|_R)$ will be assigned the

same state: either the state assigned to s or the state assigned to t (so there may be multiple state assignments that achieve the optimal score). For each of these vertices v, the subtree bipartition $SubBip(v) = X|Y$ will have *either* all leaves in X assigned state `?` *or* all leaves in Y assigned state `?`. Thus, for any two vertices v_1 and v_2 in group 3 that map to the same edge $e = s \mapsto t$, $Clade(v_1) = Clade(v_2)$ after all leaves assigned `?` are removed. It follows that v_1 and v_2 will be assigned the same state when applying condition 4. Now we just need to check the outcomes of applying condition 4 to v_1 (call v).

- If condition 4 is true for v, it is true for both s and t. Thus, v, s, t are all assigned state `0`.
- If condition 4 is false for v, it is false for t. Thus, v and t are both assigned state `1`.

This proves our second claim. Note that our procedure still works when c has no missing values because all vertices in T will be in group 1. ◀

Although our algorithm for the large Dollo parsimony problem can be extended to Camin-Sokal parsimony, extensions to Fitch parsimony are not so straight-forward. Recall that the large Fitch parsimony problem takes unordered binary characters as input and seeks an unrooted binary tree to minimize the Fitch score. Both trees in Figure 1E–F when unrooted have an edge that induces bipartition $A, B|C, D, E$; however, the Fitch-labeling of the vertex incident to this edge and adjacent to leaves A and B depends on the remainder of the tree (or at least requires more information about the tree than a single bipartition).

4 Experimental Study

We now describe an experimental study evaluating our dynamic programming algorithm for CC-LDP against traditional methods for parsimony: heuristic search and branch-and-bound. All analysis scripts and data sets simulated for this study are available on Github (`https://github.com/molloy-lab/dollo-study`).

4.1 Character Data Sets

We evaluate methods in the context of species tree estimation under the infinite sites plus neutral Wright-Fisher (IS+nWF) model [17, 40]. Under the infinite sites model, characters evolve without homoplasy, meaning parallel mutations and reversals are prohibited. Some types of retroelement insertions, like L1 in mammals [22], are typically assumed to evolve with little homoplasy [34]. The idea is that two insertions are unlikely to occur at exactly the same position in the genome (so no parallel evolution) and that insertions are unlikely to be *precisely excised* (so no reversals) [34]. Note that the presence/absence of an insertion corresponds to ancestral/derived states so these characters are ordered.

Characters that evolve without homoplasy would result in a perfect phylogeny; however, this ignores population-level processes. For sexually reproducing organisms, insertions arising in egg or sperm cells are transmitted from parent to offspring. Thus, the mutation is polymorphic in the population when it arises and its frequency in the population changes randomly assuming neutral evolution (note that the population structure is governed by the species tree). To summarize, an insertion is gained ($0 \to 1$) exactly once but then it can be lost ($1 \to 0$) due to genetic drift. These rules are suitable for Dollo parsimony, and indeed, Dollo parsimony has been used to estimate species trees from low-homoplasy retroelement insertions in prior studies (e.g., [11, 12, 33, 26, 23, 25]). Here, we re-analyze two retroelement presence/absence data sets; we also benchmark methods on a collection of synthetic data sets simulated under the IS+nWF model.

Biological Data Sets

The *Myotis* data set from [25] has 11 taxa and 10,595 characters. Each character represents the presence/absence of a Ves SINE (short interspersed nuclear element) insertion at an orthologous position across the species' genomes. No character states are ambiguous, and all characters are parsimony-informative, specifically there are at least two `0`'s and at least two `1`'s. The original analysis of this data set included maximum parsimony using branch-and-bound with the Dollo criterion score (Figure 2 in [25]).

The *Palaeognathe* data set from [8] has 13 taxa and 4,301 parsimony-informative characters (note that over 18% of the character states in this matrix are ambiguous). Each character represents the presence/absence of a CR1 LINE (long interspersed nuclear element) insertion at an orthologous position across the species' genomes. The original analysis of this data set did not include Dollo parsimony; rather insertions were used to corroborate a species tree estimated from (estimated) gene trees with `ASTRAL` and `MP-EST` [27].

Simulated Data Sets

All synthetic data sets used in our study were simulated under an approximation to the IS+nWF model using `ms` [21]. The simulation requires a model species tree. At the high-level, a gene genealogy is simulated within the model species tree under the coalescent and then a mutation arises on a branch of the genealogy so all taxa that are descendants have the mutation (all other taxa do not). This process is repeated, producing a collection of binary characters that evolved independently within the model species tree. We only utilize the parsimony-informative characters, varying the total number of characters given to methods as input from 500 to 50,000.

