Encodings of cladograms and labeled trees

Daniel J. Ford

Google Inc.
1600 Amphitheatre Pkwy,
Mountain View, CA,
USA, 94043

ford@google.com *

Submitted: May 17, 2008; Accepted: Mar 22, 2010; Published: Mar 29, 2010
Mathematics Subject Classification: 05C05, 05C85

Abstract

This paper deals with several bijections between cladograms and perfect matchings. The first of these is due to Diaconis and Holmes. The second is a modification of the Diaconis-Holmes matching which makes deletion of the largest labeled leaf correspond to gluing together the last two points in the perfect matching. The third is an entirely new encoding of cladograms, first as a bijection with a certain set of strings and then via this to perfect matchings. In this pair of bijections, deletion of the largest labeled leaf corresponds to deletion of the corresponding symbols from the string or deletion of the corresponding pair from the matching. These two new bijections are related through a common max-min labeling of internal vertices with two different choices for the label of the root node. All these encodings are extended to cladograms with edge lengths and left-right ordered children. Moving a single symbol in this last encoding corresponds to a subtree prune and regraft operation on the cladogram, making it well suited for use in phylogentics software. Finally, a perfect Gray code for cladograms is derived from the bar encoding, along with a total ordering on all cladograms, Algorithms are also provided for finding the next and previous cladogram, the cladogram at any position, and the position of any cladogram in the sequence.

A cladogram with n leaves is a rooted binary leaf labeled tree with leaves distinctly labeled $1, \ldots, n$. It has long been known that the number of such trees with exactly n leaves is $(2n-3)!!$. This is also the number of perfect matchings on $2(n-1)$ points. Diaconis and Holmes give a bijection in [7] between the set of cladograms and perfect matchings.

*Research supported by Stanford Mathematics Department and NSF grant #0241246

Currently, cladograms are most often encoded in variants of the *Newick* or *New Hampshire* format. This is an enrichment of parenthesis notation which allows additional information such as edge-lengths to be included. However, a major drawback of Newick notation is that there is in general not a unique representation for a cladogram. For example, testing equality of large cladograms given in Newick format is a non-trivial task. For this reason, a bijection is preferable.

One such bijection is that of Diaconis-Holmes. This is used in the R package APE (Analysis of Phylogeny and Evolution [14]) because it provides a unique and compact representation of a cladogram, and in a fast-mixing random walk on cladograms [6]. While simple and elegent, this bijection can be improved upon.

A desirable property which the Diaconis-Holmes bijection lacks is *deletion-stability*. There is a natural projection from the set of cladograms with n leaves to the set with $n-1$ leaves: deletion of the n-th leaf. For the Diaconis-Holmes bijection the induced map on perfect matchings is not natural.

A second direct bijection between cladograms and perfect matchings is presented here, called the *hat encoding*. This is an alteration of the Diaconis-Holmes bijection which makes deletion of the leaf labeled n correspond to gluing together the last two points in the matching. Algorithms are provided for finding the matching corresponding to a cladogram and the cladogram corresponding to a matching.

A completely new encoding of cladograms is also presented, called the *bar encoding*. This coding is a bijection between cladograms with n leaves and a subset of permutations of the set $\{2, \bar{2}, 3, \bar{3}, \ldots, n, \bar{n}\}$. This string of symbols is called the *name* of a cladogram. Deletion of the leaf labeled n corresponds to deletion of the symbols n and $\bar{n}$ from the name. The set of names is in natural bijection with the set of matchings on $2n-2$ points. For a cladogram with n leaves, deletion of the leaf labeled n corresponds to removing the last pair in the matching (pairs are labeled by starting at the last point in the set and moving to the first, labeling pairs n to 2 in the order they are first encountered).

The hat and bar encodings both involve labeling the internal vertices of a tree. Both of these labelings may be easily described in terms of maxmin labeling, covered in Section 4. Which of the labeling is generated depends on the choice of label for the root vertex.

The bar encoding is also used to give a perfect Gray code on the set of cladograms with n leaves. In this case, the Gray code is a sequential ordering of the set of cladograms so that adjacent cladograms differ by a small amount, specifically a subtree prune and regraft operation. Algorithms are provided to find the name of the next and previous cladogram in the Gray code. Algorithms are also provided which return the position of a cladogram in the Gray code given its name, and the name of the cladogram in a given position. Such functions are sometimes called *ranking* and *unranking* functions, such as those for the set of permutations given by Myrvold and Ruskey [13]. The Combinatorial Object Server [16] uses such functions to provide indexed lists for many types of objects but does not yet serve cladograms.

The necessary basic definitions are now reviewed.

Recall that a *tree* is a simple graph of vertices and edges with precisely one non-self-intersecting path between any two vertices.

A *cladogram* with n leaves is a finite rooted binary tree with non-root leaves distinctly labeled $1, 2, \ldots, n$. Note that the planar representation of the cladogram is not important: ie. 'left' and 'right' children are not distinguished. A *fat cladogram*, or *oriented cladogram*, is a cladogram where the children of each vertex are distinguished as the 'left' child and the 'right' child. In other words, the edges around each vertex have a cyclic ordering.

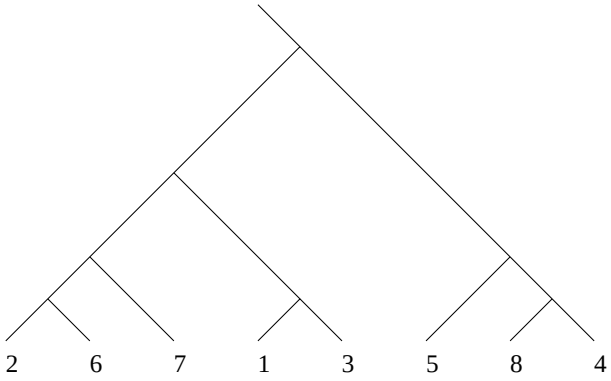

Figure 1: A cladogram with 8 leaves.

A perfect matching of $2m$ points may be thought of as an involution on a set of $2m$ points which has no fixed points. In other words, every point is paired with another point, and each point is a member of exactly one pair. The two points in a pair may be thought of as being joined by an edge. Figure 2 shows an example of a perfect matching.

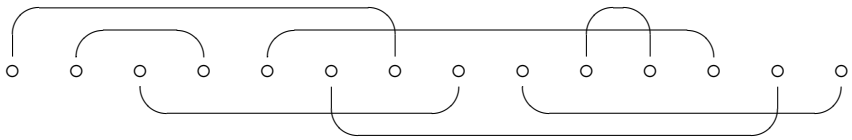

Figure 2: A perfect matching on 14 points.

There are several different possible definitions for what it means for two cladograms to be 'close' to one another. Waterman [22] defined two cladograms to be *adjacent* if one may be obtained from the other by migrating a sub-branch past a single vertex. This is often called *nearest neighbor interchange*. This was extended to the continuous case by Billera, Holmes and Vogtmann [3].

Two cladograms might also be considered *adjacent* if one may be obtained from the other by migrating a single branch from one location to another. In other words, two cladograms are adjacent if the subtree below an edge in the first cladogram can be pruned and then regrafted onto another edge of the remaining cladogram to arrive at the second cladogram. This is often called *rooted subtree prune and regraft* (rSPR). A special case of this is nearest neighbor interchange, where an edge is migrated past a neighboring edge. See [9] for a good introduction. Bonet, St John, Mahindru and Amenta give a algorithm for approximating the distance between trees under this metric [4].

1 The Diaconis-Holmes bijection

The only previously reported encoding of cladograms as perfect matchings is that of Diaconis and Holmes [7]. This encoding is now briefly described.

Let the term *sibling pair* denote a pair of vertices with the same parent vertex. Let the term *non-root branch point* denote a branch point which is not the first branch point below the root. The Diaconis-Holmes (DH) bijection may be described as a two-step process: first label internal vertices, then record sibling pairs.

Algorithm: DiaconisHolmesBijection
Input: A cladogram t with $n \geqslant 2$ leaves.
Output: A perfect matching on the set $\{1, 2, \ldots, n, n+1, \ldots, 2n-2\}$.

1: (Start by labeling the internal vertices as follows:)
2: **while** there are unlabeled non-root branch points **do**
3: Consider every sibling pair which has both siblings labeled, but not the common parent. Of these, choose the sibling pair which contains the smallest label.
4: Give the parent of this sibling pair the smallest unassigned label.
5: **end while**
6: Return the set of all sibling pairs. (This is the perfect matching corresponding to the cladogram).

For example, Figure 3 shows a cladogram before and after its internal vertices are labeled. The matching for this tree is given by taking all sibling pairs:
$(1,5)(3,4)(6,7)(2,8)(9,10)$.

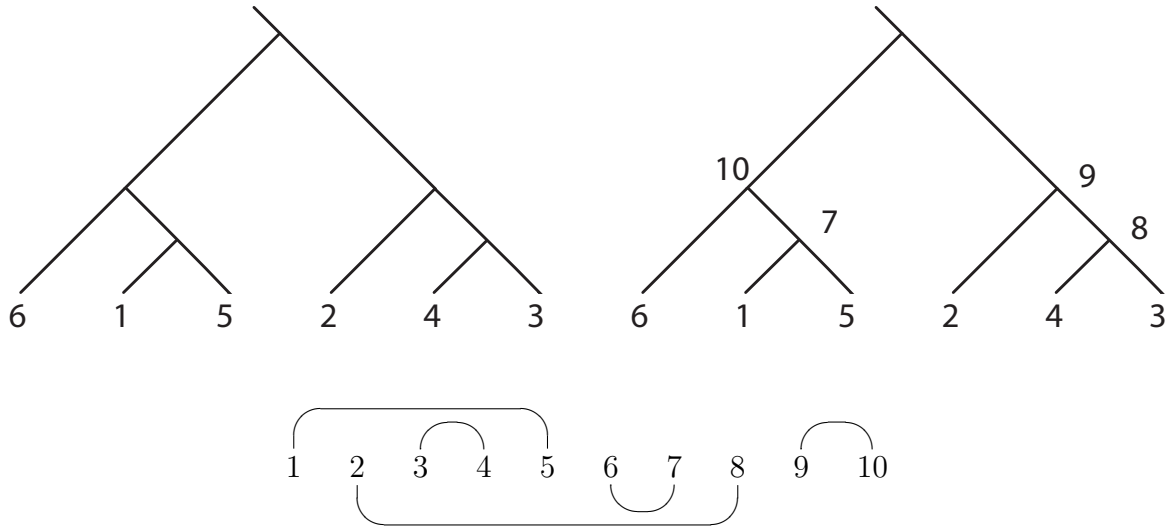

Figure 3: A cladogram with 6 leaves before and after labeling by the DH scheme, and its DH matching: $(1,5)(3,4)(6,7)(2,8)(9,10)$

The inverse algorithm from [7], which takes a perfect matching and gives a cladogram, follows the obvious procedure: connecting sibling pairs together at their parent node and doing this in the order corresponding the the labeling procedure in the previous algorithm.

Algorithm: InverseDiaconisHolmesBijection
Input: A perfect matching on the set $\{1, 2, \ldots, n, n+1, \ldots, 2n-2\}$, with $n \geqslant 2$.
Output: A cladogram t with n leaves.

1: Create a graph, G with n nodes labeled $1, \ldots, n$.
2: Create a set, S, of all the pairs in the perfect matching.
3: **for** i from 1 to $n-1$ **do**
4: Take all the pairs in S for which both their elements have corresponding labeled points in G.
5: Choose the pair, (a, b), with the smallest element from among these.
6: Create a new node in the graph labeled $n+i$.
7: Create edges from node a to node $n+i$ and from node b to node $n+i$.
8: Remove the pair (a, b) from the set S.
9: **end for**
10: Declare the node labeled $2n-1$ to be the root of the graph.
11: Remove the node labels $n+1, \ldots, 2n-1$ and return the resulting rooted graph.

For completeness, a proof that these functions form a bijection presented her. First, show that the above algorithm gives a cladogram with the desired property.

Proposition 1 *The above algorithm produces a (rooted) cladogram with n leaves and the tree with internal labels has sibling pairs equal to the pairs in the matching.*

Proof. First, show that the algorithm never gets stuck at Step 5: there is always at least one pair in S to choose in Step 5. This follows by a simple counting argument. There are $n+i-1$ points in the graph with labels from the set $\{1, \ldots, 2n-2\}$ and $n-1$ pairs in the matching on the same set so at least i pairs have both their elements in the graph. The set S contains $n-i$ of the $n-1$ pairs so it must contain at least one of the i pairs for which both elements are already labels in graph G.

Next, note that the graph has exactly $2n-1$ nodes labeled $1, \ldots, 2n-1$. Also, note that all edges are created in Step 7. Thus, nodes $1, \ldots, n$ have degree 1 since these labels occur in the matching exactly once and are not of the form $n+i$ for $i \geqslant 1$. Similarly, nodes $n+1, \ldots, 2n-2$ have degree 3 since each of these labels occur once in the matching, contributing one edge to their parent, and once in the form $n+i$ for some $i \geqslant 1$ which contributes 2 edges from their children. Finally, the root node, labeled $2n-1$ has one edge from each of its two children and does not occur in the matching.

