frontiers in Genetics

ORIGINAL RESEARCH
published: 27 October 2020
doi: 10.3389/fgene.2020.584785

Combinatorial and Computational Investigations of Neighbor-Joining Bias

Ruth Davidson[1] *and Abraham Martín del Campo*[2]*

[1] Independent Researcher, Bellingham, WA, United States, [2] Centro de Investigación en Matemáticas, A.C., Guanajuato, Mexico

The Neighbor-Joining algorithm is a popular distance-based phylogenetic method that computes a tree metric from a dissimilarity map arising from biological data. Realizing dissimilarity maps as points in Euclidean space, the algorithm partitions the input space into polyhedral regions indexed by the combinatorial type of the trees returned. A full combinatorial description of these regions has not been found yet; different sequences of Neighbor-Joining agglomeration events can produce the same combinatorial tree, therefore associating multiple geometric regions to the same algorithmic output. We resolve this confusion by defining agglomeration orders on trees, leading to a bijection between distinct regions of the output space and weighted Motzkin paths. As a result, we give a formula for the number of polyhedral regions depending only on the number of taxa. We conclude with a computational comparison between these polyhedral regions, to unveil biases introduced in any implementation of the algorithm.

Keywords: neighbor joining, polyhedral cones, Motzkin paths, agglomeration, binary tree, distance-based methods

OPEN ACCESS

Edited by:
Ruriko Yoshida, Naval Postgraduate School, United States

Reviewed by:
Daniel Irving Bernstein, Massachusetts Institute of Technology, United States
Stefan Forcey, University of Akron, United States

***Correspondence:**
Abraham Martín del Campo
abraham.mc@cimat.mx

Specialty section:
This article was submitted to Evolutionary and Population Genetics, a section of the journal Frontiers in Genetics

Received: *18 July 2020*
Accepted: *16 September 2020*
Published: *27 October 2020*

Citation:
Davidson R and Martín del Campo A (2020) Combinatorial and Computational Investigations of Neighbor-Joining Bias. Front. Genet. 11:584785. doi: 10.3389/fgene.2020.584785

1. INTRODUCTION

The Neighbor-Joining (NJ) algorithm (Saitou and Nei, 1987) is a polynomial-time phylogenetic tree construction method. It is *agglomerative*, so it constructs ancestral relationships between taxa by clustering the most closely related taxa at each step until a complete phylogeny is formed. The performance of the NJ algorithm has been studied from multiple mathematical and biological perspectives (Bryant, 2005; Gascuel and Steel, 2006; Eickmeyer and Yoshida, 2008; Eickmeyer et al., 2008). Moreover, some statistical conditions have been given to guarantee a good performance of the algorithm for different types of biological data (Jiang and Lee, 1997; Mihaescu et al., 2009). Due to its speed and theoretical performance guarantees, NJ remains a popular tool in the design of phylogenetic pipelines for large and complex data sets (Liu and Yu, 2011; Lee et al., 2014; Telles et al., 2018).

The NJ algorithm takes as input a dissimilarity map D between n taxa and returns an unrooted binary tree with edge weights forming a tree metric. One established paradigm for evaluating the performance of the NJ algorithm is to consider it as a heuristic for either the Least Squares Phylogeny (LSP) problem, which is NP-complete (Day, 1987), or the Balanced Minimum Evolution (BME) problem, which is a special case of a weighted LSP approach (Desper and Gascuel, 2004). The LSP problem asks for the tree metric T that minimizes the function

$$\sqrt{\sum_{i,j\in\binom{[n]}{2}} (D_{i,j} - T_{i,j})^2}.$$

This perspective gives rise to the problem of determining whether a given heuristic for LSP or BME is biased, favoring certain phylogenies over others in its output. Addressing this problem requires a clear accounting of the possible outputs of the NJ algorithm.

Dissimilarity maps are symmetric matrices that can be realized as points in a Euclidean space. In this geometric setting, the space of all tree metrics form a *polyhedral fan* (Eickmeyer and Yoshida, 2008), which is a union of polyhedral cones meeting along common faces. The geometry of the space of tree metrics is well-understood (Semple and Steel, 2003; Speyer and Sturmfels, 2003). Phylogenetic inference methods that take a dissimilarity map as an input divide the Euclidean space into a family of subsets indexed by the combinatorial type of the trees returned. For the NJ algorithm, these subsets are also polyhedral regions, as its decision criteria are linear inequalities on the input data.

Comparing these polyhedral cones had lead to new computational methods to evaluate the performance of the NJ algorithm as a heuristic for LSP in Davidson and Sullivant (2014) and of NJ as a heuristic for BME in Eickmeyer et al. (2008). These methods are based on measuring the spherical volume of the cones, which is the volume of their intersection with the unit sphere. The spherical volume of polyhedral cones had unveiled unexpected bias in the UPGMA method, an older agglomerative phylogenetic method that has poorer performance guarantees in comparison to NJ (Davidson and Sullivant, 2013).

The contribution of this article is that we give a formula that enumerates the polyhedral cones in the partition returned by NJ algorithm, by giving a bijection between the cones and weighted Motzkin paths. We also explore the inherent bias in the algorithm by computing the spherical fraction of the cones and noting significant variation between some cones, similar to those found in Davidson and Sullivant (2013) for the UPGMA algorithm.

In section 2 we give a in-depth description of NJ and provide related definitions necessary to describe distinct NJ outputs. In section 3 we give a bijection between weighted Motzkin paths and labeled Newick strings to describe the outputs combinatorially. Lastly, in section 4 we estimate the spherical fraction of the NJ cones and unveil a bias. We propose two ways to correct the bias, and display the performance of these corrections via computational experiments for small numbers of taxa.

2. THE NEIGHBOR-JOINING ALGORITHM

A *graph* is a pair of sets $\{V, E\}$ called vertices and edges, with the latter representing relations between pairs of vertices. The *degree of a vertex* is the number of edges adjacent to it. A *path* in a graph is a sequence of edges joining a sequence of vertices without repetition of vertices, except for possibly the start and end vertices. If the sequence starts and ends at the same vertex, the path is called a *cycle*. A graph without cycles is a *tree*. The vertices of a tree are usually called *nodes*, except for those of degree one which are called *leaves*. In trees, there is a unique path between any two vertices. A tree is *rooted* if there is a distinguished vertex ρ called the *root*. If T is a rooted tree, there is a partial order on the vertices of T induced by the root ρ and the unique path between vertices, such that $\rho \leq v$ for all vertices v of T. A *binary tree* is a tree where all vertices have degree three except for the leaves and the root, if a root exists. We will restrict our attention to unrooted binary trees. An unrooted binary tree with n leaves has $n-2$ nodes and $2n-3$ edges. We also consider the *star tree*, a non-binary unrooted tree with exactly one node that is adjacent to all leaves. We illustrate a star tree in the left image of **Figure 1**.

2.1. Description of the Algorithm

Let $[n] := \{1, \ldots, n\}$. A *dissimilarity map* is a function $D : [n] \times [n] \to \mathbb{R}_{\geq 0}$ such that $D(a, b) = D(b, a)$, $D(a, a) = 0$, and $D(a, b) \geq 0$ for all $a, b \in [n]$. A dissimilarity map is *additive* or a *tree metric*, if there exists a tree T with edges E with a weight function $w : E \to \mathbb{R}_{\geq 0}$ on T so that $D(a, b)$ equals the sum of the edge weights on the unique path from a to b in T.

The NJ algorithm takes a dissimilarity map as input and returns an unrooted binary tree with a tree metric. It is useful to view the progress of the NJ algorithm as a series of graph transformations. The algorithm starts from a star tree t_n, a graph with n leaves and only one node $\mathcal{O}$ adjacent to all leaves, as illustrated on the left of **Figure 1**. Throughout the algorithm, the node $\mathcal{O}$ plays an important role; thus, we give special names to the vertices adjacent to $\mathcal{O}$.

Definition 2.1. Vertices adjacent to $\mathcal{O}$ are called *boughs*. Vertices that are boughs can be either leaves or internal nodes, so we refer to them as *stems* and *bouquets,* respectively.

In the recursive step, the algorithm takes a tree t_k with k boughs, it selects a pair of them and joins them by adding a new node adjacent to $\mathcal{O}$ in a way that the resulting tree t_{k-1} has $k-1$ boughs. The algorithm iterates this step until there are only three boughs. **Figure 1** illustrates the first two steps in the NJ algorithm, starting from the star t_7 and the next graph constructed by the NJ algorithm corresponding to the tree t_6, obtained from the star by joining the stems a, b by introducing a bouquet u. The subgraph consisting of only two adjacent leaves is sometimes called a *cherry*, and the recursive step of the NJ algorithm is then referred as *cherry picking* step (Eickmeyer and Yoshida, 2008). This term served as inspiration for the names in Definition 2.1, as it resembles the way the NJ algorithm creates trees.