The first collection of synthetic data sets are taken from Molloy *et al.* [31]. These data sets were simulated given the *Palaeognathe* species tree estimated by Cloutier *et al.* [8] using `MP-EST`. The `ms` simulation was repeated 25 times to produce 25 replicate data sets. We created a second collection of synthetic data sets by taking the species trees generated in a prior study [30]. Specifically, Mirarab *et al.* [30] simulated species trees with varying numbers of taxa under the Yule model with `SimPhy` [28], setting the species tree height to 2 million generations and the effective population size to 200,000. This process was repeated 50 times for each number of taxa. We ran the `ms` simulation, described above, for the first 25 species trees with 10, 50, 100, and 200 ingroup taxa (and one outgroup taxa). This produced 25 replicate characters matrices for each number of taxa.

4.2 Methods

We evaluated four different methods for the large Dollo parsimony problem. All approaches implemented in `PAUP*` [36] were executed using version 4a168_centos64 (downloaded from `https://paup.phylosolutions.com`).

Branch-and-bound Implementation

Branch-and-bound performs an exhaustive search of tree space in a systematic fashion using the parsimony score of an initial tree to rule out parts of tree space that do not need to be searched. We used the implementation of branch-and-bound in `PAUP*` (see Section B.4 for command) and saved all optimal trees.

SlowH and FastH Implementations

`FastH` is a "fast heuristic search" that operates in two phases. First, a starting tree is constructed via random taxon addition, meaning that taxa are put in a random order and then iteratively added to the tree (note that each taxon is attached to an edge of the current tree so that the criterion score is maximized). This process is repeated ten times and then the best scoring tree is taken as the starting tree. Second, hill-climbing is performed from the starting tree with Tree Bisection and Reconnection (TBR) edit moves. `FastH` was implemented with `PAUP*` (see Section B.1 for the command; the reconnection limit was set to eight branches by default). The 100 best-scoring trees found during the heuristic search were saved for use with our dynamic programming method.

`SlowH` is a "slow heuristic search" that operates by performing 100 independent searches. In each search, a starting tree is built via random taxon addition and then hill climbing is initiated from the starting tree using TBR edit moves. `SlowH` was implemented with `PAUP*` (see Section B.3 for the command; again the reconnection limit was set to eight branches by default). All trees with the best criterion score found were saved.

Dollo-CDP Implementation

`Dollo-CDP` is an implementation of our dynamic programming algorithm for CC-LDP. This method requires not only a character matrix but also a set of clades to use as constraints. We evaluated two different approaches for generating the constraints, both of which rely on `ASTRAL-III` [43]. The idea is to give `ASTRAL-III` a set of trees from which it will generate a set of clusters (note that clusters are subsets of taxa, like clades).

Our two approaches differ in the set of trees given to `ASTRAL-III` as input. Our first approach gives `ASTRAL-III` the 100 best-scoring trees found by `FastH`. Our second approach gives `ASTRAL-III` the input characters reformatted as unrooted trees, as proposed by [35]. The idea is that each parsimony-informative character encodes an unrooted tree with exactly one internal branch, indicating the transition between taxa in state 0 to taxa in state `1` or vice versa. After the clusters are computed with `ASTRAL-III`, `Dollo-CDP` processes them, keeping only the clusters that form clades in a tree rooted at some set O of outgroup taxa (note that outgroups are typically available when using Dollo parsimony, as they are often used when calling variants and coding them as ancestral or derived). A cluster C produced by `ASTRAL-III` is added to the constraint (clade) set if either (1) $C \subseteq O$ or (2) $C \subseteq \{S \setminus O\}$.

When the outgroup is a single taxon, both of our approaches ensure a solution to CC-LDP exists. For the first approach, all trees produced by `FastH`, denoted $\mathcal{P}$, can be rooted at O; therefore, $\Sigma = \{Clade(T) : T \in \mathcal{P}\}$. The second approach guarantees a solution by virtue of how `ASTRAL-III` handles polytomies (i.e., vertices of degree greater than three). However, this produces a large number of clades, which, in turn, makes the second approach more computationally intensive. Consequently, we only apply it to the biological data sets.

The command for running `Dollo-CDP` is given in Section B.2. Users are responsible for providing trees for building constraints if using the first approach. They are also responsible for downloading and extracting `ASTRAL-III` into the `src` directory so that `Dollo-CDP` can find it. We used `ASTRAL-III` version 5.7.8 from Github (`https://github.com/smirarab/ASTRAL`).

4.3 Evaluation Metrics

All computational experiments were run on the compute cluster for the Center for Bioinformatics and Computational Biology at the University of Maryland, College Park. This is a homogenous compute cluster, with all compute nodes having dual socket AMD EPYC 7313

16-Core processors and two terabytes of memory. All methods were given access to 64 GB of memory, one CPU, and a maximum wallclock time of 24 hours (resources were managed by the SLURM submission system). We recorded the total wallclock time in minutes as well as the best Dollo parsimony score found. We added the runtime of `FastH` to the runtime of `Dollo-CDP`, when the former was used to construct the constraint set for the latter. Our method `Dollo-CDP` only counts losses and does not count gains, unlike `PAUP*`. To ensure scores were comparable, we recomputed the Dollo criterion score for all trees using `PAUP*` (see Section B.5 for the command).

