Stockholms universitet

With a Little Help From a Friend: Investigating an Accurate and Scalable Divide-And-Conquer NJ-Like Heuristic

Amy Lee Jalsenius

Handledare: Lars Arvestad
Examinator: Marc Hellmuth
Inlämningsdatum: January 3rd, 2024

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

Phylogenetic trees graphically represent the evolutionary history of a set of related taxa. In most cases, the exact tree topology is unknown, and must therefore be inferred by analyzing biological sequence data. While probabilistic models are known for inferring trees with excellent accuracy, these methods do not scale well for large data sets due to their computational cost. This is why the faster, computationally less demanding, though less accurate, distance based methods are useful. The distance based methods are capable of producing trees with acceptable accuracy that can be used as starting trees for other, more demanding models, or to gain an overview of large data sets.
The popular distance based method Neighbor Joining is easy to understand and implement, and is known to produce trees of relatively high accuracy. However, as modern sequencing technology is leading to data sets of ever-increasing size, its cubic runtime is becoming a hurdle. This has lead to a search for new, scalable algorithms that still feature the same good accuracy. Alternatives that have been suggested so far have either retained the unfortunate run time, or infer trees with significantly worse accuracy. In an attempt to remedy these issues, Lars Arvestad has developed a randomized, divide-and-conquer approach with significantly faster run time ($O(n \lg n)$ for many data sets and $O(n^2)$ in worst case). Unfortunately, the accuracy does not match that of Neighbor Joining. In a more recent attempt, Arvestad has proposed yet another randomized, divide-and-conquer method which would hopefully strike a better balance between speed and accuracy. The work of this report has been to investigate this. We have implemented the algorithm and assessed it on simulated data. The output and performance has been compared with the previous method and Neighbor Joining in particular. The results are promising and indicate a significant improvement over the previous method while retaining the same attractive run time.

Sammanfattning

Fylogenetiska träd används för att illustrera de evolutionära släktskapen mellan taxa. I de flesta fall är trädens topologi okända och måste därför bestämmas genom att analysera biologisk sekvensdata. Sannolikhetsmodeller är kända för att konstruera träd med utmärkt tillförlitlighet, men dessa metoder är inte speciellt skalbara då de är mycket beräkningskrävande. Av den anledningen används mindre beräkningskrävande, avståndsbaserade modeller. Dessa kan producera träd av acceptabel kvalitet vilka kan användas som grund för de mer krävande sannolikhetsmodellerna, eller för att snabbt få en uppfattning av släktskapen i stora datamängder.
En populär avståndsbaserad metod är Neighbor Joining vilken är lätt att förstå och implementera, och är känd för att producera träd av god kvalitet. Med moderna sekvenseringsmetoder ökar dock datamängderna så till den grad att även Neighbor Joining med sin kubiska tidskomplexitet blir för långsam. Detta har lett till ett behov av snabbare alternativ som även producerar träd av samma höga kvalitet. Hitintills har de snabbare alternativen inte lyckats leverera träd av tillräckligt hög kvalitet. I ett försök att råda bot på detta har Lars Arvestad utvecklat en randomiserad, divide-and-conquer-metod som är påtagligt snabbare än Neighbor Joining ($O(n \lg n)$ i många fall och $O(n^2)$ in värsta fall). Tyvärr uppvisar metoden inte tillräckligt hög kvalitet på träden den producerar. I ett nytt försök att hitta en balans mellan hastighet och kvalitet har Arvestad föreslagit en ny randomiserad, divide-and-conquer-metod. I denna rapport har vi undersökt denna nya metod och utvärderat dess prestanda. Vi har implementerat algoritmen och testat den med syntetisk data. Dess prestanda har jämförts med den tidigare metoden och Neighbor Joining. Resultaten är lovande och indikerar en rejäl förbättring av den tidigare metoden med bibehållen tidskomplexitet.

Contents

1 Introduction

1.1 Question of Interconnectedness and Why it Matters

Phylogenetic trees are often used as a graphical representation of how a given set of taxa[1] S, for whom an interlinked evolutionary history is assumed, are interrelated with one another. Each leaf in the tree represents an element of S, i.e. an individual taxon, and the interior vertices the hypothetical ancestors connecting them. We assume that each of our taxon is represented by some form of biological sequence.