Now, with the exception of node $2n-1$, each node is connected to a unique node with a larger label. This follows as edges are only created in Step 7 both a and b must be less than $n+i$ as they already exist in the graph G and, since the input is a perfect matching, each node occurs in Step 7 as a or b exactly once.

Thus, the resulting graph is a rooted tree and the parent of each node other than $2n-1$ is the unique adjacent node with a higher label. This implies that the nodes a and b in step Step 7 are sibling (share the same parent). These are exactly the pairs in the matching. □

Proposition 2 *For any integer $n \geqslant 2$, the function DiaconisHolmesBijection defined above gives a bijection between cladograms with n leaves and perfect matchings on the set of points $\{1, \ldots, 2n-2\}$.*

Proof. Take a perfect matching and use the algorithm InverseDiaconisHolmesBijection to generate a cladogram. Apply the algorithm DiaconisHolmesBijection, which labels the internal nodes of this cladogram and records the sibling pairs, to give a second matching. The aim is to show that these two matchings are identical and from there that the functions are inverse to each other.

By Proposition 1, the cladogram in Step 10 of InverseDiaconisHolmesBijection, with internal leaves labeled, has sibling pairs given by the original matching m. All that remains is showing that the labeling of the internal nodes by DiaconisHolmesBijection. This is clear, since the labeling of nodes in one happens in exactly the same way as the creation of nodes in the other: in one case the sibling pair for which the labels exist in the graph which has the smallest label, and in the other case the matching pair (soon to be sibling pair) for which both labels exist in the graph which has the smallest label.

Since the labeling of the internal nodes agrees, the set of sibling pairs agrees and so the two matchings are equal. It is well known that the set of perfect matchings on $\{1, \ldots, 2n-2\}$ and the set of cladograms with n leaves have the same cardinality ([19] and later [5]), completing the proof that these functions are inverses of each other and so are bijections. □

1.1 Encoding edge lengths and fat cladograms

Diaconis and Holmes [7] also note that if the cladogram comes equipped with edge lengths then these may also be encoded by labeling each point in the matching with the length of the edge above the corresponding vertex of the tree. These lengths may be recorded as a subscript to the label.

For example, if all (non-root) edges in the cladogram in Figure 3 have length proportional to their apparent length then the corresponding labeled matching is:

$$(1_1, 5_1)(3_1, 4_1)(6_2, 7_1)(2_2, 8_1)(9_3, 10_3)$$

The length of the root edge is not recorded. This is not a serious limitation in common use cases such as phylogenetics, where it does not make sense to consider the length of the root edge.

This encoding is used in the R package *ape* (Analysis of Phylogenetics and Evolution) [14].

Note that similar additional information may be used to extend the DH encoding to fat cladograms. A *fat cladogram*, or *oriented cladogram* is a cladogram together with a cyclic ordering of the edges at every vertex. In other words, the 'left' and 'right' child of a vertex are distinguished from each other. The term *fat* comes from the concept of a *fat graph*, where an ordering is placed on the edges incident to each vertex. Fat graphs were first introduced by Penner in [15].

This additional information may be easily added to the matching by ordering each pair: placing the 'left' child first and the 'right' child second. This may also be thought of as orienting an edge joining the two elements of a pair, or labeling this edge with ± 1. Call such a perfect matching with this extra information a *directed perfect matching*, or *edge labeled perfect matching*.

For example, considering the cladogram in Figure 3 as a fat cladogram makes the corresponding directed/edge-labeled perfect matching:
$(1,5)(4,3)(6,7)(2,8)(10,9)$

The next section introduces a further alteration to the DH bijection with improved properties. Specifically, given the deletion map on cladograms which removes the largest leaf, the corresponding map on perfect matchings induced by the bijection is very natural: gluing together the last two points of the matching.

2 The hat bijection between cladograms and perfect matchings

This section describes a new bijection between cladograms with n leaves and perfect matchings on $2n-2$ points $\{1,2,3,\hat{3},\ldots,n,\hat{n}\}$. This bijection is an alteration of the bijection of Diaconis and Holmes described in the previous section. The difference is in the way that the internal vertices are labeled before recording sibling pairs. This bijection will be called the *hat bijection*, for lack of a better name.

Some notation is now introduced to aid description of the bijection.

For a rooted tree t let the *subtree of t spanned by leaves $v_1,\ldots,v_k$* denote the usual subgraph spanned by these vertices and the root vertex, except that vertices of degree 2 are erased (so that their two adjacent vertices are now joined directly by an edge). See Figure 4 for an example.

There is a natural injection from the set of vertices of the subtree into the original tree, and from the set of edges of the subtree into edges of the original tree. The bijection is clear for the leaves themselves. An internal vertex v in the subtree is identified by the set of leaves below it. The corresponding vertex in the supertree is the lowest common ancestor of this set of leaves. In other words, the corresponding vertex is on the shortest paths from each of these leaves to the root, and contains all such vertices on its own shortest path to the root. In this way the vertices of the subtree may be considered as vertices of the supertree.

The edges of the subtree may also be considered as edges of the supertree. Specifically, if two vertices correspond to each other then the single edges immediately above them

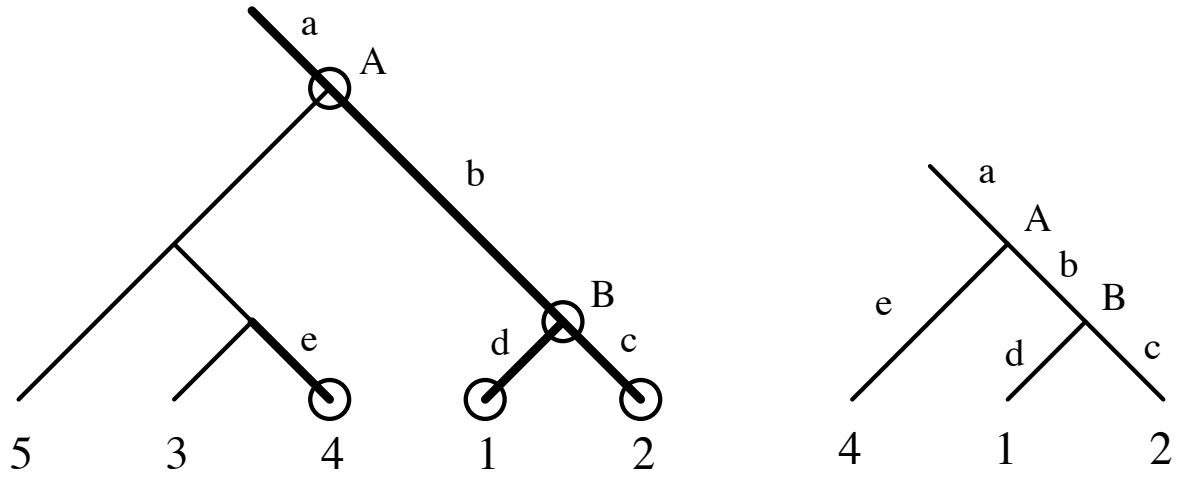

Figure 4: The tree on the right is the subtree of the one on the left spanned by leaves 1, 2 and 4. The vertices and edges in the supertree corresponding to those in the subtree are highlighted and labeled.

correspond also. An example of this is shown in Figure 4.

Let $Cl(n)$ denote the set of cladograms with n leaves

Definition 3 *Let $D_n : Cl_n \to Cl_{n-1}$ denote the operation of deleting the largest leaf of a cladogram with n leaves. Specifically, $D_n(t)$ is the cladogram given by removing from cladogram t vertex n and its parent (and the three edges incident to these two vertices) and creating a new edge between the two neighbors of the parent of n (that vertex's parent and its other child, which is a sibling of n).*

Extend this definition to cladograms with edge lengths by giving the new edge length equal to the sum of the two edges which were just removed from its two end points, thus preserving the natural distance between all surviving nodes.

Extend this definition to oriented/fat cladograms by replacing, in the cyclic ordering at each of the two surviving modified nodes, the just removed edges with the newly created edge.

In the case of a cladogram with n leaves, the subtree spanned by leaves $1, 2, \ldots, k$ is given by deleting leaves $n, n-1, \ldots, k+1$ with the deletion maps $D_n, D_{n-1}, \ldots, D_{k+1}$.

Conversely, a new leaf labeled n may be inserted into a cladogram with $n-1$ leaves at an edge e. This is done by creating two new vertices, call them n and $\bar{n}$ which are joined by an edge. A new edge is added from $\bar{n}$ to each of the two ends of edge e and then edge e itself is removed (so that the resulting graph is still a tree). See Figure 5 for an example of insertion and deletion.

Let the term *first branch point* refer to the first internal vertex below the root (for a tree with at least 2 leaves). Let the term *non-root branch point* refer to any internal vertex (branch point) which is not the first branch point.

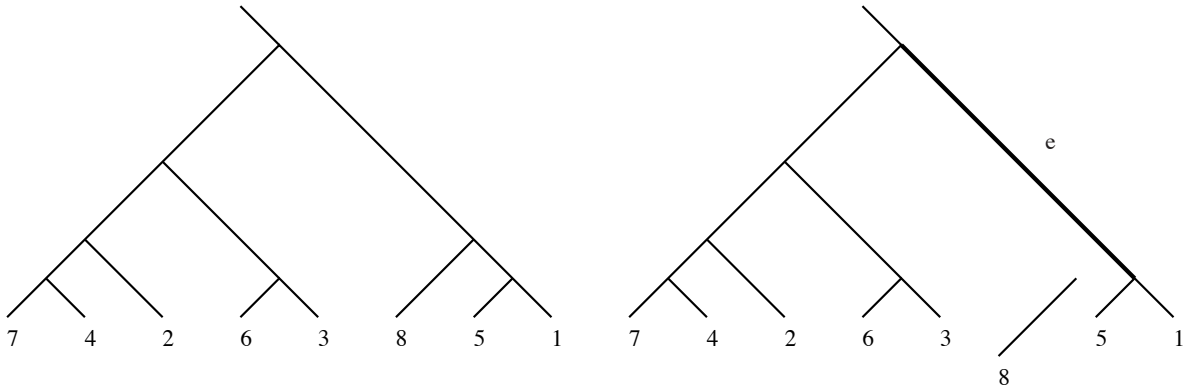

Figure 5: The tree on the right is the subtree of the one on the left gained by deleting leaf 8. Alternatively, the supertree on the left is gained by inserting leaf 8 into the highlighted edge, e, of the tree on the right.

Below is an algorithm, called *HatBijection*, for producing the perfect matching for a cladograms with at least 2 leaves.

Algorithm: HatBijection
Input: A cladogram t with $n \geqslant 2$ leaves.
Output: A perfect matching on the set $\{1, 2, \ldots, n, \hat{3}, \ldots, \hat{n}\}$.

1: Let t_k, for $i \in \{2, \ldots, n\}$, denote the subtree of t spanned by leaves $1, \ldots, k$.
2: **for** $i = 3, \ldots, n$ **do**
3: t_i has exactly one non-root branch point which is not a non-root branch point of t_{i-1}. Label this vertex $\hat{i}$.
4: **end for**
5: Return all sibling pairs.

Corollary 6 shows that this function defines a bijection between cladograms with n leaves and perfect matchings on the set $\{1, 2, \ldots, n, \hat{3}, \ldots, \hat{n}\}$. The inverse function is given in Section 2.2. Also, note that $t_{k-1} = D_k t_k$, the cladogram obtained by deleting leaf k from t_k.

For example, Figure 6 shows a cladogram labeled according to this algorithm and the corresponding perfect matching. Figure 7 shows the cladogram obtained by deleting the largest leaf, 8, and its corresponding perfect matching.

Notice that the perfect matching for this second cladogram is obtained from the first by gluing together nodes 8 and $\hat{8}$, which converts the two pairs $(4, 8)$ and $(5, \hat{8})$ into a single pair $(4, 5)$. This correspondence between deletion and gluing occurs in general.

Let h denote the map from cladograms to perfect matchings defined by algorithm HatBijection.