5 Experimental Results

We now present the results of our experimental study on biological and synthetic data sets.

5.1 Results on biological data sets

For the *Myotis*, all methods completed in less than a second, except for `Dollo-CDP` when it used the characters to construct the constraint set Σ (this analysis took 54 seconds). All approaches recovered the same tree as branch-and-bound, which was the unique optimal solution (Figure 2A). For the *Palaeognathe* data set, all methods completed in less than 3 seconds (again `Dollo-CDP` using characters to construct constraints was the slowest). Branch-and-bound recovered 60 optimal trees, and the strict consensus had just three branches (Figure 2B), all of which are in the species tree estimated by Cloutier *et al.* [8] using `MP-EST`. All methods we ran recovered one of the 60 optimal trees. Thus, on the biological data sets, which had up to 13 taxa, the data sets were small enough to leverage branch-and-bound and there was not much difference between the tested methods.

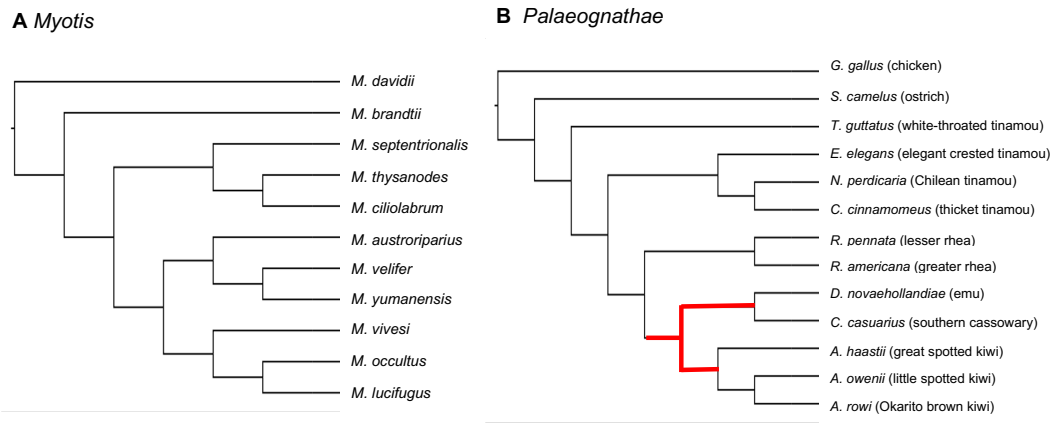

Figure 2 Subfigure A shows the tree returned by `Dollo-CDP` for the *Myotis* data set [25]. This is the same tree recovered by branch-and-bound. Subfigure B shows the tree returned by `Dollo-CDP` for the *Palaeognathae* data set [8]. A branch-and-bound analysis recovered 60 optimal trees (the `Dollo-CDP` tree is one of the 60). The three branches highlighted in red indicate the strict consensus of the 60 equally optimal trees.

5.2 Results on simulated data sets

Similar trends to the biological data sets were observed for the first collection of synthetic data sets, which were simulated from a *Palaeognathe* species tree by Molloy *et al.* [31]. For

all but one replicate, all methods recovered trees with the same Dollo parsimony score as branch-and-bound (which typically recovered one or two equally optimal trees). For the remaining replicate, `FastH` returned a tree with a slightly lower score than the other methods. Thus, `Dollo-CDP` slightly improved upon `FastH` in terms of Dollo parsimony score for one replicate. Similar trends were observed for the second collection of simulated data sets with 10 ingroup taxa. Like the biological data sets, the analyses of synthetic data sets suggest that as long as the number of taxa is sufficiently small, all methods will produce similar results and compare favorably to branch-and-bound.

We were unable to run branch-and-bound on data sets with 50 or more taxa (specifically the jobs were killed due to our maximum wallclock time of 24 hours). For these data sets, we first looked at whether `Dollo-CDP` found trees with better scores than `FastH` (note that `Dollo-CDP` cannot be worse because it is given all trees produced by `FastH` as constraints–and thus is guaranteed to find a tree with at least as good of a score as the best one found by `FastH`). For 50 ingroup taxa, `Dollo-CDP` typically improved upon `FastH` for about half of the replicates (Table 1). For 100 and 200 ingroup taxa (and at least 5000 characters), `Dollo-CDP` nearly always improved upon `FastH`. Moreover, `Dollo-CDP` was quite fast with an average runtime less than 3 minutes for all data sets.