The algorithm selects a pair of boughs a, b to agglomerate based on the Q-criterion, which is a function given by

$$Q(a, b) = (k-2)D(a, b) - \sum_{c \in X} D(a, c) - \sum_{c \in X} D(b, c), \qquad (2.1)$$

where X is the set of boughs, with $k = |X|$. The NJ algorithm selects the vertices a, b where Q is minimal. It has been shown that this criterion uniquely determines the NJ algorithm (Bryant, 2005), and that a pair of leaves minimizing Q come from a cherry in the true tree (Saitou and Nei, 1987; Studier and Keppler, 1988). To agglomerate the selected nodes a, b, the NJ algorithm introduces a new bouquet u adjacent to them, resulting in a tree with one bough less. It then estimates the distance in this new tree from the remaining vertices to u via the reduction formula

$$D(c, u) = \frac{1}{2}\left(D(a, c) + D(b, c) - D(a, b)\right). \qquad (2.2)$$

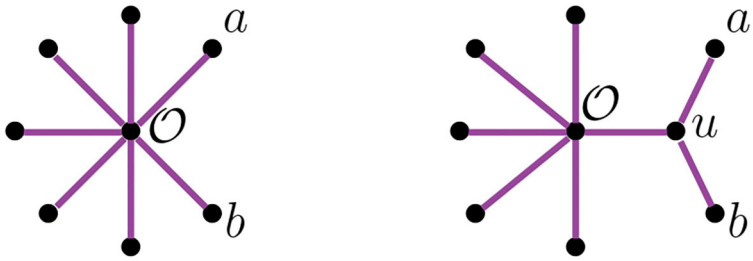

FIGURE 1 | The trees corresponding to the first and second step in the NJ algorithm.

We sometimes use D_k and Q_k to denote the dissimilarity map and the Q-criterion associated to the tree t_k with k boughs. Thus, the last step of the algorithm is decided by a pair of nodes a, b that minimize Q_4. However, in this last step there is more than one minimizing pair, as we demonstrate in the following lemma.

Lemma 2.2. For $n = 4$, the matrix Q_4 always achieves its minimum in exactly two entries.

Proof: Let $\{a, b, c, d\}$ be the four vertices corresponding to t_4 and suppose, without loss of generality, that the minimum in Q_4 is achieved in $Q(a, b)$. Then, Equation (2.1) writes as

$$\begin{aligned} Q(a,b) &= 2D(a,b) - (D(a,b) + D(a,c) + D(a,d)) - (D(a,b) \\ &\quad + D(b,c) + D(b,d)) \\ &= -(D(a,c) + D(a,d)) - (D(b,c) + D(b,d))) \\ &= -(D(a,c) + D(b,c)) - (D(a,d) + D(b,d)) \\ &= -(D(a,c) + D(b,c) + D(c,d)) - (D(a,d) + D(b,d) \\ &\quad + D(c,d)) + 2D(c,d) = Q(c,d). \end{aligned}$$

Therefore, we get that $Q(a, b) = Q(c, d)$, meaning that the minimum is achieved twice.

Remark 2.3. The previous lemma is similar to the four-point condition but not the same, as Lemma 2.2 holds even when the dissimilarity map D is not additive.

Remark 2.4. Note that if $n > 4$, the minimum in Q_n could be achieved in more than one entry too, but that happens in a very small dimensional space in $\mathbb{R}^{\binom{n}{2}}$, thus it occurs in a measure zero set. Only in the case $n = 4$ it happens always.

2.2. Newick Notation

We represent trees with the Newick notation. This is one of the most widely used notations in bioinformatics to encode information about the tree topology, branch distances, and vertex labels. It consist of parentheses that represent tree data as textual strings (see Felsenstein, 2004; Warnow, 2017). A pair of vertices enclosed within matching parentheses indicates they have a common ancestor. There are slightly different formats representing the Newick notation, so we explain the one we use with an example.

Let t be the tree in **Figure 2**, with leaves labeled by the set $\{a, b, c, d, e, f\}$. Newick notation for t can be written as $((c, (a, b)), (e, f), d)$, or $((d, (c, (a, b)))), e, f)$, or in many other ways. This non-uniqueness of Newick strings can be inconvenient, as one must determine whether two strings represent the same tree or not. We let the *length* of a string in Newick notation to be the maximum number of parentheses of one orientation contained in the string (to the right or the left). For instance, the two strings above have length 3.

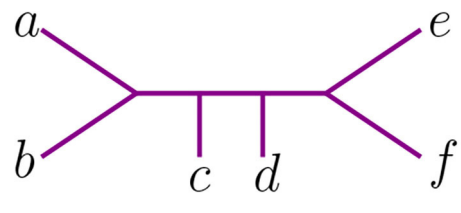

FIGURE 2 | An unrooted binary tree t with 6 taxa.

We now explain the Newick notation in the NJ algorithm. Let $[n]$ indicate the leaves of a tree. The starting point is the start tree t_n, which is indicated with the string $(1, 2, \ldots, n)$. Then, the algorithm selects two vertices and we join them by enclosing them with a parenthesis, which we append at the left to the remaining string. For instance, if 1 and 2 are the selected vertices, we would write $((1, 2), 3, \ldots, n)$ to indicate the tree obtained in the second step in the NJ algorithm. This is illustrated in the tree to the right of **Figure 1**, by letting $a = 1$ and $b = 2$. In the recursive step, NJ will take a string s_k of length k and write a string of length $k-1$ by joining two elements of s_k with a new parenthesis attached to the left. In this way, in the string $((c, (a, b)), (e, f), d)$ we know that the last step was to join c with the pair (a, b), but we do not know if the previous string was $((a, b), (e, f), c, d)$ or $((e, f), (a, b), c, d)$. A parenthesis in the Newick string indicates a node in the tree, so we could remove this ambiguity by labeling the nodes when we introduce them in the algorithm. We label these nodes with a circled number written to the left of a parenthesis, indicating the step when the node was created in the algorithm. For instance, we could write $(③(c, ①(a, b)), ②(e, f), d)$ to indicate that the first step in the NJ algorithm was to join (a, b), then (e, f), and lastly c with (a, b). We refer hereafter to this notation as *ordered Newick notation*.

Algorithm 1 | The Neighbor-Joining Algorithm.

Input : A dissimilarity map D_n on the set $\{1,\dots,n\}$.
Output: An unrooted binary tree T with leaf labels $\{1,\dots,n\}$ written in ordered Newick notation.
Initialize $k=n$, $r=0$, and $S_k=(1,\dots,k)$ representing the Newick format for t_k, the star tree on k leaves with boughs B_k.
while $k>3$ **do**

1. Identify boughs $\{a,b\}$ of t_k minimizing

$$Q_k(a,b)=(k-2)D_k(a,b)-\sum_{c\in B_k}D_k(a,c)-\sum_{c\in B_k}D_k(b,c).$$

2. Define S_{k-1} by appending $\textcircled{r}(a,b)$ to the left of S_k after dropping a and b.
3. Construct t_{k-1} from t_k by
   a. Deleting the edges from both a and b to $\mathcal{O}$.
   b. Introducing a new vertex labeled u adjacent to $\mathcal{O}$.
   c. Connecting u to both a and b.
4. Compute D_{k-1} on the new set of boughs via the formula

$$D_{k-1}(u,v)=\frac{1}{2}\left(D_k(a,v)+D_k(b,v)-D_k(A,B)\right)$$

   for all remaining boughs in t_k
5. Increase r by one and decrease k by 1.

return Label $\mathcal{O}$ in t_3 with zero and return $t_3=T$
end

3. COMBINATORIAL DESCRIPTION OF NJ

To a given data matrix D, the NJ algorithm associates a binary tree with n leaves, together with a tree metric. Without the distinction of ordered Newick notation, it can appear that the algorithm associates the same tree to different data. For instance, let us consider the following matrices:

$$D=\begin{pmatrix}0&3&5&4&7\\3&0&10&3&7\\5&10&0&6&5\\4&3&6&0&2\\7&7&5&2&0\end{pmatrix},\qquad D'=\begin{pmatrix}0&2&4&1&9\\2&0&10&3&8\\4&10&0&6&5\\1&3&6&0&7\\9&8&5&7&0\end{pmatrix}.\tag{3.1}$$

The NJ algorithm associates the same unrooted binary tree to both of them. However, their Newick notation is not the same. For D, the corresponding Newick tree built by NJ is $((d,(a,b)),c,e)$ whereas for D' we obtain $((d,(c,e)),a,b)$. We illustrate this in **Figure 3**.