Depending on what our taxa in the set S are, an understanding of their interrelation can give insight into different problems. For example, in the case that the taxon are different species or populations, we can answer interesting historical questions such as when the acquisition of new functions may have occurred, how they evolved [41] and in turn at which point a new species has arisen from an already existing species [6]. If our taxon set S is a group of related genes (i.e. a gene family) we might use the phylogenetic tree inferred to predict gene function, classify environmental DNA sequence, or identify functional residues [29].

It is therefore of great use, provided some form of evolutionary data, to infer what these phylogenetic trees may look like. In general, if the primary goal is to infer a tree T that has the highest probability of being the correct tree for your set S, then the approach would be to apply a probabilistic method such as Maximum Likelihood or Bayesian inference. However, as pointed out in [11] these methods come at a prohibitive computational cost compared to distance-based methods such as NEIGHBOR-JOINING which, while admittedly less accurate than the aforementioned, take just a fraction of the time and often have good enough accuracy for many cases. For example, distance-based methods are the go-to approach in the case of handling very large sequence sets for which at least an approximate tree may be produced [5]. In the case of a large sequence set but with low sequence divergence it has even been shown in [40] that NEIGHBOR-JOINING produces accurate results very quickly. Distance-based methods will also be used when we need to infer many trees as is required for boot-strapping, a standard method used to estimate reliability in inferred trees [10]. Furthermore, the trees inferred with distance-based methods, even in the case that they are less accurate, can still be used as so-called guide trees for alignment algorithms as shown in [9] or to generate clever starting trees required as input for one of the more costly probabilistic methods.

All of these purposes justify why there remains a great interest in distance-based methods. However, as sequencing technology is rapidly improving, the size of the sequence sets are growing, and, for reasons briefly explained below, even the current distance-based methods will start to struggle, thus creating an impending need for the development of even faster, distance-based methods.

[1]Throughout a taxon is understood to be some form of biological entity such as genes, species, individuals, or populations, from which we can sample data in the form of a biological sequence.

1.2 On Strings and Alignments

When we have sampled from each of the taxon in our set S biological sequence data (i.e. protein, DNA, or RNA), this can be used to produce evolutionary data. We start first by noting that every sequence type has a standardized alphabet associated with it, and each individual taxon sequence $s \in S$ can therefore be represented by a string of characters in the relevant alphabet. In order to identify regions of similarity and dissimilarity, one might, using some form of method computationally create a good[2] pairwise alignment for all of our sequences in S, producing a so-called MSA, or *multi sequence alignment.* Note that we assume that all of our sequences are of a similar length approximately L, and that it is the individual characters along the string being aligned against one another, with, if necessary, gaps being introduced strategically during the alignment process depending on the exact method used.[3] From our MSA, we can now, by selecting an appropriate substitution model calculate an "evolutionary distance" for each pair of sequence in our MSA. In the general case, the distance between two taxa is often measured by the number of substitutions per site that are estimated (with the model) to have taken place along the corresponding evolutionary path for each taxon since they split off from a shared ancestor, with the distance often some measure of mismatches between the sequences at the aligned character positions with areas of gaps either also counting as mismatches, or alternatively being ignored, depending on the model [3]. These models can be used not only for estimating evolutionary distance between sequences, but also for simulating sequences, something that we do with the WAG model [37] when creating some of the data used in our experiments.

This measure of distance between all of the pairs in a given taxa sequence set S can then be used to create a so-called distance matrix, an $|S| \times |S|$ matrix containing all of our distance estimates for each pair of taxa in our set S.

1.3 The Gold Standard of Distance Methods: NEIGHBOR-JOINING

A distance matrix is the required input for one of the most popular distance-based methods, NEIGHBOR-JOINING. The algorithm has been often referred to as probably the most cited algorithm in the field of bioinformatics [1, 11, 25, 29]. It was first presented by Saitou and Nei in 1987 [31] with simplified but equivalent formulas, used here, presented a year later in [33]. NEIGHBOR-JOINING has been described as a greedy heuristic for the balanced minimum evolution (BME) criterion [13]. The BME criterion requires for a given set S a tree of minimum length, with a tree's length defined as the sum of all branch weights in the tree. When we refer to it as a "greedy heuristic" we simply mean that in each iteration NEIGHBOR-JOINING attempts to construct this "tree of minimum length" with the straightforward strategy of, in each step, always joining the two vertices that

[2]Due to the prohibitive computational cost required for finding the optimal alignments of S with sequences of moderate length, most MSA are produced with a program that uses heuristic methods.