Next, we compared the performance of `Dollo-CDP` to `SlowH`. For 50 and 100 ingroup taxa, `Dollo-CDP` performed as well as `SlowH` in terms of parsimony scores and was faster, although `SlowH` still always completed in less than 16 minutes on average. For 200 ingroup taxa, `SlowH` took nearly 40 minutes on average for 5000-100,000 characters and over 1.5 hours for 100,000 characters. In contrast, `Dollo-CDP` always completed in less than 5 minutes. On the other hand, `SlowH` did recover better scoring trees in about one third of the replicates. To summarize, our results suggest that for larger numbers of taxa and larger numbers of characters, `Dollo-CDP` improves upon `FastH` and is faster than `SlowH`.

6 Discussion and Conclusion

We have introduced the clade-constrained large Dollo parsimony problem and presented a polynomial time algorithm that solves it. Although constraints and dynamic programming (CDP) have been a powerful combination in phylogenetics, to our knowledge this is the first attempt at using CDP for character parsimony. An important distinction between prior problems and character parsimony is the assignment of states at internal vertices required to compute the parsimony score of a tree. We found that Dollo as well as Camin-Sokal criterion scores have nice properties that make CDP possible (they also might make heuristic search quite effective in practice). These nice properties for state assignments did not easily extend to Fitch parsimony, so our algorithmic approach seems less favorable in this setting.

We implemented the CDP algorithm for the Dollo criterion score in a package called `Dollo-CDP`. In an experimental evaluation, we found that `Dollo-CDP` had good performance (finding high scoring trees quickly), although all existing methods performed similarly when data sets had relatively few taxa. We found, by way of a simulation study, that `Dollo-CDP` can provide a benefit for larger numbers of taxa. Most notably, branch-and-bound could not scale to data sets with 50 or more taxa, so `Dollo-CDP` was the only method run that could provide some guarantee of optimality, albeit a more limited one. In practice, we found `Dollo-CDP` often found higher scores trees than `FastH`, even though the trees from `FastH` were used to form constraints for `Dollo-CDP`. `SlowH` sometimes found higher scoring trees than `Dollo-CDP` but was much slower for large numbers of taxa and characters. A caveat of our study is that all methods were run with one thread. All searches in `SlowH` are independent so

Table 1 This table shows a comparison of the tree produced by `Dollo-CDP` and the best trees found by the fast heuristic search (`FastH`) and the slow heuristic search (`SlowH`). We provide the number of replicates for which `Dollo-CDP` is better (b), worse (w), or the same (s) in terms of Dollo criterion score than the alternative method. We also provide the absolute difference in scores for the trees estimated by the two methods averaged across these three cases. Lastly, we provide the runtime (in minutes), averaged across all 25 replicates. The runtime of `Dollo-CDP` includes the time to run `FastH`, the output of which is used to constrain tree space.

	`Dollo-CDP` *vs.* `FastH`		`Dollo-CDP` *vs.* `SlowH`		
# of characters	# of reps b/w/s	Δ score b/w/s	# of reps b/w/s	Δ score b/w/s	Runtime (min) `Dollo-CDP`/`SlowH`
50 taxa					
500	9/0/16	1.56/NA/0	0/0/25	NA/NA/0	0.03/0.14
1000	14/0/11	2.43/NA/0	0/0/25	NA/NA/0	0.04/0.19
5000	13/0/12	6.54/NA/0	0/0/25	NA/NA/0	0.05/0.45
10000	11/0/14	16.09/NA/0	0/1/24	NA/14.00/0	0.06/0.67
50000	12/0/13	53.58/NA/0	0/0/25	NA/NA/0	0.16/1.90
100 taxa					
500	0/0/25	NA/NA/0	1/0/24	1.00/NA/0	0.46/0.46
1000	3/0/22	1.00/NA/0	1/0/24	3.00/NA/0	0.38/1.26
5000	25/0/0	6.40/NA/NA	0/0/25	NA/NA/0	0.16/3.44
10000	23/0/2	12.87/NA/0	0/1/24	NA/7.00/0	0.21/5.56
50000	21/0/4	42.81/NA/0	0/2/23	NA/56.00/0	0.56/15.58
200 taxa					
500	0/0/25	NA/NA/0	5/1/19	1.20/1.00/0	2.19/2.20
1000	0/0/25	NA/NA/0	0/0/25	NA/NA/0	3.04/2.90
5000	19/0/6	1.63/NA/0	0/8/17	NA/3.00/0	1.28/39.29
10000	21/0/4	3.10/NA/0	0/6/19	NA/6.00/0	1.72/39.37
50000	25/0/0	26.68/NA/NA	0/8/17	NA/23.62/0	2.42/93.96

it would be much faster with threading. `Dollo-CDP` could also take advantage of threading, using techniques from Yin *et al.* [41] and could be better optimized. We leave this to future work. Even if the runtime of `SlowH` improves with threading, `Dollo-CDP` could then be run using trees found by `SlowH`, in addition to the ones found by `FastH`, to form constraints. The benefit here is that running `Dollo-CDP` is fast, strictly improves upon prior searches, and gives a guarantee of optimality for the constrained solution space.