These two Newick strings encode one topological type of tree. Yet they have a subtle difference if one needs to take into account the order in which taxa was agglomerated to interpret the output of the NJ algorithm, as one must do when establishing which

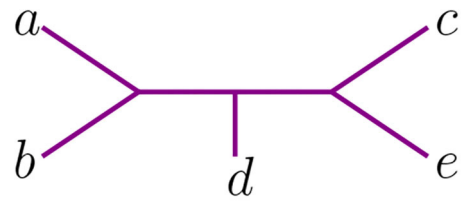

FIGURE 3 | Tree corresponding to the Newick strings $((d,(a,b)),c,e)$ and $((d,(c,e)),a,b)$.

polyhedral region of $\mathbb{R}^{\binom{n}{2}}$ contains the dissimilarity matrix input that produces a given Newick string under NJ. See section 4 for a discussion of the implications of locating the correspondence between these Newick strings and the geometry of NJ. For example, the Newick string $((d,(a,b)),c,e)$ corresponding to D indicates that a,b were joined together and that c and e were never joined to anything, whereas the Newick $((d,(c,e)),a,b)$ for D' indicates the opposite, that a,b were never joined to anything but c and e were joined to each other. To distinguish these data and relate them to ordered Newick notation, we endow binary trees with something we called *agglomeration order* and we explain it next.

3.1. Binary Trees With Agglomeration Order

The expected number of trees that arise as an output depends on the shape (topology) and the labels of the leaves. For unrooted trees with n taxa, there are $(2(n-2)-1)!!$ labeled binary trees. Thus, for $n=4$ there is only one tree shape and three ways to label the leaves. However, there are two possible ways to write them in Newick notation. For instance, we could have $((a,b),c,d)$ or $((c,d),a,b)$ to mean the same tree, but the NJ algorithm differentiates these two. Therefore, for $n=4$ there are six possible Newick strings but the NJ algorithm only returns three of them. We summarize this information for the first five cases in the following table.

We will give a combinatorial formula to compute the expected number of output trees from the algorithm. For this, we set the combinatorial definitions we will require.

Definition 3.1. For a binary tree with n leaves, an *agglomeration order* means labeling the $n-2$ internal nodes with the set $\{\infty,1,2,\dots,n-3\}$, such that the labels of the internal nodes in every path from ∞ to a leaf form a decreasing sequence.

We use circled numbers to indicate the assignment of the agglomeration order, and to simplify the schematics, we omit the label ∞, so it is easier to verify that all sequences of internal vertices starting there and terminating at a leaf are decreasing. In **Figure 4** we illustrate an agglomeration order for the tree t with 6 taxa from **Figure 2**. There, if we exchange ③ with ① we get a label of the internal nodes that is not an agglomeration order. In Newick notation, the tree in **Figure 4** would be $(③(c,①(a,b)),②(e,f),d)$.

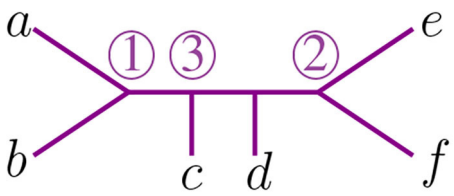

FIGURE 4 | An agglomeration order in an unrooted tree with 6 taxa.

TABLE 1 | Number of trees for small number of taxa.

Taxa	Tree topologies	Unrooted binary trees	Trees from NJ	Ordered newick strings
4	1	$3 = 3!!$	3	6
5	1	$15 = 5!!$	30	60
6	2	$105 = 7!!$	450	900
7	2	$945 = 9!!$	9,450	18,900
8	4	$10,395 = 11!!$	264,600	529,200

Remark 3.2. From Lemma 2.2, we see that the NJ algorithm associates two ordered trees to a single data matrix D, as it has to decide between two agglomeration orders in the last step.

Theorem 3.3. *The set of agglomeration orders on unrooted binary trees is in a 2-to-1 correspondence with the output space of the NJ algorithm.*

Proof: The NJ algorithm starts with the star tree t_n consisting of n leaves and just one internal node $\mathcal{O}$. From there, at each step, the algorithm applies a series of graph transformations to a given tree, preserving the number of leaves, while increasing the number of nodes by one. This new node is adjacent to $\mathcal{O}$. Thus, at each step the new node is closer (combinatorially, not metrically) to $\mathcal{O}$ than all other nodes in a path to a leaf. Numbering each node with the moment they appear in the algorithm, starting with ∞ for the node $\mathcal{O}$, results in an agglomeration order. As these steps are reversible, each agglomeration order can arise from the algorithm, giving the 2-to-1 correspondence with the output space, together with Remark 3.2.

Therefore, understanding the NJ algorithm leads to understanding orders of agglomeration for binary trees. To simplify the rest of the exposition, we give the following definitions.

Definition 3.4. We use the term *agglomerated tree* to refer to an unrooted binary tree endowed with an agglomeration order. The number of agglomerated trees with n leaves will be denoted by $\Phi(n)$.

Computing the number $\Phi(n)$ is important to understand more about the combinatorial complexity of the NJ algorithm. Due to Theorem 3.3 above, the number of trees output by the NJ algorithm is $\Phi(n)/2$. The last column of **Table 1** is precisely the value of $\Phi(n)$ for $n = 4, \ldots, 8$. We tried to determine

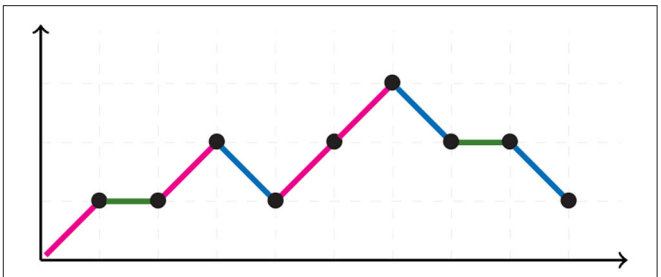

FIGURE 5 | Motzkin path.

$\Phi(n)$ by estimating first the number of agglomeration orders, knowing that the number of unrooted binary trees is given by the combinatorial formula $(2n-5)!!$. For $n = 4$ and 5, there is only one tree topology, but the number of agglomeration orders they have is 2 and 4, respectively. For these cases, it holds true that $\Phi(4) = 2 \cdot 3!!$ and $\Phi(5) = 4 \cdot 5!!$. However, as we can see in **Table 1**, this is no longer the case for other values of n, as $\Phi(n)$ is not always divisible by $(2n-5)!!$. Nonetheless, we were able to give a formula for the number $\Phi(n)$ using Motzkin paths. However, it remains open to understand more about the connection between unrooted binary trees and agglomeration orders.

Problem 3.5. *Determine the number of agglomeration orders that can be assigned to a given unrooted binary tree.*

3.2. Motzkin Paths

Motzkin paths are combinatorial structures appearing in many contexts. They are counted by Motzkin numbers, which are related to Catalan numbers (Aigner, 1998, 1999; Meshkov et al., 2010; Oste and Van der Jeugt, 2015). Note that while Catalan numbers are known to count combinatorial objects referred to as *planar rooted trees* (Deutsch and Shapiro, 2002), the trees in this paper are fundamentally different objects.

A *Motzkin path* is an integer lattice path starting and ending in the horizontal axis without crossing it, consisting of up steps $u = (1, 1)$, down steps $d = (1, -1)$, and horizontal steps $h = (1, 0)$. **Figure 5** illustrates a Motzkin path. A Motzkin path with no horizontal steps is a *Dyck path*. The number of Dyck paths from $(0, 0)$ to $(2N, 0)$ is given by the Catalan number C_N, whereas the number of Motzkin paths from $(0, 0)$ to $(0, N)$, is given by the Motzkin number M_N.

Our main result here is to give a bijection between Motzkin paths and agglomerated trees. This bijection allows the derivation of a formula to determine the number $\Phi(n)$ that counts the number of agglomerated trees. The key is to focus on the recursive step of the algorithm.

Remark 3.6. Let $\mathcal{O}$ be the unique node of the star tree t_n. In the recursive step, the NJ algorithm takes a tree t_k with ℓ stems and r bouquets, such that $k = \ell + r$. From there, it constructs the graph t_{k-1} by adding an internal node in three possible ways:

- It merges two stems. We call this step α.
- It merges two bouquets. We call this step β.
- It merges a stem to a bouquet. We call this step γ.