[3]For a very interesting oversight and introduction to the concept of MSA and the diverse algorithmic methods used in their creation, we refer the reader to [7].

have been deemed by the so-called selection function to be neighbors (i.e. adjacent) in the phylogenetic tree.

Despite its simplistic approach, NEIGHBOR-JOINING has not only been shown to perform remarkably well empirically but that these good results have been justified theoretically. For example, NEIGHBOR-JOINING has a convergence radius (known as Atteson's convergence radius) which guarantees that the "true" tree topology $\hat{T}$ for a set of taxa S will be returned if the distance function $\mathcal{D}$ is additive or nearly additive, and has been shown to return $\hat{T}$ in even more cases than that empirically. However, this inference accuracy for n sequences comes at the time complexity of $O(n^3)$ [1, 11, 13, 33], which, while remaining faster than the known statistical methods, begins to form a hurdle as n grows.

1.4 True Improvements?

Suggestions for developing algorithms with accuracy similar to NEIGHBOR-JOINING but considerably faster, have often involved primarily either clever implementations or a strategy for limiting unnecessary comparisons in the search for which taxa to join. For example, the algorithm DECENTTREE [36] uses highly optimized and parallel C++ implementations of both NEIGHBOR-JOINING and BIONJ. The algorithm BIONJ [15] utilizes the same agglomerative scheme as NEIGHBOR-JOINING but additionally takes the variance of distance estimation into consideration, and by doing so significantly improves its estimates of what constitutes a true neighbor-pair, which, while not an improvement on speed, was shown to improve accuracy in some cases.

FAST-NEIGHBOR-JOINING [11] uses the same selection function that is used in NEIGHBOR-JOINING but is capable of retaining information between iterations, and uses this to limit the search for a node's neighbor to a smaller set enabling joins to be carried out in linear time. FAST-NEIGHBOR-JOINING in particular has proven popular enough that it has served as direct inspiration for many of the other algorithms developed. DNJ and HNJ both introduced in [5] have been shown to handle sequence set sizes up to $|S| = 10^6$ by using similar operations as those in FAST-NEIGHBOR-JOINING combined with good implementation.

Unfortunately, though, as pointed out in [1, 19, 29] even if these algorithms can infer a tree on n sequences in $O(n^2)$ time, one must not forget the work needed to construct the initial distance matrix required for these methods as input. If we assume that we have n sequences, all of length L, then we have $\binom{n}{2}$ distances that must be estimated initially, with each estimate requiring L comparisons. Let us now further assume that L is comparable in size to n. This is quite a reasonable assumption as modern sequencing techniques are becoming ever faster and cheaper, leading to both massive data sets, and sequences whose lengths are ever-improving (see for example [8]). This would ultimately mean that all of the algorithms mentioned above, having time complexity $O(n^2L)$, would in fact actually have, in the reasonable case of $L \backsim n$, time complexity $O(n^3)$ and therefore offer in fact no true improvement with regards to time complexity compared to NEIGHBOR-JOINING.

Focus should therefore be on developing algorithms that do not require the costly initial distance matrix M_S, and almost certainly on reducing the number of pairwise distance estimations needed to be carried out in general [1, 18] .

1.5 Novel Randomized Divide-And-Conquer Approaches

Arvestad's DNCTREE [1] was developed to take a randomized divide-and-conquer approach when inferring a tree for a set of n aligned sequences, S, using NEIGHBOR-JOINING as a subroutine. To start with, rather than the costly distance matrix required for the algorithms above, DNCTREE instead takes the aligned sequences and a distance function $\mathcal{D}_S$ and uses this to calculate distances only when they are needed, which, thanks to the divide-and-conquer approach, is comparatively very limited.