An issue that arose when analyzing the *Palaeognathae* biological data set with branch-and-bound is that there can be many optimal trees. `Dollo-CDP` returns a single binary tree, and future work should enable users to get a consensus of the optimal trees in the constrained solution space, similar to `SIESTA` [38]. Lastly, we explored methods for Dollo parsimony in the context of species tree estimation, where data are assumed to follow a population genetics model. Dollo parsimony has also been leveraged in tumor phylogenetics [3, 4, 7, 13]. It would be interesting to explore the utility of `Dollo-CDP` in this application area, especially as the number of leaves (cells instead of species) can be quite large in this setting.

References

1 Md. Shamsuzzoha Bayzid and Tandy Warnow. Gene Tree Parsimony for Incomplete Gene Trees. In Russell Schwartz and Knut Reinert, editors, *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*, volume 88 of *Leibniz International Proceedings in Informatics*

(LIPIcs), pages 2:1–2:13, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.WABI.2017.2.

2 Md. Shamsuzzoha Bayzid and Tandy Warnow. Gene tree parsimony for incomplete gene trees: addressing true biological loss. *Algorithms for Molecular Biology*, 13(1), 2018. doi:10.1186/s13015-017-0120-1.

3 Paola Bonizzoni, Simone Ciccolella, Gianluca Della Vedova, and Mauricio Soto. Beyond perfect phylogeny: Multisample phylogeny reconstruction via ilp. In *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*, ACM-BCB '17, page 1–10, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3107411.3107441.

4 Paola Bonizzoni, Simone Ciccolella, Gianluca Della Vedova, and Mauricio Soto. Does relaxing the infinite sites assumption give better tumor phylogenies? an ilp-based comparative approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(5):1410–1423, 2019. doi:10.1109/TCBB.2018.2865729.

5 Remco Bouckaert, Mareike Fischer, and Kristina Wicke. Combinatorial perspectives on dollo-k characters in phylogenetics. *Advances in Applied Mathematics*, 131:102252, 2021. doi:10.1016/j.aam.2021.102252.

6 David Bryant and Mike Steel. Constructing optimal trees from quartets. *Journal of Algorithms*, 38(1):237–259, 2001. doi:10.1006/jagm.2000.1133.

7 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(Suppl 1):313, 2020. doi:10.1186/s12859-020-03736-7.

8 Alison Cloutier, Timothy B. Sackton, Phil Grayson, Michele Clamp, Allan J. Baker, and Scott V. Edwards. Whole-genome analyses resolve the phylogeny of flightless birds (Palaeognathae) in the presence of an empirical anomaly zone. *Systematic Biology*, 68(6):937–955, 2019. doi:10.1093/sysbio/syz019.

9 William H.E. Day, David S. Johnson, and David Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81(1):33–42, 1986. doi:10.1016/0025-5564(86)90161-6.

10 Payam Dibaeinia, Shayan Tabe-Bordbar, and Tandy Warnow. FASTRAL: improving scalability of phylogenomic analysis. *Bioinformatics*, 37(16):2317–2324, 2021. doi:10.1093/bioinformatics/btab093.

11 Liliya Doronina, Gennady Churakov, Andrej Kuritzin, Jingjing Shi, Robert Baertsch, Hiram Clawson, and Jürgen Schmitz. Speciation network in laurasiatheria: retrophylogenomic signals. *Genome Research*, 27:997–1003, 2017. doi:10.1101/gr.210948.116.

12 Liliya Doronina, Graham M. Hughes, Diana Moreno-Santillan, Colleen Lawless, Tadhg Lonergan, Louise Ryan, David Jebb, Bogdan M. Kirilenko, Jennifer M. Korstian, Liliana M. Dávalos, Sonja C. Vernes, Eugene W. Myers, Emma C. Teeling, Michael Hiller, Lars S. Jermiin, Jürgen Schmitz, Mark S. Springer, and David A. Ray. Contradictory phylogenetic signals in the laurasiatheria anomaly zone. *Genes*, 13(5), 2022. doi:10.3390/genes13050766.

13 Mohammed El-Kebir. SPhyR: tumor phylogeny estimation from single-cell sequencing data under loss and error. *Bioinformatics*, 34(17):i671–i679, 2018. doi:10.1093/bioinformatics/bty589.

14 Joseph Felsenstein. Parsimony in systematics: Biological and statistical issues. *Annual Review of Ecology and Systematics*, 14:313–333, 1983. URL: http://www.jstor.org/stable/2096976.

15 Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2 edition, 2004. doi:10.1007/BF01734359.