Recall that $Cl(n)$ denotes the set of cladograms with n leaves and $D_n : Cl_n \to Cl_{n-1}$

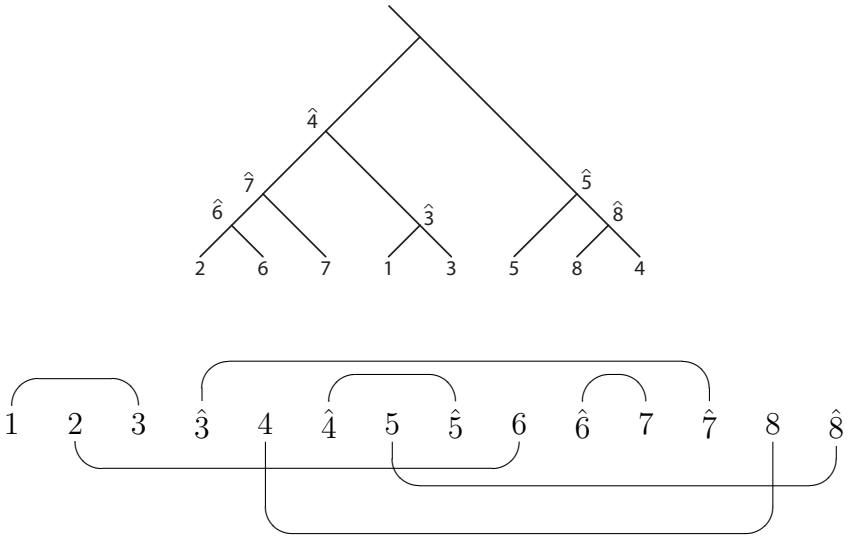

Figure 6: A cladogram with 8 leaves with internal vertices labeled according to the algorithm called hatBijection, and its corresponding perfect matching: $(1,3)(2,6)(\hat{3},\hat{7})(4,8)(\hat{4},\hat{5})(5,\hat{8})(\hat{6},7)$

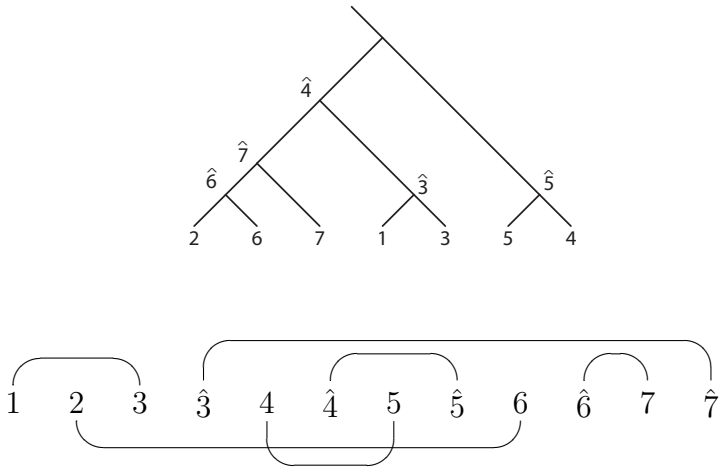

Figure 7: A cladogram with 7 leaves with internal vertices labeled according to the algorithm called hatBijection, and its corresponding perfect matching: $(1,3)(2,6)(\hat{3},\hat{7})(4,5)(\hat{4},\hat{5})(\hat{6},7)$

the operation of deleting the largest leaf. For $n \geqslant 2$, let $\mathrm{Match}(n)$ denote the set of matchings on the set $\{1, 2, \ldots, n, \hat{3}, \ldots, \hat{n}\}$ (for $n = 2$ the set is $\{1, 2\}$). For $n \geqslant 3$, let $G_n : \mathrm{Match}(n) \to \mathrm{Match}(n-1)$ denote the operation which glues points n and $\hat{n}$ together: If n and $\hat{n}$ were paired by the matching then simply remove them and the edge between them, otherwise remove them both and glue the point which was paired with n to the point which was paired with $\hat{n}$.

Proposition 4 *If t is a cladogram with $n \geqslant 3$ leaves and $h(t)$ is the corresponding perfect matching then deleting leaf n corresponds to gluing n and $\hat{n}$ together: $G_n(h(t)) = h(D_n(t))$ In other words, the following diagram commutes:*

$$\begin{array}{ccc} Cl(n) & \xrightarrow{h} & Match(n) \\ {\scriptstyle D_n}\downarrow & & \downarrow{\scriptstyle G_n} \\ Cl(n-1) & \xrightarrow{h} & Match(n-1) \end{array}$$

Proof. Consider a cladogram t with n leaves.

First, note that $t_{n-1} = D_n t$, the tree given by deleting leaf n from tree t. Now, tree t contains exactly one internal non-root branch point which is not a non-root branch point of $t_{n-1} = D_n t$.

If the parent of leaf n is the root branch point then the sibling of n is the new non-root branch point and is thus labeled $\bar{n}$. Therefore, every sibling pair in $D_n t$ is still a sibling pair in t. Thus the matching corresponding to t is precisely the matching corresponding to $D_n t$ on points $1, 2, 3, \ldots, n-1, \hat{3}, \ldots, n \hat{-} 1$ along with the new sibling pair $(n, \hat{n})$. Gluing this last pair together recovers the matching for $D_n t$.

If the parent of n in t is not the root branch point then this parent is the new non-root branch point and is therefore labeled $\hat{n}$. Let x be sibling of n and y be the sibling of $\hat{n}$ (see Figure 8). Deleting vertices n and $\hat{n}$ from tree t and joining x with an edge to the parent of $\hat{n}$ produces the tree $D_n t$. Note that in this tree the vertices x and y are now siblings. All other sibling pairs remain unaltered. Therefore, the matching for $D_n t$ is gained from the matching for t by taking the points x and y, which are paired with n and $\hat{n}$ respectively, and pairing them whilst removing points n and $\bar{n}$. This is precisely the operation of gluing n and $\hat{n}$ together. □

The reason this bijection is an improvement on the previous bijection is that it preserves some of the natural structure on the objects in question by carrying a natural operation on one set to a natural operation on the other. In this case, the new bijection allows the operation of deleting the largest leaf of a cladogram to be performed directly on the matching representation. Furthermore, moving a symbol in the bar encoding of a cladogram corresponds to a subtree prune and regraft (SPR) operation on trees. This SPR operation preserves most of the structure of the tree, allowing reuse of partial results in likelihood calculations, and is biologically natural because it describes reticulation in evolution: [12], [20].

The DH bijection is used in the R package APE (Analysis of Phylogeny and Evolution [14]) because it provides a unique and compact representation of a cladogram.

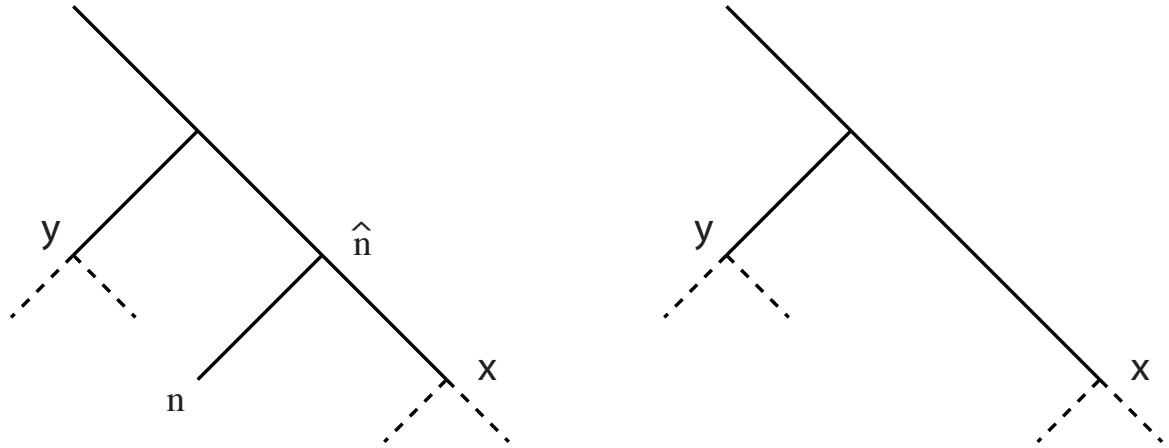

Figure 8: In the cladogram on the left, (n, x) and $(\hat{n}, y)$ are sibling pairs. In the cladogram on the right (x, y) is a sibling pair.

The advantages of the bar encoding make it well suited for this and other phylogentics software.

The next two sections briefly discuss encoding fat cladograms and cladograms with edge lengths and give the inverse map for this bijection.

The section following these, Section 3, describes an encoding of cladograms as certain types of strings and an associated bijection between cladograms and perfect matchings. The new encodings in this latter section preserve deletion of the largest leaf in a different way than the hat bijection just discussed.

2.1 Encoding edge lengths and fat cladograms

Again, given a cladogram with edge lengths, the non-root edge lengths may be recorded by labeling each point in the matching by the length of the edge above the corresponding vertex in the cladogram. If the cladogram is a fat cladogram then each pair may be ordered: 'left' child then 'right' child.

For example, considering the cladogram in Figure 6 as a fat cladogram with all edge lengths equal to 1 gives the corresponding directed, labeled perfect matching:

$$(1_1, 3_1)(2_1, 6_1)(\hat{7}_1, \hat{3}_1)(8_1, 4_1)(\hat{4}_1, \hat{5}_1)(5_1, \hat{8}_1)(\hat{6}_1, 7_1)$$

Considering all edge lengths to be proportional to their apparent length in the diagram in Figure 6 gives:

$$(1_1, 3_1)(2_1, 6_1)(\hat{7}_2, \hat{3}_3)(8_1, 4_1)(\hat{4}_3, \hat{5}_5)(5_2, \hat{8}_1)(\hat{6}_1, 7_2)$$

When deleting the largest leaf of an n-leaf cladogram, the lengths of some edges may change. The lengths associated with most edges in the tree and labels in the encoding remain the same. There are two cases to consider:

If n and $\hat{n}$ are paired then they must be the children of the first branch point and so removing them only effects the length of the root edge, which is not recorded. Thus, all remaining recorded edge lengths are remain unchanged. If n and $\hat{n}$ are not paired then removing $\hat{n}$ increases the length of the edge below it by the length of the edge above it. This corresponds in the encoding to adding the length associated with $\hat{n}$ to the length of the vertex paired with n, and leaving all other recorded lengths unchanged.

In the example above, when leaf 8 is deleted the length associated with $\overline{8}_1$ is added to the length associated with 4_1 (which was paired with 8) to give 4_2:

$$(1_1, 3_1)(2_1, 6_1)(\hat{7}_2, \hat{3}_3)(\hat{4}_3, \hat{5}_5)(5_2, 4_2)(\hat{6}_1, 7_2)$$

2.2 Recovering the cladogram given the matching

This section contains an inverse function for the algorithm HatBijection as well as a proof that they are actually inverse to each other. This implies that HatBijection is actually a bijection.

Recall the definition of inserting a leaf, given at the beginning of Section 2.

The following is the natural recursive function which is inverse to function HatBijection.

Algorithm: HBInverse
Input: A perfect matching m on the set $\{1, 2, \ldots, n, \hat{3}, \ldots, \hat{n}\}$ $(n \geqslant 2)$.
Output: A cladogram t with n leaves.

1: **if** n=2 **then**
2: Return the unique cladogram with two leaves.
3: **end if**
4: Let m' be the perfect matching given by gluing n and $\hat{n}$ together in matching m (ie. $m' := G_n(m)$).
5: Let $t' :=$ HBInverse(m').
6: **if** n and $\hat{n}$ are paired in m **then**
7: Let t be the cladogram which is the root join of the leaf n and cladogram t' (ie. insert leaf n into the root edge of t').
8: Label the sibling of n in t with symbol $\hat{n}$.
9: **else**
10: Let x be the point paired with n and y be the point paired with $\hat{n}$ in matching m.
11: Let t be the tree gained from t' by inserting a leaf labeled n into the edge immediately above the vertex labeled x.
12: Label the newly created internal vertex $\hat{n}$.
13: **end if**
14: Return labeled cladogram t.

Proposition 5 *The algorithms HatBijection and HBInverse are inverse to each other.*

Proof. Proceed by induction on the number of leaves n. Both algorithms are trivial for $n = 2$ and are inverse to each other (there is only one cladogram and only one matching).

Suppose that the algorithms are inverse to each other for all $n < k$. Let h denote the function defined by algorithm HatBijection and g the function defined by algorithm HBInverse.

Let t be a cladogram with k leaves. Show that $gh(t) = t$ as follows:

Let $t' = D_k t$, the cladogram gained by deleting leaf k from cladogram t. Let m' be the perfect matching which is in bijection with t' (ie $m' = h(t')$).

Now, if k is a child of the root branch point then the sibling of k is labeled $\hat{k}$ and so the perfect matching m given by algorithm HatBijection has k matched with $\hat{k}$. Applying algorithm HBInverse to the matching m first builds the tree for the matching restricted to $1, 2, 3, \ldots, k-1, \hat{3}, \ldots, (\hat{k-1})$ (line 5) as k and $\hat{k}$ are matched in m. By Proposition 4 this tree is $D_k t$. Finally, the cladogram obtained by inserting leaf k into $D_k t$ at the root (lines 6-8) is precisely cladogram t.

On the other hand, if k is not a child of the root branch point then the parent of k in t is labeled $\hat{k}$. Let $m = h(t)$ be the perfect matching given by applying algorithm HatBijection to t. Let x be the sibling of k and y the sibling of $\hat{k}$. Now, the tree constructed from m by algorithm HBInverse is $D_k t$ (line 5) with leaf k inserted into the edge immediately above vertex x (lines 10-12). This makes k the sibling of x in the new tree. In other words, k is reinserted into $D_k t$ at the unique edge which makes it a sibling of x. Therefore, this is precisely cladogram t. This completes the proof that $gh(t) = t$.

It is well known that the set of perfect matchings on $\{1, \ldots, 2n-2\}$ and the set of cladograms with n leaves have the same cardinality [19], proving that g is inverse to h. A direct proof that $hg(m) = m$ for any matching m is also possible, but is omitted here.