Details of the algorithm will be gone through in section 2, but by utilizing the same selection function for pairing taxa as NEIGHBOR-JOINING, the original S is partitioned recursively into three sub-problems using three randomly selected sequences from S to guide the paritioning, until a base-case case size of k is reached, at which point NEIGHBOR-JOINING will be run. While Atteson's convergence radius was proven to apply to DNCTREE in [1] and work was reduced significantly, the trees produced in DNCTREE did not exhibit the same accuracy as those produced with NEIGHBOR-JOINING, in particular when the base case size k was relatively small.

The main goal in developing DNCTREE-K was to improve the accuracy of DNCTREE while at the same time still keeping the workload (i.e. the number of pairwise distances calculated) down as much as possible, with the main strategy focusing on improving the quality of the partitions being used to create sub-problems. Empirical studies run for this thesis indicate that this approach has led to a significant improvement regarding accuracy issues, while having, like DNCTREE, a best case runtime of $O(n \lg n)$ and in a worst case $O(n^2)$.

1.6 An Oversight of the Thesis

To aid the reader, we will now give a brief oversight to the organization of this thesis. In Section 2, we will start first by looking at the most critical definitions for concepts covered in this thesis, followed then by taking a closer look at the two key "predecessor" algorithms, NEIGHBOR-JOINING and DNCTREE. We will also go over the metrics used in our experiments for tree comparison, and explain how the data was simulated. In Section 3 the new DNCTREE-K algorithm will be introduced and examined depth. In Section 4 we will provide the results of the initial experiments conducted with DNCTREE-K using simulated data. Here, a side-by-side performance comparison between DNCTREE-K, DNCTREE, and in some cases NEIGHBOR-JOINING will be presented. Section 5 is where our concluding results regarding the performance of DNCTREE-K and some interesting possible points of further investigation will be discussed. Finally, information on code availability is found in Section 6.

2 Preliminaries

We will first cover general terminology and definitions in Section 2.1, followed by an introduction to the Newick format we use to represent trees in section 2.2, and a brief overview of the two metrics we will use to compare trees in our experiments in Section 2.3. Finally we will introduce the canonical NEIGHBOR-JOINING algorithm in Section 2.5 and give an overview of Arvestad's original DNCTREE algorithm in Section 2.6. How the data used in our experiments was simulated will be covered in Section 2.7 and in Section 2.8 we will conclude with a very brief description of Snakemake, the automation tool used for setting up the workflow when evaluating the algorithms.

2.1 Terminology and Definitions

Throughout this thesis we often use the letter S to denote a set of taxa. We use $\mathcal{D}_S$ to denote a distance function $\mathcal{D}_S : S \times S \to \mathbb{R}^+$, where $\mathcal{D}_S$ always satisfies $\mathcal{D}_S(x, y) = \mathcal{D}_S(y, x)$ (i.e., it is symmetric) and $\mathcal{D}_S(x, x) = 0$ for all $x, y \in S$. Instead of writing $\mathcal{D}_S(x, y)$ we will often, as a shorthand, use the notation d_{xy} to denote $\mathcal{D}_S(x, y)$, where S and $\mathcal{D}_S$ are understood from the context.

A *phylogenetic tree* T for a set of taxa S is an unrooted, edge-weighted tree with $|S|$ leaves where each leaf corresponds to one of the taxa in S. More precisely, there is a bijection between S and the leaves of T. Each internal node of T has degree 3. The idea with the phylogenetic tree is to capture the relationships between taxa such that the weighted path from one leaf x to another leaf y is similar to the distance $\mathcal{D}_S(x, y)$. We formalize this idea next. Because there is a bijection between the leaves of T and S we may use S to denote the leaf set of T.

Given a phylogenetic tree T for a set of taxa S, let $w(x, y)$ be the edge weight between any two nodes x, y of T. As a heads-up, note that we will often use the word node and vertex interchangeably. Because T is a tree, there is a unique, well-defined simple path between any two nodes x and y. Let P_{xy} denote this path. The weight of P_{xy}, denoted $w(P_{xy})$ is defined as the sum of all edge weights along P_{xy}. That is, suppose $P_{xy} = x, a, b, c, d, y$ then

$$w(P_{xy}) = w(x, a) + w(a, b) + w(b, c) + w(c, d) + w(d, y).$$

We also define $w(P_{xx}) = 0$ for all x. The distance function $\mathcal{D}_T : S \times S \to \mathbb{R}$ is defined as

$$\mathcal{D}_T(x, y) = w(P_{xy}) \tag{1}$$

for $x, y \in S$. Using this notation we are ready to formalize the idea of a phylogenetic tree T "explaining" the set of taxa S with respect to $\mathcal{D}_S$.