16 Joseph Felsenstein. Phylip (phylogeny inference package), 2005. Accessed on XX. URL: https://evolution.genetics.washington.edu/phylip.html.

17 Ronald A. Fisher. On the dominance ratio. *Proceedings of the Royal Society of Edinburgh*, 42:321–341, 1923. doi:10.1017/S0370164600023993.

18 Walter M. Fitch. Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Biology*, 20(4):406–416, 1971. doi:10.1093/sysbio/20.4.406.

19 Ronald L. Graham and Les R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60(2):133–142, 1982. doi:10.1016/0025-5564(82)90125-0.

20 Michael T. Hallett and Jens Lagergren. New algorithms for the duplication-loss model. In *Proceedings of the Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00, page 138–146, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/332306.332359.

21 Richard R. Hudson. Generating samples under a Wright–Fisher neutral model of genetic variation. *Bioinformatics*, 18(2):337–338, 02 2002. doi:10.1093/bioinformatics/18.2.337.

22 Roy N. Platt II, Michael W. Vandewege, and David A. Ray. Mammalian transposable elements and their impacts on genome evolution. *Chromosome Research*, 26:25–43, 2018. doi:10.1007/s10577-017-9570-z.

23 Roy N. Platt II, Yuhua Zhang, David J. Witherspoon, Jinchuan Xing, Alexander Suh, Megan S. Keith, Lynn B. Jorde, Richard D. Stevens, and David A. Ray. Targeted capture of phylogenetically informative ves sine insertions in genus myotis. *Genome Biology and Evolution*, 7(6):1664–1675, 2015. doi:10.1093/gbe/evv099.

24 Mazharul Islam, Kowshika Sarker, Trisha Das, Rezwana Reaz, and Md. Shamsuzzoha Bayzid. STELAR: a statistically consistent coalescent-based species tree estimation method by maximizing triplet consistency. *BMC Genomics*, 21(1):136, 2020. doi:10.1186/s12864-020-6519-y.

25 Jennifer M. Korstian, Nicole S. Paulat, Roy N. Platt II, Richard D. Stevens, and David A. Ray. Sine-based phylogenomics reveal extensive introgression and incomplete lineage sorting in myotis. *Genes*, 13(3):399, 2022. doi:10.3390/genes13030399.

26 Fritjof Lammers, Moritz Blumer, Cornelia Rücklé, and Maria A. Nilsson. Retrophylogenomics in rorquals indicate large ancestral population sizes and a rapid radiation. *Mobile DNA*, 10:5, 2019. doi:10.1186/s13100-018-0143-2.

27 Liang Liu, Lili Yu, and Scott V. Edwards. A maximum pseudo-likelihood approach for estimating species trees under the coalescent model. *BMC Evolutionary Biology*, 10:302, 2010. doi:10.1186/1471-2148-10-302.

28 Diego Mallo, Leonardo De Oliveira Martins, and David Posada. SimPhy : Phylogenomic simulation of gene, locus, and species trees. *Systematic Biology*, 65(2):334–344, 11 2015. doi:10.1093/sysbio/syv082.

29 Siavash Mirarab, Rezwana Reaz, Md. Shamsuzzoha Bayzid, Théo Zimmermann, Michelle S. Swenson, and Tandy Warnow. ASTRAL: genome-scale coalescent-based species tree estimation. *Bioinformatics*, 30(17):i541–i548, 2014. doi:10.1093/bioinformatics/btu462.

30 Siavash Mirarab and Tandy Warnow. ASTRAL-II: coalescent-based species tree estimation with many hundreds of taxa and thousands of genes. *Bioinformatics*, 31(12):i44–i52, 2015. doi:10.1093/bioinformatics/btv234.

31 Erin K. Molloy, John Gatesy, and Mark S Springer. Theoretical and practical considerations when using retroelement insertions to estimate species trees in the anomaly zone. *Systematic Biology*, 71(3):721–740, 2021. doi:10.1093/sysbio/syab086.

32 Erin K. Molloy and Tandy Warnow. FastMulRFS: fast and accurate species tree estimation under generic gene duplication and loss models. *Bioinformatics*, 36(Supplement_1):i57–i65, 07 2020. doi:10.1093/bioinformatics/btaa444.

33 Abdel-Halim Salem, David A. Ray amd Jinchuan Xing, Pauline A. Callinan, Jeremy S. Myers, Dale J. Hedges, Randall K. Garber, David J. Witherspoon, Lynn B. Jorde, and Mark A. Batzer. Alu elements and hominid phylogenetics. *Proceedings of the National Academy of Sciences of the United States of America*, 100(22):12787–12791, 2003. doi:10.1073/pnas.2133766100.

34 Andrew M. Shedlock, Michael C. Milinkovitch, and Norihiro Okada. SINE evolution, missing data, and the origin of whales. *Systematic Biology*, 49:808–817, 2000.