This completes the inductive step for the converse direction (that HBInverse followed by HatBijection is the identity). □

This leads immediately to the following corollary:

Corollary 6 *The function HatBijection provides a bijection between cladograms with n leaves and perfect matchings on the set of points $\{1, 2, \ldots, n, \hat{3}, \ldots, \hat{n}\}$.*

Proof. This follows directly from the previous Proposition. □

3 The bar encoding of cladograms as strings or perfect matchings

This section presents a completely new encoding of cladograms, first as strings and then as matchings. The *bar coding* is a deletion stable coding for cladograms with n leaves as certain strings of length $2n$ on the alphabet $\{1, \bar{1}, 2, \bar{2}, \ldots, n, \bar{n}\}$.

As with the previous bijections, the internal vertices are first labeled. Two algorithms are presented for this labeling. The previous two encodings would return the set of

sibling pairs at this point. However, for this encoding the completely labeled tree gives a string encoding, which then leads to a perfect matching. This string encoding, called the *name* of the cladogram is discussed in Section 3.2, while the bijection with perfect matchings is covered in Section 3.3. These are both extended to fat/oriented cladograms and cladograms with edge lengths in Section 3.4.

The labeling and string encoding are now presented. Consider a cladogram with leaves labeled $1, 2, \ldots, n$, such as that in figure 1. The following algorithm labels the internal vertices.

An algorithm for labeling the internal vertices of a cladogram.
Algorithm: barLabeling (a)
Input: A cladogram t with n leaves.
Output: A cladogram t with n leaves and all internal leaves labeled.

1: **for** i=n,...,2 **do**
2: Follow the path from leaf i towards the root and label the first encountered unlabeled vertex with symbol $\bar{i}$.
3: **end for**
4: (Notice that the root is always labeled $\bar{1}$. This label is sometimes omitted from diagrams.)

Later, Proposition 8 shows this gives an identical labeling to a recursive algorithm, called barLabeling (b).

This labeling of the internal vertices leads to the string encoding of the cladogram via the following algorithm.
Algorithm: nameOfCladogram
Input: A cladogram t with n leaves.
Output: The name of cladogram t.

1: Label internal vertices of t by the algorithm barLabeling (a).
2: Start with an empty string s.
3: Append symbol $\bar{1}$ to string s.
4: **for** i=1,...,n **do**
5: Append symbol i to string s.
6: Follow the path from leaf i towards the root and append each symbol encountered to string s until symbol $\bar{i}$ is encountered. (Do not append $\bar{i}$).
7: **end for**
8: Return string s.

Figure 9 shows a cladogram with 8 leaves labeled according to the above scheme and the resulting name.

Notice that the string always begins with $\bar{1}1$. This initial segment is sometimes omitted from the name of the cladogram. Sometimes the label of the smallest leaf is kept in brackets at the beginning of the string, as in Figure 9. This is useful when joining two trees at the root (see Section 3.5), as all of these algorithms extend to trees with leaves distinctly labeled by integers, such as those in Figure 19.

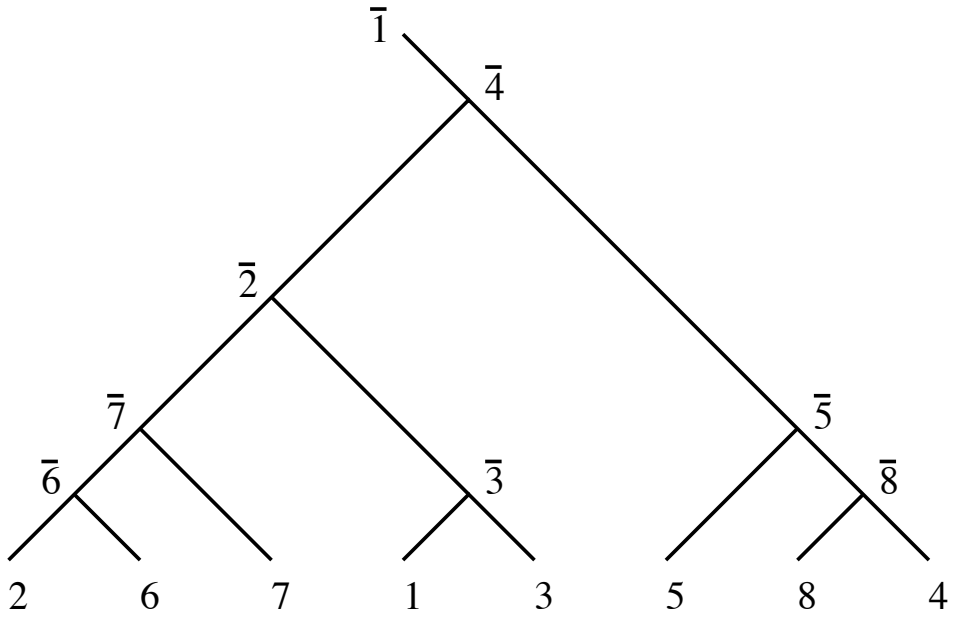

Figure 9: The cladogram with name $(1)\bar{3}\bar{2}4\bar{2}6\bar{7}34\bar{8}\bar{5}5678$

This function, *nameOfCladogram*, is a bijection between cladograms and a certain set of strings, called *names of cladograms* (Corollary 14).

Definition 7 *Define the set of* names of cladograms *with $n \geqslant 2$ leaves, denoted Name(n), to be the set of strings satisfying the following three conditions:*
1 - Each of the symbols $2, 3, \ldots, n, \bar{2}, \ldots, \bar{n}$ occurs exactly once in the string and no other symbols occur.
2 - If $k < l$ then symbol k occurs to the left of symbol l in the string
3 - Symbol $\bar{k}$ occurs to the left of the symbol k.

The name of a cladogram is also deletion stable in the sense that removing leaf n corresponds to deleting symbols n and $\bar{n}$ from the name. The inverse function which creates a cladogram given its name is also provided in Section 3.2.

First, however, the labeling produced by this algorithm is examined.

3.1 The bar labeling

The internal vertex labeling given by barLabeling algorithm (b), below, and a recursive definition for the name of a cladogram was first discovered by demanding that it and the name it defines satisfy the deletion property. The easier to use barLabeling algorithm (a) and the piecewise sequential reading of the name was derived later.
Algorithm: barLabeling (b)
Input: A cladogram t with n leaves.

Output: A cladogram t with n leaves and all internal leaves labeled.

1: If t has one leaf then label the root vertex $\bar{1}$ and return the tree.
2: Otherwise, the cladogram $D_n t$ contains one internal node which does not correspond to an internal vertex of t. This is the immediate parent of leaf n.
3: Label $D_n t$ according to barLabeling and transfer these labels to the corresponding internal vertices of t.
4: Label the remaining internal vertex $\bar{n}$.

This algorithm for labeling the internal vertices of a tree barLabeling, (a) and (b), produce identical labelings.

Proposition 8 *The internal labeling given by the two barLabeling algorithms, (a) and (b), are identical for every cladogram.*

Proof. Proceed by induction. The statement is trivially true for the cladogram with one leaf, as there are no internal vertices to label. Let $n \geqslant 2$ and suppose the proposition is true for all cladograms with less than n leaves. Let t be any cladogram with n leaves. After the first visit to line 2 in algorithm (a) the only internal vertex labeled is the parent of leaf n. Thereafter, this internal vertex is already labeled and so always skipped over in line 2. Therefore algorithm (a) proceeds along the other internal vertices of t in the same order that it would along the corresponding vertices of $D_n t$, the tree t with leaf n deleted. By induction, this labeling of the other internal vertices is identical to the labeling of $D_n t$ produced by algorithm (b). Finally, algorithm (b) also labels the parent on leaf n with label $\bar{n}$ so the two labeling agree everywhere. The proposition now follows by induction on n. □

Let l_n denote the function which takes a cladogram with n leaves and labels it internal leaves with algorithm barLabeling. Let D_n denote the operation of deleting the n-th leaf from a cladogram with n leaves.

Corollary 9 *The bar labeling is deletion stable: For a cladogram t with n leaves, $D_n l_n t = l_{n-1} D_n t$.*

Proof. This follows directly from the recursive definition of barLabeling (algorithm (b), Step 3). □

This labeling, and the hat labeling, are generalized in Section 4 as *maxmin* labelings.

Finally, for the algorithm nameOfCladogram to make sense, the following proposition must be true:

Proposition 10 *For any bar labeled cladogram with n leaves, and for all $k = 1, \ldots n$, the vertex labeled $\bar{k}$ lies above leaf k (on the shortest path from k to the root).*

Proof. The statement is true for the unique cladogram with 1 leaf. Suppose that statement is true for all cladograms with $n-1$ leaves. Inserting leaf n anywhere in the

tree does not change the fact that $\bar{k}$ is above k. Finally the vertex $\bar{n}$ lies immediately above n. The proposition now follows by induction on n. □

3.2 The name of a cladogram

The properties of the name of a cladogram are now discussed. In particular, the names satisfy a natural deletion property and are easily classified (as the set given in Definition 7). An inverse function which takes the name of a cladogram and produces the cladogram is also provided.

Definition 11 *Define the deletion operation $D_n : Name(n) \to Name(n-1)$ on the set of names of cladograms as follows: $D_n(s)$ is the string s with the symbols n and $\bar{n}$ removed. (It is clear from the definition above that if $s \in Name(n)$ then $D_n(s) \in Name(n-1)$)*

Let b_n denote the function which takes a cladogram with n leaves and returns its name (the output of algorithm nameOfCladogram).

Proposition 12 *The names of cladograms are deletion stable: For a cladogram t with n leaves, $D_n b_n t = b_{n-1} D_n t$.*

Proof. First, recall Corollary 9: the bar labeling is deletion stable. Let t be a cladogram with n leaves, and l_n the bar labeling function. By Corollary 9, the labeled trees t and $D_n t$ are identical except that $D_n t$ has leaf n and its parent vertex $\bar{n}$ deleted. Thus, for t and $D_n t$, the traversal and recording of vertices in Line 6 of algorithm nameOfCladogram for $i = 1, \ldots, n-1$ is identical except that vertex $\bar{n}$ is missing in the case of $D_n t$. In the case of t, the leaf n is also recorded after all of this. In other words, the only difference in the two strings is that the string produced for $D_n t$ is missing $\bar{n}$ and n. □

For following fact is comforting to know:

Proposition 13 *There are exactly $(2n-3)!!$ elements in the set of names of cladograms (Definition 7) with $n \geqslant 2$ leaves.*

Proof. This is true for $n = 2$ as there is one string, $\bar{2}2$, and $(4-3)!! = 1$. Suppose the statement is true for all $n < k$.

Note that D_k is surjective. In other words, for every string s' in Name$(k-1)$ there is a string s in Name(k) such that $D_k s = s'$. In particular, s may be the string s' with string $\bar{n}n$ appended to its end.

Now consider the inverse image under D_k of any string s' in Name$(k-1)$. For any string $s \in D_k^{-1} s'$, the symbol n must be the last symbol in s. On the other hand, the symbol $\bar{n}$ may occur in any of $2k-3$ positions before n. Since these are the only choices to be made, there must be exactly $(2k-3)$ strings in the inverse image $D_k^{-1} s'$.

Thus the proposition follows by induction. □

The following proposition shows that these so-called names of cladograms are actually the strings produced by the algorithm *nameOfCladogram*, without the initial $\bar{1}1$. Let the set of strings produced by the algorithm *nameOfCladogram* henceforth refer to these modified strings which have the leading $\bar{1}1$ removed (in other words, skip line 8 and the first visit to line 10 in the algorithm) .

Proposition 14 *The algorithm* nameOfCladogram *provides a bijection from the set of cladograms with n leaves to the set of names set Name(n) of* names of cladograms *with n leaves given in Definition 7.*

Proof. Proceed by induction. Applying the algorithm to the unique 2 leaf cladogram produces the string $\bar{2}2$ as required.

Assume that the statement holds for all cladograms with k leaves for $2 \leqslant k < n$.

Let s be a string which is in the set of *names of cladograms* with $n > 2$ leaves. Let $s' = D_n s$ be the string s with n and $\bar{n}$ deleted. By the inductive assumption, there exists a tree t' which has name s'.

Let x be the symbol in s which is just before symbol $\bar{n}$. Let t be the cladogram created by inserting leaf n into the edge above the vertex of t' labeled x. The claim now is that cladogram t has name s.

By the previous proposition, the the name of t with $\bar{n}$ and n deleted, $D_n l_n t$, is the name of $t' = D_n t$. The symbol n is necessarily the last symbol in the name of t. The symbol $\bar{n}$ occurs somewhere in the string. This is because the number of internal vertices below $\bar{n}$ is one less than the number of leaves (not including n) so there must be some leaf k below $\bar{n}$ for which $\bar{k}$ lies above $\bar{n}$.

Therefore, the symbol $\bar{n}$ must be located immediately after the symbol of its child which is not leaf n, by Line 6 of algorithm nameOfCladogram. Thus, the name of t is precisely string s.