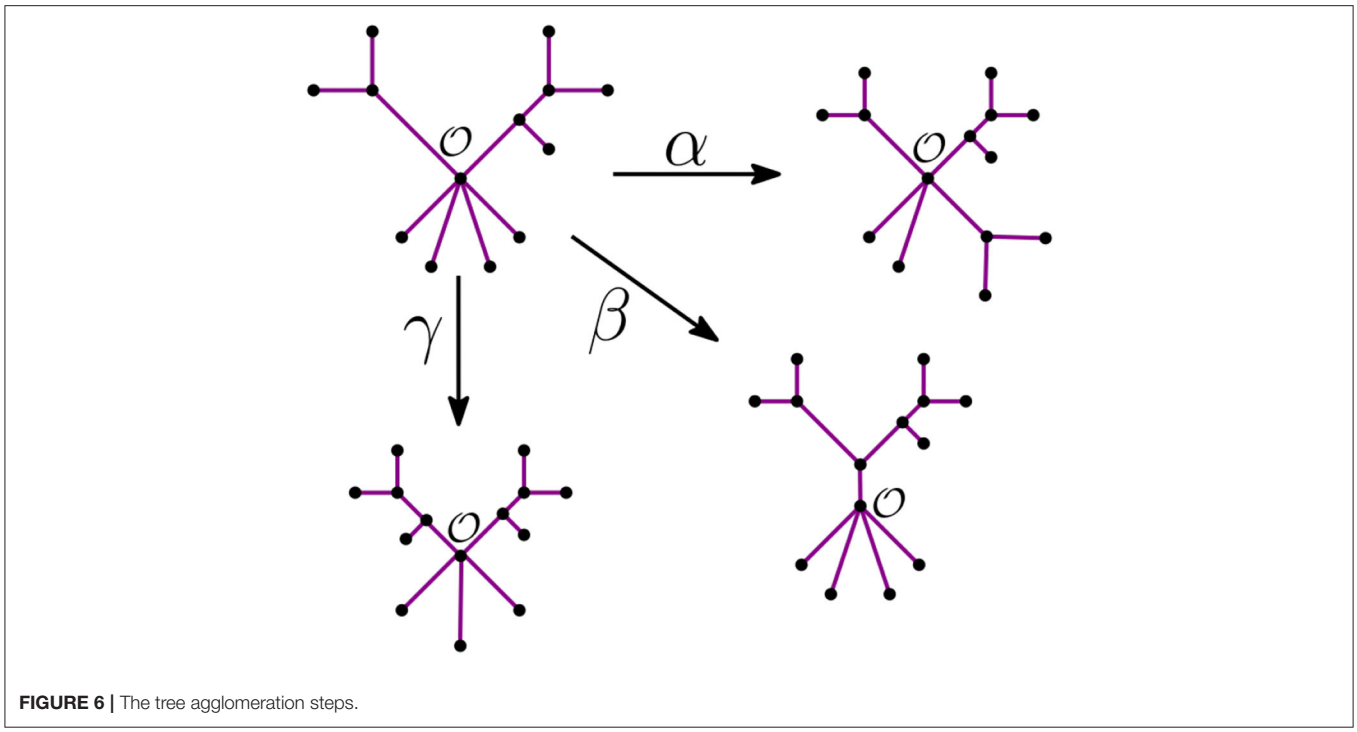

FIGURE 6 | The tree agglomeration steps.

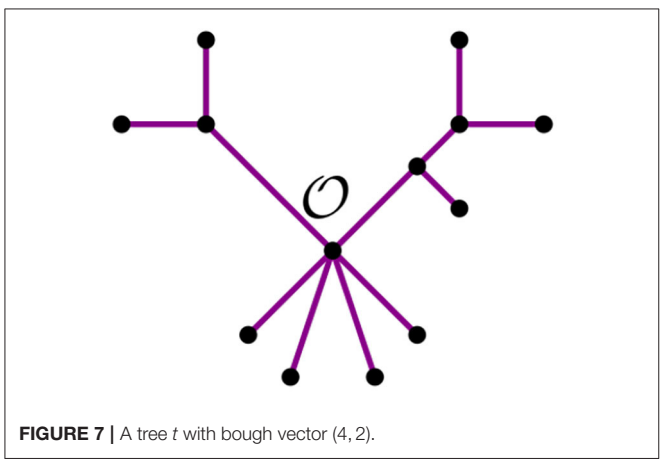

FIGURE 7 | A tree t with bough vector $(4, 2)$.

We illustrate these three steps in **Figure 6** below.

From this remark, we see that we can summarize a given tree t with a distinguished node $\mathcal{O}$ by its *bough vector* (ℓ, r), where ℓ and r are the number of stems and bouquets in t, respectively. For instance, the tree t in **Figure 7** has three nodes and nine leaves, but just four stems and two bouquets. Thus, the bough vector $(4, 2)$ summarizes t. The bough vector for the star with n leaves is $(n, 0)$. Let t_k be a tree from the NJ algorithm, and let (ℓ, r) be its bough vector, so $k = \ell + r$. Note that the tree obtained from t_k after an α step is summarized by the bough vector $(l, r) + (-2, 1)$. Similarly, after a step β or γ, the bough vector of the tree is $(l, r) + (0, -1)$, or $(l, r) + (-1, 0)$, respectively. Note that the NJ algorithm ends with a tree T_3 consisting only of three boughs. Thus, for $n \geq 4$, the possible bough vectors for the last step are $(2, 1), (1, 2)$, or $(0, 3)$. Note that the first step of the algorithm is forced to be always an α step, thus we can omit it and analyze the rest of the steps, starting at $(n-2, 1)$.

Definition 3.7. For $n \geq 4$ we define an *NJ path* of length $n-4$ as an integer lattice path consisting of steps $\alpha = (-2, 1)$, $\beta = (0, -1)$, and $\gamma = (-1, 0)$, starting at $(n-2, 1)$ and ending at one of $(2, 1), (1, 2)$, or $(0, 3)$, without crossing the horizontal axis $y = 1$.

Theorem 3.8. For every $n \geq 4$, there is a bijection between NJ paths of length $n-4$ and Motzkin paths of length $n-4$ from $(1, 1)$ to either $(n-3, 1), (n-3, 2)$, or $(n-3, 3)$.

Proof: The matrix $\left(\begin{smallmatrix} -1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$ sends steps $\{\alpha, \beta, \gamma\}$ into $\{u, d, h\}$, respectively, giving the bijection.

Motzkin paths starting at the origin and ending at (ℓ, r) are called *partial Motzkin paths*. Thus, after translating $(1, 1) \rightarrow (0, 0)$, we could write Theorem 3.8 in terms of partial Motzkin paths ending at $(n-4, 0), (n-4, 1)$, or $(n-4, 2)$.

For 6 taxa, all possible NJ paths and their corresponding Motzkin paths are depicted in **Figure 8**. NJ paths there start at $(6, 0)$ corresponding to the start tree t_6, and the first step is always an α step. From there, one chooses from the three steps $\{\alpha, \beta, \gamma\}$ consecutively until reaching one of the points $(0, 3), (1, 2)$, or $(2, 1)$. In this case, there are only five paths, each corresponding to an agglomeration order in **Figure 9**. For instance, the path formed by the sequence $\alpha\alpha\gamma$ (read from left to right) is in correspondence to the agglomeration order of the tree in **Figure 4**, denoted in Newick

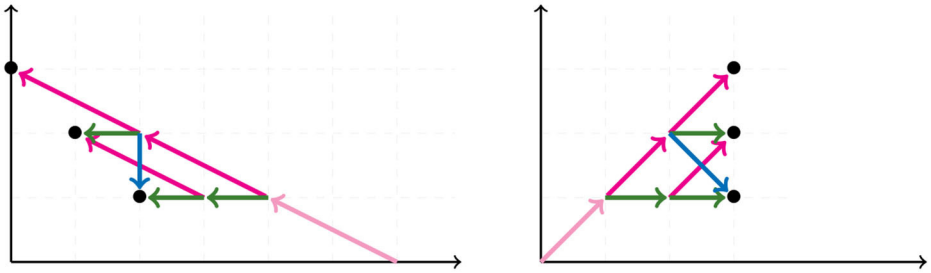

FIGURE 8 | All NJ paths **(Left)** and partial Motzkin paths **(Right)** for 6 taxa.

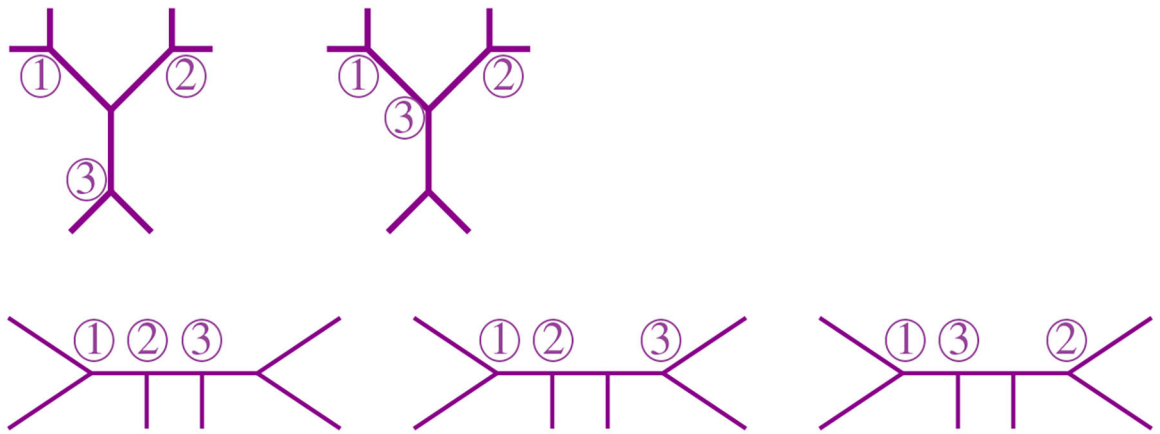

FIGURE 9 | Some agglomeration orders corresponding to the five Motzkin paths in **Figure 8**.

format by $(③(c, ①(a, b)), ②(e, f), d)$. Note that the sequence $\alpha\alpha\gamma$ is in correspondence with more than one agglomeration order. For instance, it is also in correspondence with the tree $(③(c, ①(e, f)), ②(a, b), d)$.