35 Mark S Springer, Erin K Molloy, Daniel B Sloan, Mark P Simmons, and John Gatesy. ILS-aware analysis of low-homoplasy retroelement insertions: Inference of species trees and introgression using quartets. *Journal of Heredity*, 111(2):147–168, 2019. doi:10.1093/jhered/esz076.

36 David L. Swofford. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4.* Sinauer Associates, Sunderland, Massachusetts, 2003.

37 Pranjal Vachaspati and Tandy Warnow. FastRFS: fast and accurate robinson-foulds supertrees using constrained exact optimization. *Bioinformatics*, 33(5):631–639, 09 2016. doi:10.1093/bioinformatics/btw600.

38 Pranjal Vachaspati and Tandy Warnow. SIESTA: enhancing searches for optimal supertrees and species trees. *BMC Genomics*, 19(Suppl 5):252, 2018. doi:10.1186/s12864-018-4621-1.

39 Tandy Warnow. *Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation*. Cambridge University Press, Cambridge, United Kingdom, 2017.

40 Sewall Wright. Evolution in mendelian populations. *Genetics*, 16(2):97–159, 1931. doi:10.1093/genetics/16.2.97.

41 John Yin, Chao Zhang, and Siavash Mirarab. ASTRAL-MP: scaling ASTRAL to very large datasets using randomization and parallelization. *Bioinformatics*, 35(20):3961–3969, 2019. doi:10.1093/bioinformatics/btz211.

42 Yun Yu, Tandy Warnow, and Luay Nakhleh. Algorithms for MDC-based multi-locus phylogeny inference: Beyond rooted binary gene trees on single alleles. *Journal of Computational Biology*, 18(11):1543–1559, 2011. doi:10.1089/cmb.2011.0174.

43 Chao Zhang, Maryam Rabiee, Erfan Sayyari, and Siavash Mirarab. ASTRAL-III: Polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinformatics*, 19(6):153, 2018. doi:10.1186/s12859-018-2129-y.

44 Chao Zhang, Celine Scornavacca, Erin K. Molloy, and Siavash Mirarab. ASTRAL-Pro: Quartet-based species-tree inference despite paralogy. *Molecular Biology and Evolution*, 37(11):3292–3307, 2020. doi:10.1093/molbev/msaa139.

A Algorithms

Algorithm 1 `GetState`

Input: Subtree bipartition $X|Y$, the set S of n species, and an $n \times k$ character matrix $\mathbf{C}$, with $\mathbf{C}[i, j]$ indicating the state assigned to leaf i for character j

Output: State assignment for subtree bipartition $X|Y$ for each of the k characters in $\mathbf{C}$

function GetState($X|Y, S, \mathbf{C}$)
  $states \leftarrow$ an array of length k containing all `0`'s ; $A \leftarrow X \cup Y$; $Z \leftarrow S \setminus A$
  for $i \in \{0, 1, 2, \ldots, k-1\}$ **do**
    if for all $x \in A$, $\mathbf{C}[x, i] =$ `?` **then** $states[i] \leftarrow$ `?`
    else
      $flag \leftarrow 0$
      if $\exists x \in X$ s.t. $\mathbf{C}[x, i] = 1$ **then** $flag \leftarrow flag + 1$
      if $\exists x \in Y$ s.t. $\mathbf{C}[x, i] = 1$ **then** $flag \leftarrow flag + 1$
      if $\exists x \in Z$ s.t. $\mathbf{C}[x, i] = 1$ **then** $flag \leftarrow flag + 1$
      if $flag \geq 2$ **then** $states[i] \leftarrow$ `1`
  return $states$

Algorithm 2 `CountLosses`

Input: Three k-vectors of state assignments: $statesU$, $statesV$, $statesW$

Output: Number of losses assuming that $statesU$ are associated with a vertex u and $statesV$ and $statesW$ are associated with children of u

function CountLosses($statesU, statesV, statesW$)
  $nlosses \leftarrow 0$
  for $i \in \{0, 1, 2, \ldots, k-1\}$ **do**
    if $statesU[i] = 1$ and $statesV[i] = 0$ **then** $nlosses \leftarrow nlosses + 1$
    if $statesU[i] = 1$ and $statesW[i] = 0$ **then** $nlosses \leftarrow nlosses + 1$
  return $nlosses$

Algorithm 3 Construct subtree bipartitions from clades

Input: Set Σ of clades

Output: Subtree bipartitions allowed from Σ stored as a dictionary, where $SubBip[A]$ is a list of the allowed subtree bipartitions for clade A (note that only one side of the subtree bipartition is stored)

function ConstructSubBipsFromClades(Σ)
  Sort Σ by cardinality from least to greatest
  for $i \in \{0, 1, \ldots, |\Sigma| - 1\}$ **do**
    $A \leftarrow \Sigma[i]$; $SubBip[A] \leftarrow [\,]$
    for $j \in \{0, 1, \ldots, i-1\}$ **do**
      $X \leftarrow \Sigma[j]$
      if $X \subset A$ **then** $SubBip[A].append(X)$ (already have ptr to $SubBip[A]$)
  return $SubBip$