Definition 1. A phylogenetic tree T for a set of taxa S *realizes* $\mathcal{D}_S$ if $\mathcal{D}_T(x, y) = \mathcal{D}_S(x, y)$ for all $x, y \in S$.

The concept of "realizing $\mathcal{D}_S$" has been borrowed from [11]. We now have the following definition.

Definition 2. A distance function $\mathcal{D}_S$ for a set of taxa S is *additive* if there is a phylogenetic tree T for S that realizes $\mathcal{D}_S$.

A property of an additive distance function $\mathcal{D}_S$ is that the phylogenetic tree T that realizes it is unique [11]. We will use the notation $\hat{T}$ to denote this tree. The additive property is a rather strong property which does not necessarily hold for arbitrary distance functions $\mathcal{D}_S$. A slightly more relaxed property is the idea of "nearly additive" which we define below.

We define the difference between $\mathcal{D}_S$ and $\mathcal{D}_T$ as

$$|\mathcal{D}_S - D_T|_\infty = \max_{x,y \in S} |\mathcal{D}_S(x,y) - D_T(x,y)|.$$

Note that in the case of $\mathcal{D}_S$ being additive, we have

$$|\mathcal{D}_S - D_{\hat{T}}|_\infty = 0.$$

For a phylogenetic tree T, let μ_T denote the smallest edge weight over all edges of T. The concept of realizing $\mathcal{D}_S$ can now be extended as follows.

Definition 3. A phylogenetic tree T for a set of taxa S *nearly realizes* $\mathcal{D}_S$ if

$$|\mathcal{D}_S - D_T|_\infty \leq \frac{\mu_T}{2}. \tag{2}$$

Similarly to the concept of additive, the notion of "nearly additive" now follows.

Definition 4. A distance function $\mathcal{D}_S$ for a set of taxa S is *nearly additive* if there is a phylogenetic tree T for S that nearly realizes $\mathcal{D}_S$.

The uniqueness of a tree that realizes $\mathcal{D}_S$ also holds for the concept of nearly realizes. That is, if $\mathcal{D}_S$ is nearly additive then there is a unique tree T that nearly realizes $\mathcal{D}_S$[2]. We use $\hat{T}$ to denote this tree. Obviously, whenever $\mathcal{D}_S$ is additive then it is also nearly additive, hence $\hat{T}$ is referring to the same unique tree. It has been proven in [1, 2] that whenever $\mathcal{D}_S$ is nearly additive, both NEIGHBOR-JOINING and DNCTREE will return a tree with the same topology, or structure, as the phylogenetic tree $\hat{T}$. Note that DNCTREE does not return a tree with edge weights. However, in the case of $\mathcal{D}_S$ being nearly additive, it returns $\hat{T}$ without edge weights.

In the literature the distance between taxa is often expressed with a distance matrix rather than through a distance function $\mathcal{D}_S$. For example, the description of NEIGHBOR-JOINING in Section 2.5 states that the algorithm takes a distance matrix as input. We will use M_S to denote the $|S| \times |S|$ matrix containing all pairwise distances between taxa in S. In our implementation of NEIGHBOR-JOINING, DNCTREE and DNCTREE-K we do not use a matrix in a strict sense. Instead we use dictionaries that contain the

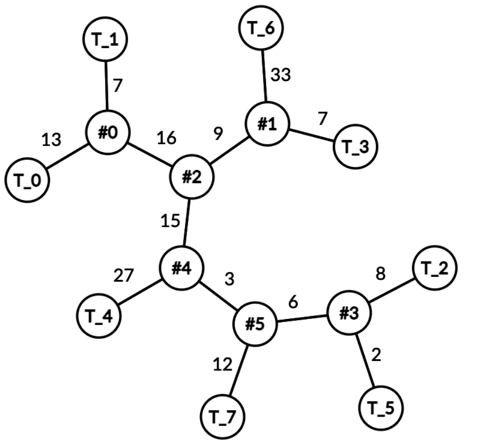

Figure 1: The string `(T_4,((T_0,T_1),(T_3,T_6)),(T_7,(T_2,T_5)));` is the Newick format of this tree. Note that the "`;`" is always used to indicate the end of the string. Here the node #4 has been used as the root stand-in.

distances of $\mathcal{D}_S$. This is a technical solution to the problem of storing and retrieving distances, as well as being able to add new distances between new taxa and new nodes ("pseudo taxa") which are created in the recursive steps of the algorithms. More on this will be detailed later.