3.3. The Number of Agglomerated Trees

The results of the previous discussion reduce the counting of the number of trees obtained from the NJ algorithm to enumerating some partial Motzkin paths, which are counted by Motzkin numbers. Hence, we let M_k be the number of partial Motzkin paths of length k. More specifically, we let $M_{k,j}$ be the number of partial Motzkin paths on length k that end at level j. It is known (see Oste and Van der Jeugt, 2015) that the Motzkin numbers M_k satisfy the following formulas involving Catalan numbers:

$$M_k = \sum_i \binom{k}{2i} C_i, \qquad \text{and} \qquad C_{k+1} = \sum_i \binom{k}{i} M_i. \tag{3.2}$$

While the numbers $M_{k,j}$ satisfy the following formula (Bóna, 2015, Theorem 10.8.1)

$$M_{k,j} = \sum_{i=0}^{k} \binom{k}{i} \left[\binom{k-i}{(k+j-i)/2} - \binom{k-i}{(k+j-i+2)/2} \right], \tag{3.3}$$

where, by convention, a binomial coefficient is 0 if its bottom parameter is not an integer. These numbers $M_{k,j}$ construct the *Motzkin triangle* that enumerates all partial Motzkin paths (Lando, 2003), which is shown in **Figure 10**. The triangle is defined recursively by

$$M_{0,0} = 1, \qquad M_{k+1,j} = M_{k,j-1} + M_{k,j} + M_{k,j+1}, \text{ for } k \geq 1. \tag{3.4}$$

The numbers in the triangle form the sequence A026300 in Sloane (2020), and the first and second rows (from bottom to top) are the sequences A001006, A002026, respectively. Notice in **Figure 8** that for all NJ paths, there is only one way to reach the point $(0, 2)$ from the ending points $(0, 3), (1, 2)$, and $(2, 1)$. This holds in general due to the recursion (3.4). Thus, counting the number of NJ paths that end in one of these three points is equivalent to counting all NJ paths ending at $(0, 2)$, or equivalently, all partial Motzkin paths ending at $(n-3, 1)$. In this way, we conclude that the number NJ paths is given by the Motzkin number $M_{n-3,1}$, and Equation (3.3) gives a formula for them. We summarize these observations in the following theorem.

Theorem 3.9. *The number of NJ paths for n taxa equals the Motzkin number $M_{n-3,1}$. Thus, it can be written as*

$$\sum_{i=0}^{n-3} \binom{n-3}{i} \left[\binom{n-i-3}{(n-i-2)/2} - \binom{n-i-3}{(n-i)/2} \right],$$

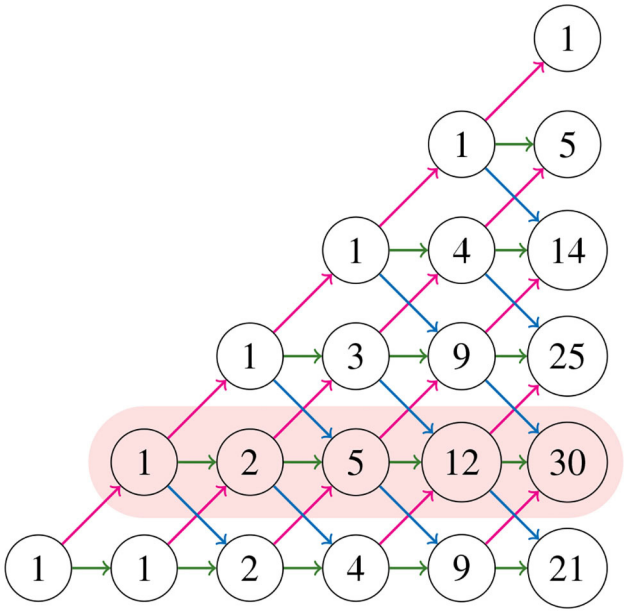

FIGURE 10 | Motzkin triangle with the second row highlighted.

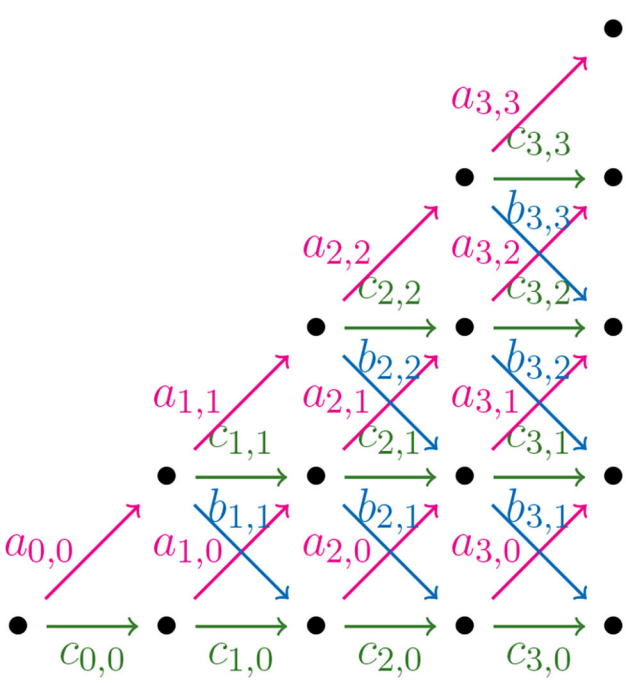

FIGURE 11 | Motzkin triangle with multiplicities.

where, by convention, a binomial coefficient is 0 if its bottom parameter is not an integer, or if it is larger than the top parameter.

In order to find a formula for $\Phi(n)$, the output size of the NJ algorithm, we consider *weighted partial Motzkin paths* which are partial Motzkin paths with weight assignments of non-negative numbers $\{a_{k,j},\ b_{k,j},\ c_{k,j}\}$ to the steps $\{u, d, h\}$. We interpret the weights as the multiplicity of the arrow, or equivalently, as the number of arrows in the given direction. In **Figure 11**, we show the weight assignment to the arrows of the Motzkin triangle. Triangles of this kind are called *Motzkin triangles with multiplicities* (Lando, 2003). They generalize the Motzkin triangle from **Figure 10**, as this is the case $a_{k,j} = b_{k,j} = c_{k,j} = 1$ for all $k, j \geq 0$. The recursion in Equation (3.4) generalizes to the following recursion for Motzkin triangles with multiplicities

$$M_{k+1,j} = a_{k,j-1}M_{k,j-1} + c_{k,j}M_{k,j} + b_{k,j+1}M_{k,j+1}, \text{ for } k, j \geq 0. \tag{3.5}$$

Let t_k be a tree in the NJ algorithm with bough vector (ℓ, r), such that $k = \ell + r$. If the next step in the NJ algorithm is a step α, one needs to choose two of the ℓ stems to join. There are $\binom{\ell}{2}$ ways to do this. In a similar way, there are $\binom{r}{2}$ choices for a β step, and $\binom{\ell}{1}\binom{r}{1}$ for a γ step. We can use these as weights for computing the number of agglomerated trees after each step. Thus, translating NJ paths to partial Motzkin paths starting at (1,1) by Theorem 3.8, and letting $s = n-1-k$, we need to assign weights in the following way:

- $a_{s,j} = \binom{n-s-j-2}{2}$,
- $b_{s,j} = \binom{j+1}{2}$,
- $c_{s,j} = \binom{n-s-j-2}{1}\binom{j+1}{1}$,

for $0 \leq s \leq n-4$ and $0 \leq j \leq s$, or zero otherwise.