Therefore, since there are exactly $(2n-3)!!$ names of cladograms with n leaves and $(2n-3)!$ cladograms with n leaves, the algorithm nameOfCladogram is a bijection from cladograms with n leaves to names of cladograms (given by Definition). □

Notice that this proof provides an indication of how to recursively build a cladogram from its name.

A non-recursive algorithm for taking a name and returning the corresponding cladogram is now presented:

Algorithm: cladogramOfName
Input: a string s satisfying the conditions of Proposition 14.
Output: a cladogram with n leaves.

1: Append symbol 1 to the beginning of string s.
2: Create a vertex and label it 1.
3: Set variable v this vertex 1 just created.
4: Create a vertex and label it $\bar{1}$.
5: Set variable u to be this vertex $\bar{1}$ just created.
6: Set variable r to be this vertex $\bar{1}$ just created (the root).

7: **for** symbol x in string s **do**
8: **if** x is a barred symbol **then**
9: Create a vertex labeled x and join x to v with an edge.
10: Set variable v to be this vertex labeled x just created.
11: **else**
12: Join vertex v to the vertex u.
13: Create a vertex labeled x.
14: Set v to be this vertex labeled x just created.
15: Set u to be the vertex with label $\bar{x}$ (which has already been created by the properties required of the string s).
16: **end if**
17: **end for**
18: Join the vertex labeled n to vertex labeled $\bar{n}$ with an edge.
19: Return the constructed tree, rooted at vertex r.

Let h_n denote the putative map from names of cladograms with $n \geqslant 2$ leaves to cladograms with n leaves given by the above algorithm.

Proposition 15 *Given a string, $s \in Name(n)$, which is the name of a cladogram with n leaves, the algorithm* cladogramOfName *produces a cladogram with n leaves.*

Proof. First, note that the graph produced by algorithm *cladogramOfName* is a tree. The graph has no loops because each vertex created is attached to at most one previously created vertex. Also, at all times, the graph consists of at most two connected components, one of which is the chain currently under construction (containing vertex v) which is connected to the other component (containing vertex u) upon completion of the chain (line 12).

Next, there is a bijection between the vertices of this tree and the symbols in the given string (with 1 and $\bar{1}$ adjoined) since exactly one new vertex is created for each symbol read from the string.

The vertex labeled x, for $x \in \{1, 2, \ldots, n\}$, has degree 1. This follows since once it is created (line 13), variable v is assigned to this vertex (line 14) then at the next visit to line 9 of line 12 it is joined to another vertex. Variable v is then immediately reassigned (line 10 or line 14) and the vertex labeled x is never referenced again.

The vertex labeled $\bar{x}$, for $\bar{x} \in \{\bar{2}, \ldots, \bar{n}\}$, has degree 3. The proof is as follows: Once the vertex labeled $\bar{x}$ is created, it is immediately connected to another vertex (line 9). Variable v is then assigned to x (line 10). On the next visit to line 9 or line 12 x is connected to another vertex, and variable v is immediately reassigned (line 10 or line 14). Since symbol x comes after symbol $\bar{x}$ in the string, when the vertex labeled x is created (line 13), variable u is then set to $\bar{x}$ (line 15). On the next visit to line 12, $\bar{x}$ is connected to a new vertex and variable u is then reassigned (line 15) and $\bar{x}$ is never referenced again.

Therefore, the graph output by the algorithm is rooted, binary tree with n leaves labeled $\{1, 2, \ldots, n\}$. In other words, the output is a cladogram with n leaves. □

Let $b_n : Cl(n) \to \text{Name}(n)$ denote the map from cladograms to names given by algorithm nameOfCladogram.

Proposition 16 *The algorithm* cladogramOfName *is the inverse of algorithm* nameOfCladogram*. In other words, for any cladogram t with n leaves $h_n b_n(t) = t$ and for any string s which is in the set of names of cladogram with n leaves $b_n h_n(s) = s$.*

Proof. Let t be a cladogram with n leaves and s its corresponding name, given by algorithm nameOfCladogram.

Now, for each $k \in \{1, \ldots, n-1\}$, the algorithm cladogramOfName constructs chains with vertices labeled by the symbols in the string s from symbol k to the symbol before $k+1$. The top of each such chain (the end which is barred; ie. not k) is attached with an edge to vertex $\bar{k}$. Thus, following the shortest path from leaf k to vertex $\bar{k}$ in the resulting tree, as in the last step of algorithm *nameOfCladogram*, gives the desired substring of s between symbols k and $k+1$.

Therefore $b_n h_n b_n(t) = b_n(t)$. Since b_n is a bijection (Corollary 14), h_n must be its inverse. □

Figure 9 shows the labeling of the cladogram in Figure 1 and the resulting name. Figures 10 to 15 show the labelings and names for the cladograms obtained by successive deletions of the largest remaining leaf.

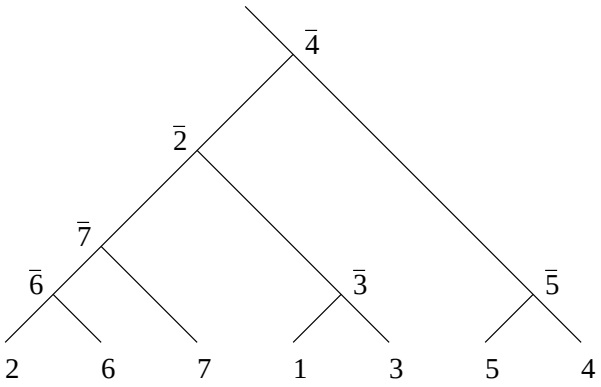

Figure 10: The cladogram with name (1)$\bar{3}\bar{2}\bar{4}2\bar{6}\bar{7}34\bar{5}567$

3.3 Perfect matchings

This section covers the construction of a perfect matching from the name of a cladogram and some of its properties. In particular, deleting the largest leaf of a cladogram corresponds to deleting the last point in the matching and its paired point.

To convert the name of a cladogram with n leaves to a perfect matching on $2(n-1)$ points, first label the points in order by the symbols in the name then for each $k = 2, \ldots, n$ pair point $\bar{k}$ with point k. In other words, lay $2n-2$ points in a line and connect the i-th

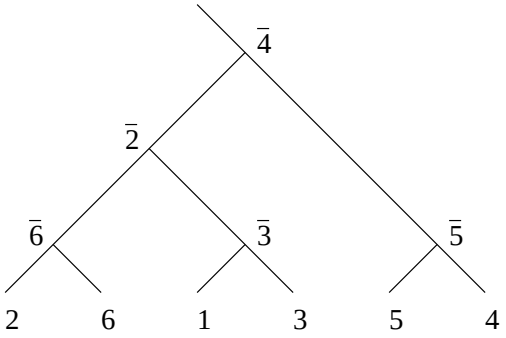

Figure 11: The cladogram with name $(1)\bar{3}\bar{2}\bar{4}2\bar{6}34\bar{5}56$

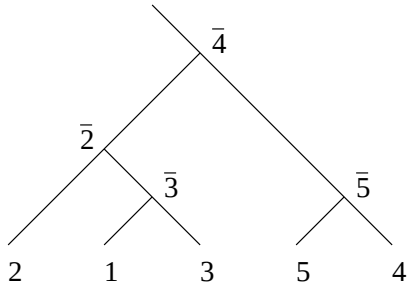

Figure 12: The cladogram with name $(1)\bar{3}\bar{2}\bar{4}234\bar{5}5$

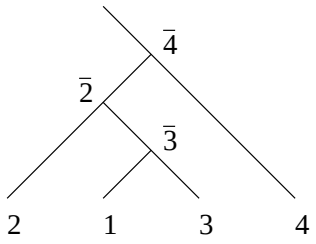

Figure 13: The cladogram with name $(1)\bar{3}\bar{2}\bar{4}234$

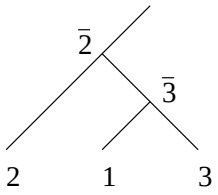

Figure 14: The cladogram with name $(1)\bar{3}\bar{2}23$

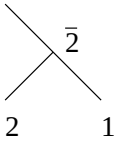

Figure 15: The cladogram with name $(1)\bar{2}2$

and j-th point if symbol $\bar{k}$ and k are in position i and j in the name. See Figure 16 for an example. Note that points in the perfect matching are identified by their position in the linear ordering and the labeling of the points is not part of the matching.

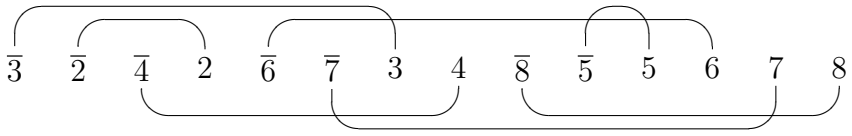

Figure 16: The perfect matching with name $(1)\overline{324}\overline{26}\overline{7}34\overline{85}5678$. The corresponding cladogram is shown in Figure 9. The labeling of the ordered points in the matching is simply to aid understanding its construction and this is the same matching as shown in Figure 2.

Conversely, given a perfect matching on $2(n-1)$ points, start with the last point and label it n and label its paired point $\bar{n}$. Continue in this was backwards through the points, labeling each unlabeled point by the next highest unused label in $\{n-1,\ldots,1\}$, say k, and labeling its paired point with the corresponding $\bar{k}$. The name of the cladogram is now read off from the first point to the last.

These two operations are inverse and form a bijection between names of trees with $n \geqslant 1$ leaves and perfect matchings on $2(n-1)$ points. Denote this map from names to perfect matchings p_n.

This mapping is deletion stable in the following sense. For all $n \geqslant 2$, let D_n be a function from perfect matchings on $2(n-1)$ points to perfect matchings on $2(n-2)$ points which acts as follows: Remove the last point of the matching and its paired point.

With this definition of deleting the 'largest pair' from a perfect matching, we have the desired deletion stability: $D_n p_n s = p_{n-1} D_n s$ for all

Since 'deletion' is preserved by the mappings from cladograms to names and from names to perfect matchings, it is preserved by the composite bijection from cladograms to perfect matchings.

3.4 Fat/oriented cladograms and cladograms with edge lengths

This section discusses a simple alteration to the previous encoding so that it encodes fat cladograms. Recall that a *fat cladogram*, or *oriented cladogram* is a cladogram together with a cyclic ordering of the edges at every vertex. In other words, the 'left' and 'right' child of a vertex are distinguished from each other.

The encoding for fat cladograms is as follows: Label the internal vertices as before, with algorithm barLabeling.

Read the labels as before, but this time record internal vertex k-bar as $\overline{k}$ if it is encountered coming from the left child and $\underline{k}$ if it is encountered coming from the right child. In other words, the label of an internal vertex v which would have previously been labeled $\bar{k}$ is $\overline{k}$, respectively $\underline{k}$, if the leaf k is in its left, respectively right, subtree below it.

This gives a bijection between fat cladograms and a certain set of strings, called *names of fat cladograms*.

Definition 17 *Define the set of* names of fat cladograms *with* $n \geqslant 2$ *leaves, denoted* $FatName(n)$*, to be the set of strings satisfying the following three conditions:*
1 - Each of the symbols $2, 3, \ldots, n$ *occurs exactly once in the string and for each* $k \in \{2, \ldots, n\}$ *exactly one of the symbols* $\bar{k}$ *or* $\underline{k}$ *occurs. No other symbols occur.*
2 - If $k < l$ *then symbol* k *occurs to the left of symbol* l *in the string*
3 - If a symbol $\bar{k}$ *or* $\underline{k}$ *occurs it is to the left of the symbol* k*.*

The name of a fat cladogram is also deletion stable in the sense that removing leaf n corresponds to deleting from the name symbols n and either $\bar{n}$ or $\underline{n}$ depending on which occurs. An inverse function which creates a fat cladogram from it's name would be very similar to that for ordinary cladograms given in Section 3.2. This algorithm, the proof of the bijection and correctness of the definition are omitted as they are almost identical to those for cladogram case.

Figure 17 shows the name associated with a fat cladogram with 8 leaves. Compare this with the name of the thin cladogram in Figure 9.

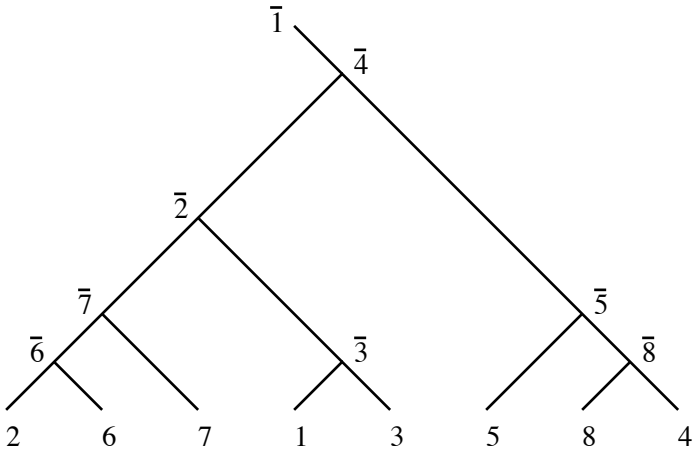

Figure 17: The fat cladogram with name $(1)\bar{3}\underline{2}\bar{4}2\bar{6}\bar{7}34\underline{8}\underline{5}5678$

A *directed perfect matching* on $2k$ points is a pairing of these points, such that every point belongs to exactly one pair, along with a sign of ± 1 assigned to each pairing. This sign on a pair (a, b) may be thought of as a direction on an edge between a and b.