2.2 The Newick Format

Within bioinformatics one of the most commonly used notations for representing trees is the Newick tree format[4] which uses a series of nested parenthesis and commas to express the tree topology. The representation is constructed recursively. While there are several variations of the Newick tree format regarding what information about the tree is relayed, the one we use only relays the leaf names, and ignores edge weights and internal nodes. Note that when an unrooted tree is expressed in Newick notation, an arbitrary vertex, which most often will be internal, will be selected as the root stand-in, that is starting point for the recursion. To give a concrete example, refer to Figure 1. This tree has following representation in Newick notation when we select node #4 as the root stand-in:

```
(T_4,((T_0,T_1),(T_3,T_6)),(T_7,(T_2,T_5)));
```

Here we ignore spelling out the edge weights and internal vertices. Thus, the representation above only captures the tree topology.

[4]How the format works was gleaned from the ETE3 tutorial page[12], and by studying the output of the tree data structure in the original DNCTREE algorithm, which we then used for DNCTREE-K.

2.3 Tree difference metrics

In order to compare phylogenetic trees we employ two difference metrics: the Robinson-Foulds Distance (RF distance) and the Tree Matching Distance (TMD). We begin by explaining the RF distance and its shortcoming, followed by a brief explanation to why the TMD provides a more robust tree difference metric.

2.3.1 The Robinson-Foulds Distance

The Robinson-Foulds Distance [30] is a popular metric used to compare trees. Its popularity is likely in part due to being intuitive to understand and relatively easy to compute. Given a phylogenetic tree T on the set of taxa S, let $\Gamma(T)$ denote the set of all subsets of S that can be obtained by partitioning S though a split of the tree T at an edge. That is, removing an edge naturally splits T into two sub-trees T_1 and T_2, where S_1 is the set of leaves in T_1 and S_2 is the set of leaves in T_2. The two sets S_1 and S_2 are therefore in the set $\Gamma(T)$. The set $\Gamma(T)$ contains all such leaf sets obtained by splitting T at every edge.

For two phylogenetic trees T_1 and T_2 on the same set of taxa S, the RF distance between T_1 and T_2 is defined as

$$\text{RF}(T_1, T_2) = |\Gamma(T_1) \setminus \Gamma(T_2)| + |\Gamma(T_2) \setminus \Gamma(T_1)|. \tag{3}$$

Another distance metric is the *relative RF distance* which is obtained by normalizing the RF distance with $2|S| - 6$. Thus, the relative RF distance is given by

$$\text{rel-RF}(T_1, T_2) = \frac{\text{RF}(T_1, T_2)}{2|S| - 6}. \tag{4}$$

To see why we normalize with $2|S| - 6$ we will show that $|\Gamma(T_1) \setminus \Gamma(T_2)| \leq 2|S| - 6$. By symmetry is follows that the same bound holds for $|\Gamma(T_2) \setminus \Gamma(T_1)|$. First observe that for a phylogenetic tree T, $|\Gamma(T)|$ equals the number of edges in T, times two. We prove the following lemma.

Lemma 5. *The number of edges of a phylogenetic tree with n leaves is $2n - 3$.*

Proof. Let T be a phylogenetic tree with n leaves. The tree can be made a rooted binary tree by taking an arbitrary edge and inserting a root on this edge. Since the number of leaves is n, the number of internal nodes of the binary tree, including the root, is $n - 1$. Hence there are $2(n - 1)$ edges in the rooted binary tree. By removing the root and replacing its two outgoing edges with the original edge of T, we conclude that the number of edges of T is $2n - 3$. □

Using the lemma above we conclude that $|\Gamma(T)| = 4|S| - 6$. Finally observe that for two phylogenetic trees T_1 and T_2 on the same taxa set S, $|\Gamma(T_1) \cap \Gamma(T_2)| \geq 2|S|$ since $|S|$ of the sets in $\Gamma(T_1)$ and $\Gamma(T_2)$ are represented by the single leaves S, and $|S|$ of the sets are represented by the complements of single leaves. Thus,

$$|\Gamma(T_1) \setminus \Gamma(T_2)| \leq (4|S| - 6) - 2|S| = 2|S| - 6,$$

which is what we wanted to show. By further dividing by 2 we scale the relative RF distance so that it has a maximum value of 1.