Lemma 3.10. For $n \geq 4$ and $s \leq n-3$, the weights $a_{s,j}, b_{s,j}, c_{s,j}$ satisfy

$$a_{s,j} + b_{s,j} + c_{s,j} = \binom{n-s-1}{2}. \tag{3.6}$$

Proof: From definition, Equation (3.6) can be written as

$$\begin{aligned}
&\binom{n-s-j-2}{2} + \binom{j+1}{2} + \binom{n-s-j-2}{1}\binom{j+1}{1} = \\
&= \frac{(n-s-j-2)(n-s-j-3)}{2} + \frac{(j+1)j}{2} + \frac{2(n-s-j-2)j}{2} \\
&= \frac{(n-s-j-2)^2 - (n-s-j-2)}{2} + \frac{(j+1)^2 - (j+1)}{2} \\
&+ \frac{2(n-s-j-2)j}{2} \\
&= \frac{(n-s-j-2)^2 + 2j(n-s-j-2) + (j+1)^2}{2} \\
&- \frac{(n-s-j-2) + (j+1)}{2} \\
&= \frac{\big((n-s-j-2) + (j+1)\big)^2}{2} - \frac{(n-s-j-2) + (j+1)}{2} \\
&= \frac{(n-s-1)(n-s-2)}{2} \quad = \quad \binom{n-s-1}{2}.
\end{aligned}$$

Lemma 3.11. Let $n \geq 4$, and for $2 \leq s \leq n-2$, we have

$$\sum_{j=0}^{s} M_{n-s-2,j} = \binom{s+2}{2} \sum_{j=0}^{s'} M_{n-s-3,j}, \tag{3.7}$$

with the sum in the right-hand side ending at

$$s' = \begin{cases} s+1 & \text{if} \quad n-s-2 > \lfloor \frac{n-2}{2} \rfloor + 1, \\ s+1 & \text{if} \quad n-s-2 = \lfloor \frac{n-2}{2} \rfloor + 1, \text{and } n \text{ is even}, \\ s & \text{if} \quad n-s-2 = \lfloor \frac{n-2}{2} \rfloor + 1, \text{and } n \text{ is odd}, \\ s-1 & \text{if} \quad n-s-2 \leq \lfloor \frac{n-2}{2} \rfloor. \end{cases}$$

Proof: The recursion (3.5) for weighted Motzkin paths writes the left-hand side of (3.7) as

$$\begin{aligned} \sum_{j=0}^{s} M_{n-s-2,j} = &\sum_{j=0}^{s} \big(a_{n-s-3,j-1}M_{n-s-3,j-1} + c_{n-s-3,j}M_{n-s-3,j} \\ &+ b_{n-s-3,j+1}M_{n-s-3,j+1}\big) \\ =&(a_{n-s-3,0} + c_{n-s-3,0})M_{n-s-3,0} \\ &+ \sum_{j=1}^{s-1}(a_{n-s-3,j} \\ &+ c_{n-s-3,j} + b_{n-s-3,j})M_{n-s-3,j} \\ &+ (b_{n-s-3,s} + c_{n-s-3,s})M_{n-s-3,s} \\ &+ b_{n-s-3,s+1}M_{n-s-3,s+1}. \end{aligned}$$

Notice that, by Lemma 3.10, we have $a_{n-s-3,j} + c_{n-s-3,j} + b_{n-s-3,j} = \binom{s+2}{2}$, and by definition, $b_{n-s-3,s+1} = \binom{s+2}{2}$. Lastly, notice that $n - (n-s-3) - s - 2 = 1$, so

$$\begin{aligned} (a_{n-s-3,0} + c_{n-s-3,0}) &= (b_{n-s-3,s} + c_{n-s-3,s}) \\ &= \binom{s+1}{2} \\ &+ \binom{1}{1}\binom{s+1}{1} = \binom{s+2}{2}. \end{aligned}$$

Hence, the sum in the right-hand side equals

$$\binom{s+2}{2} \sum_{j=0}^{s+1} M_{n-s-3,j}.$$

We conclude the proof by noting that, when using the recursion (3.5) to write a sum of Motzkin paths of length $i+1$ in terms of those of length i, the maximum amount of summands that can appear in the sum is attained when $i = \lfloor \frac{n-2}{2} \rfloor$.

Hence, we obtain the following formula for the number $\Phi(n)$.

Theorem 3.12. *For $n \geq 4$, let $\Phi(n)$ be the number of labeled agglomerated trees with n leaves. Then,*

$$\Phi(n) = \binom{n}{2}\binom{n-1}{2}\cdots\binom{5}{2}\binom{4}{2} = \frac{n(n-1)!^2}{3 \cdot 2^{n-1}}.$$

Proof: From Theorem 3.8, computing $\Phi(n)$ is equivalent to computing $M_{n-4,0} + M_{n-4,1} + M_{n-4,2}$ in the weighted version of the partial Motzkin paths. From Lemma 3.11 we obtain

$$M_{n-4,0} + M_{n-4,1} + M_{n-4,2} = \binom{4}{2} \sum_{j=0}^{3} M_{n-5,j}.$$

We can use Lemma 3.11 again to compute the sum in the right-hand side. Continuing recursively we see that

$$M_{n-4,0} + M_{n-4,1} + M_{n-4,2} = \binom{4}{2} \cdots \binom{n-1}{2} M_{0,0}.$$

Defining $M_{0,0} = \binom{n}{2}$ we obtain the result. The last equality in the theorem comes from writing $\binom{n}{2}$ as $\frac{n(n-1)}{2}$ and expanding the product of binomial coefficients.

We conclude this section by noticing that this formula for $\Phi(n)$ produces the sequence of the last column of **Table 1**; thus, the number of trees output by the NJ algorithm is $\Phi(n)/2$. Moreover, the sequence formed by our formula in Theorem 3.12 does not appear in Sloane (2020); however, it can be obtained from the product of consecutive binomial coefficients. This product forms the sequence A006472, which counts the number of ranked trees. Thus, our sequence is obtained from A006472 starting from $n = 4$ by dividing each element by 3. This suggest a connection between agglomerated trees and ranked trees (Disanto and Wiehe, 2013).

4. ESTIMATED VOLUMES

The Neighbor-Joining and UPGMA algorithms take dissimilarity maps as inputs and return trees with an additive dissimilarity map. The decisions in both algorithms are based on linear inequalities that divide $\mathbb{R}^{\binom{n}{2}}$ into half-spaces. To elucidate, label the coordinates of $\mathbb{R}^{\binom{n}{2}}$ with the 2-element subsets of $[n] = \{1, 2, \ldots, n\}$, so that the symmetric matrix D_n is a point in $\mathbb{R}^{\binom{n}{2}}$. Recall, for instance, the example in the beginning of section 3. The two matrices D and D' from (3.1) are both in correspondence with the tree in **Figure 3**. For the matrix D, the tree returned by the NJ algorithm can be represented as $((d,(a,b)),c,e)$ in Newick format. This means that the first cherry formed by the algorithm was (a,b). For this to hold true, the Q-criterion of the matrix D must satisfy the following nine inequalities:

$$Q(a,b) \leq Q(a,c), Q(a,d), \ldots, Q(c,e), Q(d,e). \tag{4.1}$$

Since each entry $Q(i,j)$ is a linear equation on the entries of D, these equations form semialgebraic sets in $\mathbb{R}^{15}$. All input matrices D_5 lying in the intersection of the nine half-spaces listed in (4.1) will return two possible agglomerated trees. In general, these halfspaces form a polyhedral cone in $\mathbb{R}^{\binom{n}{2}}$, and every input that satisfies them will receive the same output. Thus, we identify this cone with the two labeled agglomerated trees. For D' the NJ algorithm associates the trees $((d,(c,e)),a,b)$ and $((a,b),(c,e),d)$. Thus, the nine inequalities for D' are not the same as those for D. Therefore, the trees lie on different cones, even though they are the same topologically.

Eickmeyer and Yoshida (2008) and Eickmeyer et al. (2008) studied the defining inequalities of the NJ cones for small taxa. In other words, the cones were described using hyperplanes, also known as an H-representation. To give a full combinatorial description, one requires a V-representation: to describe the vectors for the extreme rays of the cones. In contrast, for

the UPGMA algorithm (Sokal and Michener, 1958), a V-representation for the corresponding polyhedral cones was given in Davidson and Sullivant (2013).

Remark 4.1. Ideally, a phylogenetic method would depend only on the quality of the data it receives as input, so we would expect it to return each tree with equal probability. This would imply that the algorithm has no bias toward a specific type of tree.

One can estimate this probability by measuring the *spherical volume* of the polyhedral cones, which is the proportional volume when intersecting the cones with the unit sphere in $\mathbb{R}^{\binom{n}{2}}$. A bias in the algorithm toward some trees can be detected when the spherical volume is not roughly the same for each tree. Computational methods based in this paradigm are used to evaluate the performance of NJ as a heuristic for LSP in Davidson and Sullivant (2014) and of NJ as a heuristic for BME in Eickmeyer et al. (2008). Davidson and Sullivant (2013) approximated the spherical volume of the UPGMA cones finding cones with considerable less volume than others, unveiling a bias of the algorithm toward balanced trees (rooted trees with two children of the root defining subtrees with a similar amount of leaves).

We studied the approximated spherical volume of the NJ cones, by analyzing how simulated data lies in the cones. We implemented the NJ algorithm in Mathematica (Wolfram Research Inc, 2020). Our implementation takes as input a dissimilarity map, represented as a symmetric real matrix, and returns the agglomerated tree as well as all the agglomeration steps. Our software, as well as the results of the computations discussed in the rest of the paper, is available at Davidson and Martín del Campo (2020).