The name of a fat cladogram with n leaves corresponds naturally to a *directed perfect matching*. The structure of the undirected perfect matching is as before and the direction/sign of each edge/pair is determined by whether the bar is above or below k-bar: *out of* $\bar{k}$ and *into* $\underline{k}$ (or $+1$ for a pair with $\bar{k}$ and -1 for a pair with $\underline{k}$)

Figure 18 shows the directed perfect matching on $14 = 2*8-2$ points corresponding to the name $(1)\bar{3}\underline{2}\bar{4}2\bar{6}\bar{7}34\underline{8}\underline{5}5678$. Remember that the ordering of the points in the diagram is what is important. The labeling of the points is not part of the matching.

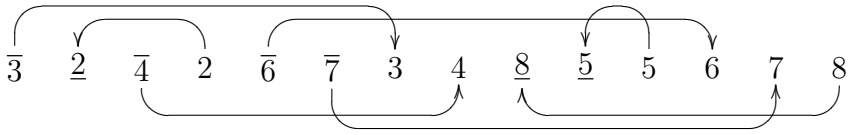

Figure 18: The directed perfect matching with name $(1)\overline{3}\underline{2}\overline{4}2\overline{6}\overline{7}34\underline{8}55678$. The labeling of the ordered points in the matching is simply to aid understanding its construction. Figure 17 shows the corresponding fat cladogram.

Again, this bijection from 'oriented' names to directed perfect matchings preserves deletion (as defined earlier for names and perfect matchings). Thus the composite bijection from fat/oriented cladograms to directed perfect matchings also respects these deletion maps.

Recording edge lengths in the bar coding is similar to the case of the Diaconis-Holmes and hat encodings. The length of the edge above each vertex is recorded immediately after the vertexes label in the name. In this case, the length of the root edge is recorded. For example, the cladogram in Figure 17 with edge lengths proportional to their apparent length has name

$$(1):1,\overline{3}:3,\underline{2}:3,\overline{4}:1,2:1,\overline{6}:1,\overline{7}:2,3:1,4:1,\underline{8}:1,\underline{5}:5,5:1,6:1,7:2,8:1$$

$$(1)_1\overline{3}_3\underline{2}_3\overline{4}_1 2_1,\overline{6}_1\overline{7}_2 3_1 4_1\underline{8}_1\underline{5}_5 5_1 6_1 7_2 8_1$$

Notice that when deleting the largest leaf, n, the edge lengths of the resulting tree are gained by discarding the edge length for n and adding the length of $\overline{n}$ to the length of the symbol immediately preceding it. In the example above, the length of $\overline{8}$ is added to the length of 4, for a combined length of 2.

3.5 Adjoining two trees

This section describes how to construct the name of a tree gained by joining two trees at their roots. Recall that algorithm nameOfCladogram makes sense for any rooted binary leaf-labeled tree with distinctly labeled leaves from a total ordering.

Consider two trees with disjoint sets of leaf labels from the same totally ordered set and names $(a_0)a_1a_2\ldots a_k$ and $(b_0)b_1b_2\ldots b_l$, To construct the name of their root-join, begin by breaking each name into its disjoint blocks. Each block starts with a leaf label and ends with the last symbol before the next leaf label, or the end of the name. Without loss of generality, let symbol a_0 (the smallest leaf label of the first tree) be less than symbol b_0 (the smallest leaf label of the second tree). Append symbol $\overline{b_0}$ to the block starting with a_0. Finally, reassemble all of the blocks according to the ordering of their initial symbols (the leaf labels). This is the name of the root join of the two initial trees.

The following is an example using two trees with disjoint sets of integer leaf labels. Figures 19 and 20 show the effect of adjoining these two trees at the root.

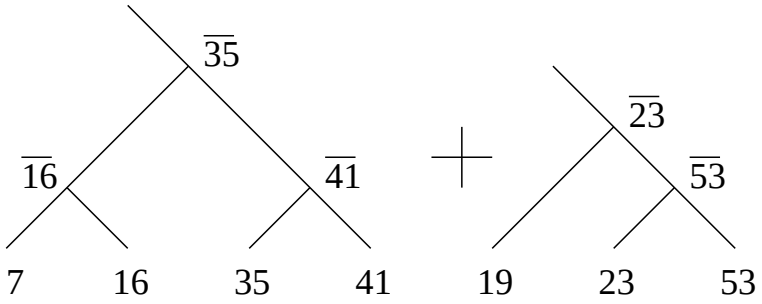

Figure 19: Trees '$(7), \overline{10}, \overline{35}, 10, 35, \overline{41}, 41$' and '$(19), \overline{23}, 23, \overline{53}, 53$'

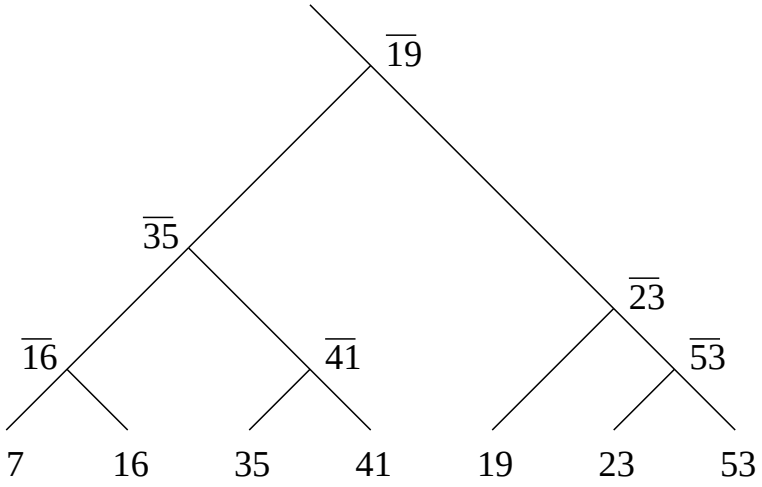

Figure 20: The tree named '$7, \overline{10}, \overline{35}, \overline{19}, 10, 19, \overline{23}, 23, \overline{53}, 35, \overline{41}, 41, 53$'

Given trees with names $t_a = (7), \overline{10}, \overline{35}, 10, 35, \overline{41}, 41$ and $t_b = (19), \overline{23}, 23, \overline{53}, 53$, begin by breaking each name into its blocks. Each *block* starts with a leaf label and proceeds until the symbol before the following leaf label. In this case the blocks are:

$$7, \overline{10}, \overline{35} - 10 - 35, \overline{41} - 41 - 19, \overline{23} - 23, \overline{53} - 53$$

Since $7 < 19$ the new internal vertex where the trees are joined will have label $\overline{19}$ so add this to the end of the 7-block:

$$7, \overline{10}, \overline{35}, \overline{19} - 10 - 35, \overline{41} - 41 - 19, \overline{23} - 23, \overline{53} - 53$$

Finally, reassemble the blocks in order to get the name of the root-joined tree:

$$7, \overline{10}, \overline{35}, \overline{19}, 10, 19, \overline{23}, 23, \overline{53}, 35, \overline{41}, 41, 53$$

3.6 Moving a symbol in the name of a tree

This section demonstrates the effect of changing the position of a symbol $\bar{k}$ in the name of a cladogram. Recall that the only conditions on the name of a cladogram are that the symbols $2, \ldots, n$ occur in this relative order and symbol $\bar{k}$ occurs before symbol k.

Moving a symbol $\bar{k}$ from one position in the name of a cladogram to another allowable position corresponds to pruning the subtree below vertex $\bar{k}$ which contains vertex k and regrafting it onto another edge of the tree. The edge which the subtree is regrafted onto is the edge immediately above the vertex with label occurring just before $\bar{k}$'s new position.

These are *branch-regraft* moves, where the subtree below an edge is pruned and regrafted onto the original tree in a different position are usually called *subtree prune and regraft* (SPR) moves, [8]. A special case of SPR moves are *nearest neighbor interchange* (NNI) moves, described in [9], [2], [3] and [1]. SPR moves are also a subset of *tree bisection and reconnection* (TBR) moves, which were first described in [21] and are also discussed in [9], [1] and [18]. Each of these three types of moves are implemented in a number of popular phylogenetic software packages.

For example, Figure 21 shows the effect of taking the name of the cladogram in Figure 9 and moving symbol $\bar{4}$ to a new position. Notice that the subtree below vertex $\bar{4}$ containing leaf 4 has been pruned and regrafted into the edge which was between vertices $\bar{6}$ and $\bar{7}$.

Note that the labels of all vertices in the pruned subtree are of the form $\bar{j}$ or j for some $j > k$. Thus, since symbol $\bar{k}$ must be reinserted into a position before symbol k, the regrafting position must be on the remaining tree and not the pruned subtree.

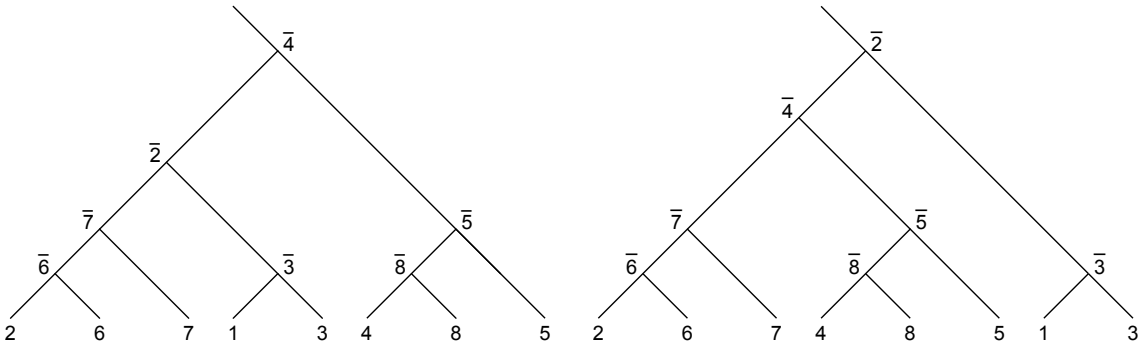

Figure 21: Cladograms with names $(1)\bar{3}\bar{2}\bar{4}2\bar{6}\bar{7}34\bar{8}\bar{5}5678$ and $(1)\bar{3}\bar{2}2\bar{6}\bar{4}\bar{7}34\bar{8}\bar{5}5678$

4 Maxmin labeling

This section introduces *maxmin labeling*. Both the hat and bar labelings are special cases of maxmin labeling, after making a choice for the label of the root vertex.

Given an internal vertex, v, of a tree, removing this vertex breaks the tree into a number of connected subtrees equal to its degree. Call these the *component subtrees* of vertex v.

Definition 18 *A total labeling of a finite leaf labeled tree with leaves labeled from a totally ordered set such as* $\mathbb{N} \cup \{\infty\}$ *is a* maxmin labeling *if each internal vertex, v, has label $\bar{k}$, where k is the maximum of the minimum leaf label in each component subtree of v.*

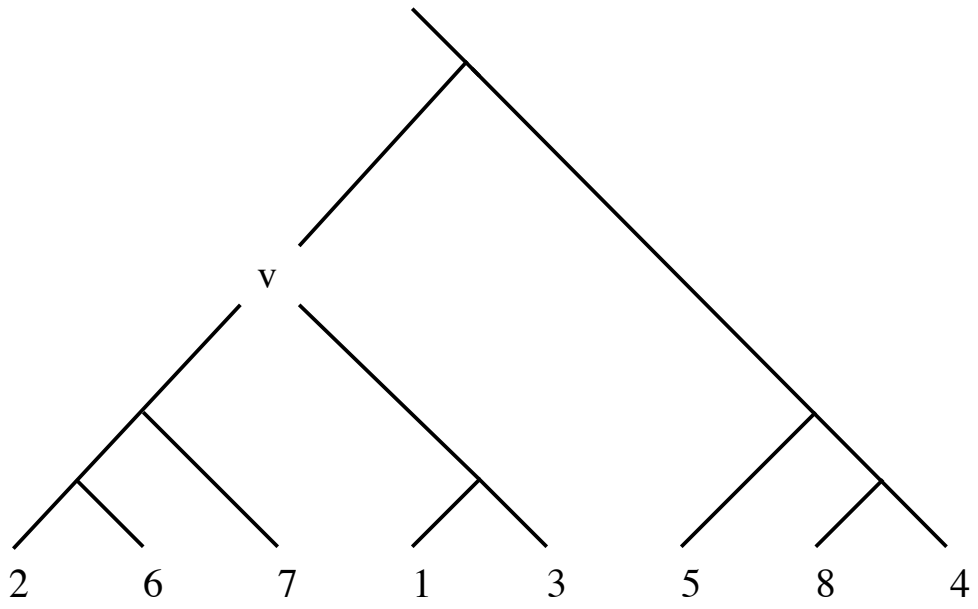

Figure 22: The three component subtrees corresponding to vertex v.