Algorithm 4 Dynamic Programming for CC-LDP

Input: Set Σ of clades (sorted by cardinality from least to greatest), a dictionary $SubBip$ of allowed subtree bipartitions previously computed from Σ, an $n \times k$ character matrix $\mathbf{C}$, with each character on species set S

Output: Fills in the dynamic programming matrix $Dollo$ and the traceback matrix $TraceBack$

$Lab \leftarrow dict()$
$SubBipByLab \leftarrow dict()$
for $A \in \Sigma$ **do**
  if $|A| = 1$ **then**
    Do base case for leaves
    $st \leftarrow \mathbf{C}[A, :]$
    $Dollo[A, st] \leftarrow 0$
    $Lab[A] \leftarrow st$
    $SubBipByLab[A][st] \leftarrow \varnothing$
    $Traceback[A][st] \leftarrow NULL$
  else
    Find state assignments for clade A and their associated subtree bipartitions
    $Lab[A] \leftarrow \varnothing$
    $SubBipByLab[A] \leftarrow dict()$
    for $X \in SubBip[A]$ **do**
      $Y \leftarrow A \setminus X$
      $st \leftarrow \texttt{GetState}(X|Y, S, \mathbf{C})$
      Add st to set $Lab[A]$
      Add X to set $SubBipByLab[A][st]$ (initialize if it does not exist)
    For each unique state assignment fill in DP matrix
    $bestDollo \leftarrow \infty$
    $bestX, bestY, bestXState, bestYState \leftarrow NULL$
    for $st \in Lab[A]$ **do**
      for $X \in SubBipByLab[A][st]$ **do**
        $Y \leftarrow A \setminus X$
        for $St_X \in Lab[X]$ **do**
          for $St_Y \in Lab[Y]$ **do**
            $cX \leftarrow Dollo[X, St_X]$
            $cY \leftarrow Dollo[Y, St_Y]$
            $cA \leftarrow \texttt{CountLosses}(st, St_X, St_Y)$
            $score \leftarrow cX + cY + cA$
            if $bestDollo > score$ **then**
              $bestDollo \leftarrow score$
              $bestX \leftarrow X$
              $bestXState \leftarrow St_X$
              $bestY \leftarrow Y$
              $bestYState \leftarrow St_Y$
    $Dollo[A, st] \leftarrow bestDollo$
    $TraceBack(A, st) \leftarrow (bestX, bestXState, bestY, bestYState)$ (two children)

B Software Commands

In all PAUP* commands for heuristic search, we specified seeds so that the results would be reproducible.

B.1 PAUP* command for running the fast heuristic search (FastH)

```
#NEXUS
BEGIN PAUP;
set autoclose=yes warntree=no warnreset=no;
execute <character matrix nexus file>;
outgroup <outgroup name>;
ctype dollo:1-<total number of characters>;
hsearch start=stepwise addSeq=random swap=None nreps=10 rseed=55555;
hsearch start=1 swap=TBR nbest=100 rseed=12345;
rootTrees;
savetrees File=<output file> root=yes trees=all format=newick;
END;
```

B.2 Dollo-CDP command

```
./dollo-cdp \
  -i <character matrix nexus file> \
  -g <outgroup> \
  -t <best 100 trees found from fast heuristic search> \
  -o <output file name>
```

B.3 PAUP* command for running the slow heuristic search (SlowH)

```
#NEXUS
BEGIN PAUP;
set autoclose=yes warntree=no warnreset=no;
execute <character matrix nexus file>;
outgroup <outgroup name>;
ctype dollo:1-<total number of characters>;
hsearch start=stepwise addSeq=random swap=TBR nreps=100 rseed=12345;
rootTrees;
savetrees File=<output file> root=yes trees=all format=newick;
END;
```

B.4 PAUP* command for running branch-and-bound

```
#NEXUS
BEGIN PAUP;
set autoclose=yes warntree=no warnreset=no;
execute <character matrix nexus file>;
outgroup <outgroup name>;
ctype dollo:1-<total number of characters>;
bandb;
rootTrees;
```

```
savetrees File=<output file> root=yes trees=all format=newick;
END;
```

B.5 PAUP* command for computing the Dollo criterion score

```
#NEXUS
BEGIN PAUP;
set autoclose=yes warntree=no warnreset=no;
execute <character matrix nexus file>;
execute <tree nexus file file>;
set criterion=parsimony;
ctype dollo:1-<total number of characters>;
pscores / single=var;
END;
```