4.1. Simulation Results

In order to approximate the volume of the polyhedral cones, for $n = 4, \ldots, 8$, we uniformly sampled 1,000,000 data matrices lying inside the unit sphere and the positive orthant of $\mathbb{R}^{\binom{n}{2}}$. For each, our software computed its corresponding agglomerated tree. We counted the number of matrices that were associated to each agglomerated tree.

Table 2 displays our results for 4 taxa. The first column and third represents the agglomerated trees in Newick format, each row has the two trees identified by the NJ algorithm. The second and forth columns show the percentage of the million random instances output by NJ. Note that in this case, the NJ algorithm should output 3 trees, due to Lemma 2.2. Hence, we should expect that three of the six trees should not had been observed. However, the three trees in the first column accumulated only 86.7771% of the random input matrices, and the remaining instances were equally distributed among the other three cones. We noted this behavior is due to the way Mathematica resolves the double minimum conflict.

Each row of **Table 2** reports the observed instances in each polyhedral cone, so adding the corresponding percentage for each row gives the distribution of the instances lying in each cone. Here, we noted that the row sum is around 33.333%, showing that each instance is equally distributed among the three cones. In light of this fact the NJ algorithm does not have a bias resulting from variation in cone size in the case of 4 taxa. We shall see this is not the case for larger numbers of taxa. In the case where $n \geq 5$ the last decision step of NJ can be studied in $\mathbb{R}^6$, but at least one of $\{a, b, c, d\}$ is a bouquet of size at least 2.

TABLE 2 | Percentage of the 1,000,000 samples in each agglomerated tree with 4 taxa.

Tree	Percentage	Tree	Percentage
4 taxa			
((*ab*)*cd*)	29.1610	((*cd*)*ab*)	4.1150
((*ac*)*bd*)	28.9944	((*bd*)*ac*)	4.4149
((*ad*)*bc*)	28.6217	((*bc*)*ad*)	4.6930

TABLE 3 | Percentage of 1,000,000 samples found in each agglomerated tree.

Tree	Percentage	Tree	Percentage
Uniform bias correction			
4 taxa			
((*ab*)*cd*)	16.8740	((*cd*)*ab*)	16.4977
((*ac*)*bd*)	16.7543	((*bd*)*ac*)	16.5304
((*ad*)*bc*)	16.5622	((*bc*)*ad*)	16.7814

We remark that all implementations have to resolve the decision of the tie in the last step, and the original paper of Saitou and Nei (1987) noticed the tie in equations (11b) and (11c), suggesting to resolve a tie between (*ab*) and (*cd*) by choosing to agglomerate the one with the smallest index (lexicographic), in this case (*ab*). We call this a *representative bias*, and we explored two ways to correct it.

4.2. Representative Bias Correction

There could be many ways to resolve the tie in the last step of the algorithm. One could choose a distinguished set of pairs to resolve the clash of minima. For instance, we could always choose to agglomerate the pair involving the lowest (or highest) index. This choice would make it so that all instances in **Table 2** would be equally distributed between the trees $((1, 2), 3, 4)$, $((1, 3), 2, 4)$, and $((1, 4), 2, 3)$, and the other three would not observe any instances. This could be desirable for practical reasons. However, this convenient choice has an impact on the interpretation of the output, as we would be choosing beforehand a step in which the algorithm joins specific taxa. Thus, we explored the impact of some different choices.

Besides from the lexicographic choice, another natural way to correct the ambiguity in the last step of the NJ algorithm is to choose uniformly at random one of the two coinciding agglomerations. We repeated the experiment with this correction in our software, and found this choice equally distributes the number of instances observed in each agglomerated tree. We refer to this representative bias correction just as *Uniform*. **Table 3** displays the results for 4 taxa after this correction.

Lastly, we also consider another representative bias correction method, that considers previous agglomeration information. As noticed, the last step of the algorithm could have to decide

TABLE 4 | Fraction of 1,000,000 samples in each tree for different bias corrections.

Tree	Regular	Uniform	Baggage	Tree	Regular	Uniform	Baggage
5 taxa							
((*bd*)(*ac*)*e*)	2.9818	1.6625	0.3924	((*e*(*ac*))*bd*)	0.3349	1.6562	2.9713
((*ad*)(*bc*)*e*)	3.0313	1.6897	0.3798	((*e*(*bc*))*ad*)	0.3401	1.6503	2.9405
((*cd*)(*ab*)*e*)	3.0006	1.7055	0.3833	((*e*(*ab*))*cd*)	0.3384	1.6309	2.9239
((*bc*)(*ad*)*e*)	2.9995	1.6981	0.3864	((*e*(*ad*))*bc*)	0.3362	1.6335	2.9338
((*ae*)(*cd*)*b*)	2.9926	1.6741	0.3619	((*b*(*cd*))*ae*)	0.3553	1.6657	2.9849
((*ab*)(*cd*)*e*)	3.0014	1.6839	0.3991	((*e*(*cd*))*ab*)	0.3369	1.6553	2.9486
((*bc*)(*ae*)*d*)	3.0251	1.6716	0.3839	((*d*(*ae*))*bc*)	0.3347	1.6510	2.9594
((*bd*)(*ae*)*c*)	2.9969	1.6679	0.3538	((*c*(*ae*))*bd*)	0.3689	1.6716	2.9748
((*ce*)(*ab*)*d*)	2.9730	1.6668	0.3619	((*d*(*ab*))*ce*)	0.3524	1.6545	2.9693
((*a*(*bc*))*de*)	2.9880	1.6459	2.9704	((*de*)(*bc*)*a*)	0.3717	1.6588	0.3558
((*ae*)(*bd*)*c*)	2.9930	1.6647	0.3560	((*c*(*bd*))*ae*)	0.3581	1.6634	2.9807
((*be*)(*ad*)*c*)	2.9162	1.6854	0.3611	((*c*(*ad*))*be*)	0.3510	1.6643	2.9518
((*b*(*ac*))*de*)	2.9415	1.6558	2.9923	((*de*)(*ac*)*b*)	0.3825	1.6723	0.3709
((*ae*)(*bc*)*d*)	2.9802	1.6918	0.3610	((*d*(*bc*))*ae*)	0.3489	1.6358	2.9594
((*ac*)(*be*)*d*)	3.0146	1.7032	0.3772	((*d*(*be*))*ac*)	0.3459	1.6668	2.9248
((*c*(*ab*))*de*)	2.9679	1.6371	2.9771	((*de*)(*ab*)*c*)	0.3800	1.6696	0.3582
((*a*(*cd*))*be*)	2.9911	1.6576	2.9827	((*be*)(*cd*)*a*)	0.3826	1.6674	0.3530
((*a*(*be*))*cd*)	2.9537	1.6768	2.9807	((*cd*)(*be*)*a*)	0.3776	1.6902	0.3647
((*a*(*ce*))*bd*)	2.9624	1.6526	2.9743	((*bd*)(*ce*)*a*)	0.3668	1.6915	0.3623
((*ad*)(*ce*)*b*)	2.9964	1.6863	0.3570	((*b*(*ce*))*ad*)	0.3746	1.6500	2.9664
((*ad*)(*be*)*c*)	2.9102	1.6850	0.3823	((*c*(*be*))*ad*)	0.3603	1.6433	2.9816
((*be*)(*ac*)*d*)	2.9823	1.6898	0.3651	((*d*(*ac*))*be*)	0.3610	1.6402	2.9838
((*a*(*de*))*bc*)	2.9475	1.6272	2.9928	((*bc*)(*de*)*a*)	0.3823	1.6826	0.3604
((*b*(*ad*))*ce*)	2.9571	1.6316	2.9331	((*ce*)(*ad*)*b*)	0.3788	1.6946	0.3600
((*ac*)(*de*)*b*)	2.9495	1.6927	0.3682	((*b*(*de*))*ac*)	0.3603	1.6336	2.9816
((*ab*)(*ce*)*d*)	2.9766	1.6883	0.3851	((*d*(*ce*))*ab*)	0.3364	1.6620	2.9377
((*ab*)(*de*)*c*)	2.9929	1.6864	0.3712	((*c*(*de*))*ab*)	0.3298	1.6542	2.9533
((*b*(*ae*))*cd*)	2.9349	1.6516	2.9651	((*cd*)(*ae*)*b*)	0.3642	1.6660	0.3585
((*a*(*bd*))*ce*)	2.9472	1.6508	2.9784	((*ce*)(*bd*)*a*)	0.3736	1.6882	0.3614
((*ac*)(*bd*)*e*)	2.9800	1.7034	0.3772	((*e*(*bd*))*ac*)	0.3304	1.6521	2.9564

between joining two sets of taxa A, B, over other two C, D, and we could take biological information in consideration for making this choice. For instance, one could choose the representative that joins the pair containing more taxa: we call this representative bias correction method *Baggage*. If, in the last step, the NJ algorithm has to decide between an α or a β step, Baggage would chose β over α as it is joining two bouquets having at least 2 leaves in each, as opposed to α that is joining two stems, i.e., two leaves. Formally, we think of the NJ algorithm as joining sets of taxa, starting from a list of n singletons, and stopping until we obtain 3 taxa sets. For a taxa set A we let $|A|$ denote the size of A, which is the number of taxa it contains (alternatively, we can think of $|A|$ as the size of a bough). Then, Baggage chooses to join taxa sets A and B over C and D, if $|A| + |B| > |C| + |D|$. If both pairs of sets have the same taxa, so $|A| + |B|$ equals $|C| + |D|$, we could use the uniform method to solve this case. We note that making the opposite choice –joining a pair A, B that is more balanced– could produce a representative bias toward topologically balanced trees similar to the one discovered in the UPGMA algorithm (Davidson and Sullivant, 2014) and observed in NJ in some cases for $n \leq 14$.