Proposition 19 *Given a maxmin labeled finite bifurcating tree with all leaves labeled distinctly from a totally ordered set, such as $\mathbb{N} \cup \{\infty\}$, each internal vertex is distinctly labeled. Furthermore, the only leaf labels, k, for which a corresponding internal labels, $\bar{k}$, do not occur are the two smallest leaf labels.*

Proof. Prove the proposition by contradiction.

Suppose that such a maxmin labeled tree exists, with two internal vertices x and y with identical labels $\bar{k}$. The vertices x and y naturally divide the tree into five pieces, as indicated in Figure 23. Note that the piece lying between x and y may be an empty tree.

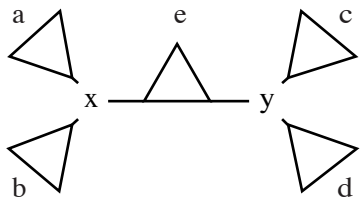

Figure 23: The five disconnected subtrees gained by removing vertices x and y. The smallest label in each subtree, if it exists, is called a, b, c, d, e respectively.

Let a, b, c, d, and e if it exists, denote the smallest leaf label in each of the four (or five) connected subtrees gained by removing vertices x and y. Since $\bar{k}$ is the label for x, $k \geqslant a, b$ and $k \geqslant e$ if e exists. On the other hand, since $\bar{k}$ is the label of y, $k \leqslant a, b$ and

$k \leqslant e$ if e exists. Therefore $k = a, b$, and if e exists then $k = e$. This contradicts the assumption that all leaves are distinctly labeled.

Finally, the two smallest leaf labels do not occur, since for any internal vertex, they must always be less than the smallest leaf label in one of the other component subtrees for that vertex. Since there are exactly 2 fewer internal vertices than leaf vertices, uniqueness implies that labels corresponding to all other leaf labels must occur exactly once. □

In fact, the uniqueness of internal vertex labels remains true for trivalent trees with leaves distinctly labeled from a totally ordered set, whether or not the tree is finite.

4.1 Maxmin labeling and the hat labeling

The hat labeling, from Section 2, is a special case of the maxmin labeling once the root vertex is labeled ∞. When comparing the maxmin labeling and the hat labeling, $\hat{k}$ and $\bar{k}$ are treated as synonymous.

Proposition 20 *Consider a cladogram labeled according to the hat labeling. Labeling the root vertex ∞ and the root branch point $\hat{\infty}$ makes this a maxmin labeling.*

Proof. The proposition is trivially true for a cladogram with one leaf, since there are no internal nodes to be labeled. Assume that the proposition is true for all cladograms with less than $n \geqslant 2$ leaves.

Now consider a cladogram t with n leaves. As in the algorithm for hat labeling, let $t_n = t$ denote the whole tree with n leaves and t_{n-1} denote the tree gained by deleting the leaf labeled n. Recall that there is a single non-root branch point of t_n which does not correspond to a non-root branch point of t_{n-1} (via the natural mapping shown in Figure 4).

The addition of a leaf labeled n does not effect the label of any of the non-root branch points coming from t_{n-1} because they all have labels $\hat{k}$ where $k < n$. This is because the minimum label of all of their component subtrees is at most $k < n$ and this minimum cannot be effected by the addition of a leaf with label $n > k$ to any of them.

The root branch point is labeled $\hat{\infty}$ because one of it's component subtrees comprises the single root vertex with label ∞. Proposition 19 guarantees that there is some internal vertex labeled n, so this must be the new non-root branch point in t_n which was not a non-root branch point in t_{n-1}. This is exactly the labeling given by the hat bijection. The proposition now follows by induction on n. □

4.2 Maxmin labeling and the bar labeling

The bar labeling, from Section 3, is a special case of the maxmin labeling if the root label $\bar{1}$ is considered as smaller than all leaf labels.

Proposition 21 *The bar labeling of a cladogram is a maxmin labeling, if the root label $\bar{1}$ is considered smaller than all leaf labels.*

Proof. The proposition is trivially true for the cladogram with one leaf, since there are no internal leaves to label. Let $n \geqslant 2$ and suppose the proposition is true for all cladograms with less than n leaves.

Let t be a cladogram with n leaves and t_{n-1} denote the cladogram gained by deleting the vertex labeled n from t. There is one internal vertex of t which does not correspond to an internal vertex of t_{n-1} via the natural mapping shown in Figure 4. This is the parent of leaf n. For every other internal vertex, v, the label given by the maxmin labeling remains unchanged from t_{n-1} to t because the addition of the new leaf labeled n to any of the component subtrees of v does not change the minimum leaf label in that subtree (since n is the largest leaf). By the inductive assumption, these are precisely the labels given to these internal vertices by the bar labeling.

Finally, the internal vertex adjacent to leaf n must have label $\bar{n}$ since this the the minimum leaf value of one of it's component subtrees and also the maximum possible label. This is precisely the label given to this internal vertex by the bar labeling.

The proposition now follows by induction on n. □

4.3 Exchanging the labels of two leaves

This section describes the effect on the maxmin labeling of exchanging the labels of two leaves on a tree.

Before beginning, define a *cherry* to be a pair of leaves which have a common immediate parent node. For example, in Figure 22 leaves 1 and 3 form a cherry.

Proposition 22 *If the labels of leaves k and $k+1$ are exchanged then all internal labels of the max-min labeling remain unchanged except, possibly, the vertices labeled $\bar{k}$ and $k \bar{+} 1$. If $k = 1, 2$ then nothing happens. If $k \geqslant 3$ and k and $k+1$ do not form a cherry then the labels $\bar{k}$ and $k \bar{+} 1$ are swapped*

Proof. This is because every other leaf label is either greater than both k and $k+1$ or less than both k and $k+1$. Thus, if a component subtree has minimum leaf label $x \neq k, k+1$ before the exchange then it still does afterwards. Also, a component subtree with minimum leaf label k or $k+1$ before the exchange will have either k or $k+1$ as it's minimum leaf label after the exchange. If an internal vertex has label $\bar{x}$ before the exchange, with $x \neq k, k+1$, then $x > y, z$ where x, y, z are the minimum leaf labels for the three corresponding component subtrees. After the exchange, x must still be greater than y and z since they are either unchanged or were equal to one of k or $k+1$, which implies $x > k+1$ and they are still either k or $k+1$.

If $k = 1$ then nothing happens, since there are no labels $\bar{1}$ and $\bar{2}$ and all other labels remain unchanged as shown above. If $k = 2$ then all internal labels remain unchanged except possibly $\bar{3}$, and since it must occur at some internal vertex it remains at the only available vertex.

Now suppose that $k \geqslant 3$. If the leaves labeled k and $k+1$ are adjacent then the tree is unchanged and so the maxmin labeling is unchanged. Suppose k and $k+1$ are

not together in a cherry. Consider Figure 23, showing the tree divided into component subtrees by the vertices $x = \bar{k}$ and $y = k \bar{+} 1$. Let a, b, c, d (and e if it exists) denote the smallest leaf labels in each component subtree. Therefore, $a = k - 1$, $b < k + 1$, $d = k$, $c < k$, which means that $b, c < k, k + 1$ since $b \neq k$. Therefore, swapping leaf labels k and $k + 1$ swaps labels $\bar{k}$ and $k \bar{+} 1$. □

What about exchanging two general labels?

Proposition 23 *If the labels of two leaves are exchanged then the label of any internal node not along the shortest path between these two leaves remains unchanged.*

Proof. For any internal vertex, v, not on the shortest path between these two leaves, the two leaves are contained in the same component subtree of that vertex and so exchanging their labels does not change the minimum leaf label of that component subtree. Therefore, the maxmin labeling of vertex v remains unchanged. □

5 A Gray code for cladograms

This section describes a perfect Gray code for cladogram names based on the bar coding. This is a systematic sequence which contains the name of every cladogram with n leaves exactly once (Gray code) and for which adjacent elements differ by a minimal amount (perfect). The names of adjacent cladograms in this Gray code differ by a single transposition of symbols. The corresponding cladograms differ by a single branch-regraft move, as described in Section 3.6. See [11] for a definition and discussion of perfect Gray codes. A recursive construction is given here, as well as an algorithms for finding the next and previous cladogram, the number of a cladogram in the Gray code, and the cladogram corresponding to a given number. Gray codes exist for other sets of trees, such as binary trees with n leaves [17]. The Combinatorial Object Server [16] uses Gray codes to provide lists of many types of objects but does not yet provide cladograms.

5.1 Defining the Gray code

This section gives a recursive definition of the perfect Gray code which flows naturally from the bar encoding. In general, a perfect Gray code is a sequential enumeration of a set so that adjacent elements of the sequence differ by 'a small amount'. Gray codes were first developed by Frank Gray [10] for the purposes of minimizing error in digital signal transmissions over analogue channels. In the context of cladograms, they provide an easy way of iterating over all cladograms of a particular size. This is useful, for example, when computing the exact distribution of statistics on trees, such as the number of cherries or the average depth, or the exact likelihood of observed data, such as a DNA sequence at each leaf, given a prior on the set of cladograms.

Diaconis and Holmes [7] indicate a Gray code for cladograms which flows from a Gray code on perfect matchings. This Gray code on perfect matchings similarly defines a Gray

code on cladograms for each of the two new bijections presented above. However, this section is devoted to a perfect Gray code for the name of a Cladogram.

Let $Cl(n)$ denote the set of cladograms with n leaves. This Gray code has the property that the code for $Cl(n+1)$ when projected to $Cl(n)$ by deletion of the largest leaf is given by taking the Gray code for $Cl(n)$ and, starting with the first element, repeating an element of the code $(2n+1)$ times in succession before moving onto the next element.

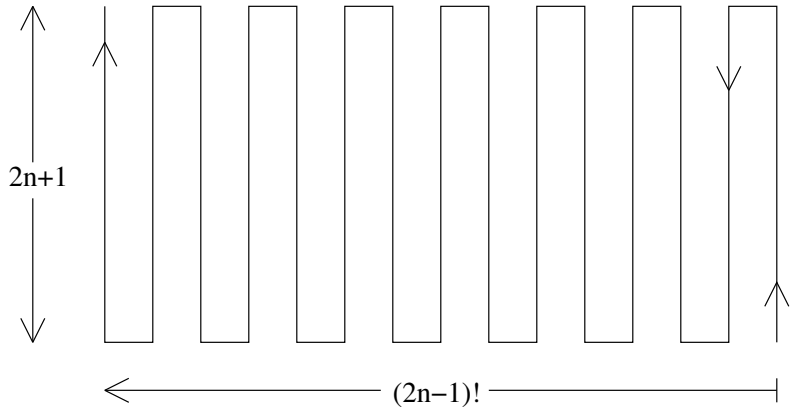

Figure 24: How the Gray code for trees of with $n+2$ leaves is related to the code for trees with $n+1$ leaves.

Call a tree *odd*, respectively *even*, if its position in the Gray code is odd, respectively even. As is often the case, counting starts at 0 rather than 1 as this makes later definitions cleaner. Thus, the first cladogram is even. The sign of a cladogram is discussed in more detail in the next section.

An algorithm for constructing the Gray code, for cladograms of size n.
Algorithm: GrayCode
Input: a positive integer n.
Output: A sequence of names of cladograms of size n.

1: **if** $n == 1$ **then**
2: Return a sequence with one element, the empty name (corresponding to the unique one-leaf cladogram), and terminate.
3: **end if**
4: Let s be an empty sequence.
5: **for** $i = 0$ to $(2n-5)!! - 1$ **do**
6: Let x be the i-th name in sequence GrayCode$(n-1)$.
7: **for** $j = 1$ to $2n-3$ **do**
8: Let y be a copy of name x.
9: Let $k = 2n-2-j$ if x is even (ie. i is even) and $k = j$ otherwise.
10: Append symbol n to y. Insert symbol $\overline{n}$ immediately before the k-th symbol of y.
11: Append name y to sequence s.

12: **end for**
13: **end for**
14: Return sequence s.

The Gray codes for cladograms with 1 to 4 leaves are:

$$\begin{aligned}
\text{GrayCode}(1) &= [\ '' \ ] \\
\text{GrayCode}(2) &= [\ \bar{2}2 \ ] \\
\text{GrayCode}(3) &= [\ \bar{3}\bar{2}23, \quad \bar{2}\bar{3}23, \quad \bar{2}2\bar{3}3 \ ] \\
\text{GrayCode}(4) &= [\ \bar{4}\bar{3}\bar{2}234, \quad \bar{3}\bar{4}\bar{2}234, \quad \bar{3}\bar{2}\bar{4}234, \quad \bar{3}\bar{2}\bar{2}\bar{4}34, \quad \bar{3}\bar{2}\bar{2}3\bar{4}4, \\
&\quad \bar{2}\bar{3}23\bar{4}4, \quad \bar{2}\bar{3}2\bar{4}34, \quad \bar{2}\bar{3}\bar{4}234, \quad \bar{2}\bar{4}\bar{3}234, \quad \bar{4}\bar{2}\bar{3}234, \\
&\quad \bar{4}\bar{2}2\bar{3}34, \quad \bar{2}\bar{4}2\bar{3}34, \quad \bar{2}2\bar{4}\bar{3}34, \quad \bar{2}2\bar{3}\bar{4}34, \quad \bar{2}2\bar{3}3\bar{4}4 \ ]
\end{aligned}$$

Cladograms which are adjacent in the sequence have names which differ only by a single transposition. As noted in Section 3.6, this means they differ by a single branch-regraft move: a single branch has been pruned and regrafted onto the cladogram in a different position.
In general, the first cladogram in the Gray code GrayCode(n) is the comb
$(1)\bar{n}\overline{(n-1)}\ldots\bar{3}\bar{2}23\ldots(n-1)n$
and the last cladogram is the comb
$(1)\bar{2}2\bar{3}3\ldots\overline{(n-1)}(n-1)\bar{n}n$.
See Figures 25 and 26 for examples of the first and last trees in the Gray code for cladograms with 6 leaves.

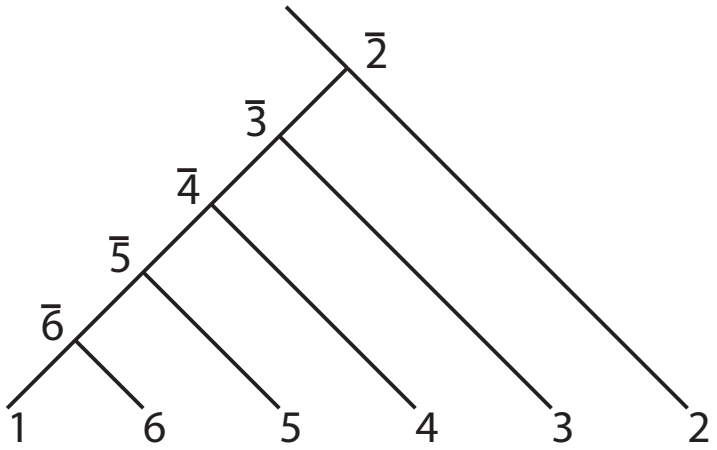

Figure 25: The first cladogram, $(1)\bar{6}\overline{\bar{5}\bar{4}\bar{3}\bar{2}}23456$, in the Gray code for cladograms with 6 leaves, GrayCode(6)

These trees differ only in the position of the leaf 1 (ignoring the internal node labels): leaf 1 has moved from the bottom of the comb to the top. Thus, the sequence GrayCode(n), for $n > 2$, makes a cycle where adjacent cladograms differ by exactly one branch-regraft move.

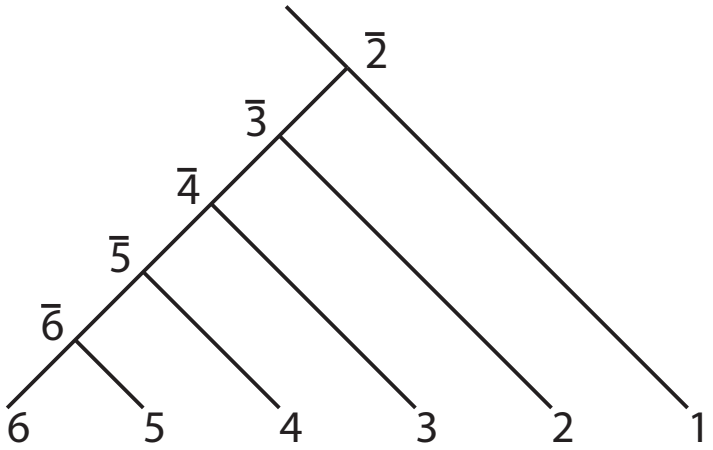

Figure 26: The last cladogram, $(1)\bar{2}2\bar{3}3\ldots\bar{6}6$, in the Gray code for cladograms with 6 leaves, GrayCode(6)

Finally, these Gray codes for each $n > 0$ give a way to enumerate all non-empty cladograms in a single sequence. Start with the cladogram with 1 leaf, then append the cladogram with 2 leaves, then append the Gray code for cladograms with 3 leaves, then the reversal of sequence GrayCode(4), then GrayCode(5), then the reversal of GrayCode(6) and so on, alternating direction each time. The inductive construction ensures that the transitions between cladograms with n leaves and those with $n+1$ leaves involves only the insertion of the new leaf $n+1$ into the previous cladogram with n leaves. This assigns a unique number to any finite non-empty cladogram, its position in this sequence.

Section 5.4 gives algorithms for determining the position in the Gray code of a cladogram with n leaves, GrayCode(n), given its name, as well as the name of the cladogram in a given position.

5.2 The sign of a cladogram

This section introduces the *sign* of a cladogram, analogous to the sign of a permutation. An easy way to determine the sign of a cladogram, presented below, leads to an easy way to determine the next and previous elements of the Gray code.

The sign of a cladogram was defined in the previous section as the parity of the position of the cladogram in the Gray code: either *even* or *odd*. Recall that the counting starts at 0 so the first cladogram in the sequence is even.

The sign/parity of an n-leaf cladogram is also the sign of the number of adjacent transpositions required to change the name of the cladogram to $\bar{2}2\ldots\bar{n}n$. This is the same as the parity of the number of adjacent transpositions of symbols required to move from the cladogram $(1)\bar{n}\overline{(n-1)}\ldots\bar{3}\bar{2}23\ldots(n-1)n$ to the cladogram in question. The equivalence of these two definitions for the sign of a cladogram follows directly from the corresponding results for representations of the symmetric group with adjacent transpositions and the fact that adjacent names in the Gray code differ by exactly one adjacent transposition.

5.3 The next and previous elements of the Gray code

Now that the sign of a cladogram name is easily determined, the next and previous elements in the Gray code are also easy to determine.

In this section a cladogram and its name are often treated as synonymous. The leading symbol 1 is not included in the names used here. Let sign(t) denote the sign of a cladogram: $+1$ for an *even* cladogram and -1 for an *odd* cladogram.

The algorithm for the next element in the Gray code is as follows:

Algorithm: nextGrayElement
Input: The name, s, of a cladogram t with n leaves.
Output: The name of the next cladogram in the Gray code, or the same string if this is the last element.

1: **if** s is empty **then**
2: Return s
3: **end if**
4: Let $s' = D_n s$, the name s with symbols $\bar{n}$ and n deleted.
5: **if** sign(s') == $+1$ **then**
6: **if** $\bar{n}$ is not adjacent to n in s **then**
7: Move $\bar{n}$ one position to the right in s and return this string.
8: **else**
9: Let s_2 = nextGrayElement(s').
10: Return s_2 with string $\bar{n}n$ added to the end.
11: **end if**
12: **else**
13: **if** $\bar{n}$ is not the first symbol in s **then**
14: Move $\bar{n}$ one position to the left in s and return this string.
15: **else**
16: Let s_2 = nextGrayElement(s').
17: Return s_2 with symbol $\bar{n}$ added to the front and n added to the end.
18: **end if**
19: **end if**

Changing sign(s') == $+1$ to sign(s') == -1 gives the previous element rather than the next one. Correctness of these algorithms follows directly from the recursive definition of the Gray code.

5.4 The number of a cladogram

Following on from the previous section, this section gives a simple recursive algorithm to calculate the number of a given cladogram (its position in the Gray code), a so-called *ranking* algorithm. This algorithm follows directly from the recursive construction of the Gray code. An algorithm to produce the name of the cladogram in position k, an *unranking* algorithm, is also given. Recall that cladograms in the Gray code are counted starting from 0.

Algorithm: numberOfName
Input: The name, s, of a cladogram with n leaves.
Output: The position of this cladogram in the Gray code.

1: **if** s is empty **then**
2: Return 0.
3: **end if**
4: Let k_1 be the position of $\bar{n}$ in string s (counting from 0 to $2n-4$).
5: Let s' be the string given by removing $\bar{n}$ and n from string s.
6: Let $k_2 = \text{numberOfName}(s')$.
7: **if** $\text{sign}(s') == +1$ **then**
8: Return $(2n-3)k_2 + k_1$.
9: **else**
10: Return $(2n-3)k_2 + (2n-4-k_1)$.
11: **end if**

This leads to a simple recursive construction for the name of the cladogram with a give number:
Algorithm: nameOfNumber
Input: A number $n \geqslant 1$ (the number of leaves) and a number k between 0 and $(2n-3)!!-1$ inclusive
Output: The name of the cladogram with n leaves which has position k in the Gray code on cladograms with n leaves.

1: **if** n==1 **then**
2: Return the empty string.
3: **end if**
4: Let $k_2 = \lfloor \frac{k}{2n-3} \rfloor$.
5: Let $s_2 = \text{nameOfNumber}(n-1, k_2)$.
6: Let $k_3 = k - (2n-3)k_2$.
7: **if** $\text{sign}(s_2) == +1$ **then**
8: Let s be string s_2 with symbol $\bar{n}$ inserted into position k_3 and n appended to the end.
9: **else**
10: Let s be string s_2 with symbol $\bar{n}$ inserted into position $2n-4-k_3$ and n appended to the end.
11: **end if**
12: Return string s.

Note: inserting a symbol into position x places it immediately after the x-th symbol in the sequence. Inserting a symbol at position 0 makes it the first symbol.

Correctness of this algorithm follows directly from the recursive definition of the Gray code.

Note that s_2 is the name s with symbols $\bar{n}$ and n removed. In general, given a cladogram with n leaves and number k, the number of the cladogram obtained by deleting

the m largest leaves is $\lfloor \frac{k}{(2n-3)(2n-5)\cdots(2n-1-2m)} \rfloor$.

Acknowledgements

I would like to thank Persi Diaconis and Susan Holmes for their guidance, encouragement and many helpful comments, and Antonio Ramirez, Veronique Godin and the reviewer for their helpful comments. This work grew out of a homework exercise in Persi's combinatorics class.

References

[1] Benjamin L. Allen and Mike Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5:2001, 2000.

[2] Oliver Bastert, Dan Rockmore, Peter F. Stadler, and Gottfried Tinhofer. Landscapes on spaces of trees. *Journal of Applied Mathematics and Computation*, 131(2–3):439–459, September 2002.

[3] Louis J. Billera, Susan P. Holmes, and Karen Vogtmann. Geometry of the space of phylogenetic trees. *Adv. in Appl. Math.*, 27(4):733–767, 2001.

[4] Maria Luisa Bonet, Katherine St. John, Ruchi Mahindru, and Nina Amenta. Approximating subtree distances between phylogenies. *J. Comput. Biol.*, 13(8):1419–1434 (electronic), 2006.

[5] L.L. Cavalli-Sforza and A.W.F Edwards. *Evolution*, 21:550–570, 1967.

[6] Persi Diaconis and Susan P. Holmes. Random walks on trees and matchings. *Electron. J. Probab.*, 7:no. 6, 17 pp. (electronic), 2002.

[7] Persi W. Diaconis and Susan P. Holmes. Matchings and phylogenetic trees. *Proc. Natl. Acad. Sci. USA*, 95(25):14600–14602 (electronic), 1998.

[8] Steven N. Evans and Anita Winter. Subtree prune and regraft: a reversible real tree-valued markov process. *Annals of Probability*, (34):918–961, 2006.

[9] Joe Felsenstein. *Infering Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.

[10] Frank Gray. *Bell Systems Technical Journal*, 18:252, 1939.

[11] Donald E. Knuth. *The Art of Compute Programming: Volume4, Fascicle 4: Generating all Trees – History of Combinatorial Generation*. Addison-Wesley, 2006.

[12] Wayne P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

[13] Wendy Myrvold and Frank Ruskey. Ranking and unranking permutations in linear time. *Information Processing Letters*, 79:281–284, 2001.

[14] Emmanuel Paradis et al. Ape (analysis of phylogenetics and evolution) v1.8-2, 2006. `http://cran.r-project.org/doc/packages/ape.pdf`.

[15] R. C. Penner. Perturbative series and the moduli space of riemann surfaces. *J. Differential Geom.*, 27(1):35–53, 1988.

[16] Frank Ruskey. Combinatorial object server. `http://www.theory.csc.uvic.ca/~cos/`.

[17] Frank Ruskey and Aaron Williams. The coolest way to generate balanced parentheses strings. *CATS 2008, Computing: The Australasian Theory Symposium, Wollongong, Australia*, 2008.

[18] Charles Semple and Mike Steel. *Phylogenetics*, volume 24 of Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford, 2003.

[19] E. Shroder. Vier combinatorische probleme. *Zeit. fur Math. Phys*, 15:361–376, 1870.

[20] Yun S. Song and Jotun Hein. *Parsimonious Reconstruction of Sequence Evolution and Haplotype Blocks*. Lecture Notes in Compute Science. Springer Berlin / Heidelberg, 2003.

[21] D.L. Swofford and G. J. Olsen. *Phylogeny reconstruction*, pages 411–501. Sinauer Associates, Sunderland, Mass., 1990.

[22] M. S. Waterman. On the similarity of dendrograms. *J. theor. Biol.*, 73:789–800, 1978.