For $n = 4$, the Baggage correction and the Uniform coincide, but this is not the case for $n \geq 5$. We show in **Table 4** the comparison for 5 taxa of our implementation without representative bias correction (column labeled Regular), vs. the Uniform, and Baggage bias correction methods. To interpret the results, in each row we display the percentage obtained for the two trees identified by the NJ algorithm. We notice that for every row, the sum of the entries of each representative bias correction for both trees is around 3.3333%. Thus, from Remark 4.1 the NJ method have no bias. However, if the NJ method had no representative bias, each agglomerated tree should be returned with probability 1.666%, which is very close to what is obtained in the Uniform method.

In **Table 4** we see again that our original implementation had an unexpected bias due to the way Mathematica sorted

the double minima conflict in the last step. The values in each row corresponding to the column Regular should be zero for all the trees in the columns of the right-hand side, according to the lexicographic choice. The Uniform method seems to correct the representative bias completely, making each agglomerated tree occur with equal probability. Lastly, the Baggage method creates a representative bias toward agglomerated trees formed with less α steps. There could be biological reasons to correct the bias to agree with what is observed, such as high confidence in the appearance of a subtree within a larger taxon set. We realized these computations for $n = 4, \ldots, 8$ taxa, obtaining similar results to those displayed in **Table 4** and are available at Davidson and Martín del Campo (2020).

We conclude by remarking that it is left to understand other reasonable ways to explore methods of bias correction, and understand their combinatorial implications. Knowing about different representative biases that can arise in the algorithm not only helps to develop other uses for this agglomerative method, but also to aid in the study of bias from other heuristics for LSP or BME as well as phylogenetic pipelines that contain NJ as a component.

DATA AVAILABILITY STATEMENT

Publicly available datasets were analyzed in this study. This data can be found here: https://www.cimat.mx/~abraham.mc/NJCones/NJCones.html.

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

RD was partially supported by US–NSF grant DMS-1401591. Work of AM was supported by CONACYT under grant A1-S-30035 and Cátedras-1076.

REFERENCES

Aigner, M. (1998). Motzkin numbers. *Eur. J. Combin.* 19, 663–675. doi: 10.1006/eujc.1998.0235

Aigner, M. (1999). Catalan-like numbers and determinants. *J. Combin. Theory Ser. A* 87, 33–51. doi: 10.1006/jcta.1998.2945

Bóna, M. (Ed.). (2015). *Handbook of Enumerative Combinatorics. Discrete Mathematics and Its Applications (Boca Raton)*. Boca Raton, FL: CRC Press.

Bryant, D. (2005). On the uniqueness of the selection criterion in neighbor-joining. *J. Classif.* 22, 3–15. doi: 10.1007/s00357-005-0003-x

Davidson, R., and Martín del Campo, A. (2020). *Supplementary Materials. "Combinatorial and Computational Investigations of Neighbor-Joining Bias"*. Available online at: https://www.cimat.mx/~abraham.mc/NJCones/NJCones.html

Davidson, R., and Sullivant, S. (2013). Polyhedral combinatorics of UPGMA cones. *Adv. Appl. Math.* 50, 327–338. doi: 10.1016/j.aam.2012.10.002

Davidson, R., and Sullivant, S. (2014). Distance-based phylogenetic methods around a polytomy. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 11, 325–335. doi: 10.1109/TCBB.2014.2309592

Day, W. (1987). Computational complexity of inferring phylogenies from dissimilarity matrices. *Bull. Math. Biol.* 49, 461–467.

Desper, R., and Gascuel, O. (2004). Theoretical foundation of the balanced minimum evolution method of phylogenetic inference and its relationship to weighted least-squares tree fitting. *Mol. Biol. Evol.* 21, 587–598. doi: 10.1093/molbev/msh049

Deutsch, E., and Shapiro, L. W. (2002). A bijection between ordered trees and 2-Motzkin paths and its many consequences. *Discrete Math.* 256, 655–670. doi: 10.1016/S0012-365X(02)00341-2

Disanto, F., and Wiehe, T. (2013). Exact enumeration of cherries and pitchforks in ranked trees under the coalescent model. *Math. Biosci.* 242, 195–200. doi: 10.1016/j.mbs.2013.01.010

Eickmeyer, K., Huggins, P., Pachter, L., and Yoshida, R. (2008). On the optimality of the neighbor-joining algorithm. *Algorith. Mol. Biol.* 3:5. doi: 10.1186/1748-7188-3-5

Eickmeyer, K., and Yoshida, R. (2008). "The geometry of the neighbor-joining algorithm for small trees," in *Algebraic Biology*, eds K. Horimoto, G. Regensburger, M. Rosenkranz, and H. Yoshida (Berlin; Heidelberg: Springer Berlin Heidelberg), 81–95. doi: 10.1007/978-3-540-85101-1_7

Felsenstein, J. (2004). *Inferring Phylogenies*. Sunderland, MA: Sinauer Associates, Inc.

Gascuel, O., and Steel, M. (2006). Neighbor-joining revealed. *Mol. Biol. Evol.* 23, 1997–2000. doi: 10.1093/molbev/msl072

Jiang, T., and Lee, D. (Eds.). (1997). *The Performance of Neighbor-Joining Algorithms of Phylogeny Reconstruction*, Vol. 1276. Berlin; Heidelberg: COCOON; Springer.

Lando, S. K. (2003). *Lectures on Generating Functions, Volume 23 of Student Mathematical Library*. Providence, RI: American Mathematical Society.

Lee, T., Guo, H., Wang, X., Kim, C., and Paterson, A. (2014). Snphylo: a pipeline to construct a phylogenetic tree from huge snp data. *BMC Genomics* 15:162. doi: 10.1186/1471-2164-15-162

Liu, L., and Yu, L. (2011). Estimating species trees from unrooted gene trees. *Syst. Biol.* 60, 661–667. doi: 10.1093/sysbio/syr027

Meshkov, V. R., Omelchenko, A. V., Petrov, M. I., and Tropp, E. A. (2010). Dyck and Motzkin triangles with multiplicities. *Mosc. Math. J.* 10, 611–628, 662. doi: 10.17323/1609-4514-2010-10-3-611-628

Mihaescu, R., Levy, D., and Pachter, L. (2009). Why neighbor-joining works. *Algorithmica* 54, 1–24. doi: 10.1007/s00453-007-9116-4

Oste, R., and Van der Jeugt, J. (2015). Motzkin paths, Motzkin polynomials and recurrence relations. *Electron. J. Combin.* 22:19. doi: 10.37236/4781

Saitou, N., and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* 4, 406–425.

Semple, C., and Steel, M. (2003). *Phylogenetics. Oxford Lecture Series in Mathematics and Its Applications*. Oxford: Oxford University Press.

Sloane, N. J. A. (2020). *The On-line Encyclopedia of Integer Sequences*. OEIS Foundation Inc. Available online at: http://oeis.org

Sokal, R. R., and Michener, C. D. (1958). A statistical method of evaluating systematic relationships. *Univ. Kansas Sci. Bull.* 38, 1409–1438.

Speyer, D., and Sturmfels, B. (2003). The tropical grassmannian. *Adv. Geometry* 4, 389–411. doi: 10.1515/advg.2004.023

Studier, J. A., and Keppler, K. J. (1988). A note on the neighbor-joining algorithm of Saitou and Nei. *Mol. Biol. Evol.* 5, 729–731.

Telles, G., Araújo, G., Walter, M., Brigido, M., and Almeida, N. (2018). Live neighbor-joining. *BMC Bioinformatics* 19:172. doi: 10.1186/s12859-018-2162-x

Warnow, T. (2017). *Computational Phylogenetics: An Introduction to Designing Methods for Phylogeny Estimation*. Cambridge: Cambridge University Press.

Wolfram Research Inc (2020). *Mathematica, Version 12.1*. Champaign, IL: Wolfram Research Inc. Available online at: https://www.wolfram.com/mathematica

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.