The Robinson-Foulds distance does have shortcomings. As pointed out in [23] the RF distance is poorly distributed, and therefore does not give as refined a measure of difference as one might hope, and simultaneously is easily saturated (its maximum value is reached) since it is extremely sensitive to very small changes.

We utilized the Python programming toolkit Environment for Tree Exploration (ETE3) [12] to compute the relative RF distance.

2.3.2 Tree Matching Distance

In order to address some of the issues with the Robinson-Foulds metric, the Tree Matching Distance, TMD, was developed as an alternative by Lin, Rajan and Moret in [23]. TMD is described as more robust compared to the RF distance, as it is not biased and does not saturate as easily. TMD also compares sets of subsets of taxa, but, unlike the RF distance, it also takes into account by how much subsets of taxa differ. For example, the two subsets $S_1 \subseteq S$ and $S_2 \subseteq S$ could be large and differ just on a single taxon. In terms of the RF distance, S_1 and S_2 are two completely different sets despite being very similar. In the TMD however, S_1 and S_2 are considered similar, and this therefore contributes to the distance metric differently than if S_1 and S_2 had been totally different. We refer to [23] for details. In order to calculate the TMD, Arvestad's Python module TREE-MATCHING-DISTANCE was used.

2.4 Center vertex of a Tree

In the DNCTREE-K algorithm in Section 3 we use the concept of a center vertex of a tree (not necessarily a phylogenetic tree). Here we define what a center vertex is and prove that a tree has either one or two center vertices. The notion of a center vertex given below and used in the DNCTREE-K algorithm should not be confused with the notion of the center of a star shaped tree which we use when explaining NEIGHBOR-JOINING.

Definition 6. A center vertex c of a tree T is a vertex for which the length (number of edges) of a longest simple path from c to a leaf of T is minimized.

Theorem 7. For any tree there are at most two center vertices. If there are two center vertices then they are adjacent.

Proof. Suppose that the vertex v of a tree T is a center vertex. Suppose that P is a longest path from v to a leaf and it goes via a neighbor w of v. Let $|P|$ denote the length of P.

Let T_v and T_w be the two subtrees of T obtained by removing the edge between v and w, where v belongs to T_v, and w belongs to T_w.

First we observe that v is the only vertex of T_v that can be a center vertex. Any other vertex of T_v has a longest path to a leaf that is longer than P. Namely, the path that goes to v and then follows P.

Second we note that there must be a longest path from v to a leaf of T_v of length exactly $|P| - 1$ or $|P|$. Suppose that this is not the case and the length of a longest path from v to a leaf of T_v is less than $|P| - 1$. In this case v could not be a center vertex since a longest path from w to a leaf would have length $|P| - 1$. Moreover, the length of a longest path from v to a leaf of T_v cannot be greater than $|P|$ by the assumption that $|P|$ is indeed the length of a longest path from v to a leaf.

Suppose that the length of a longest path from v to a leaf of T_v is $|P|$. In this case v is the unique center vertex of T. The longest path from a vertex of T_w to a leaf of T_v would have length at least $|P| + 1$.

Lastly, suppose that the length of a longest path from v to a leaf of T_v is $|P| - 1$. In this case w is also a center vertex; the length of a longest path from w to a leaf in T_v is $|P|$ and the length of a longest path from w to a leaf of T_w is $|P| - 1$. For any other vertex of T_w there is a path of length at least $|P| + 1$ to a leaf of T_v. Thus, w is the only vertex in addition to v that is a center vertex. □

2.5 The NEIGHBOR-JOINING Algorithm

The Neighbor Joining algorithm is a remarkably popular distance-based method for inferring phylogenetic trees, and is often used within bioinformatics pipelines [5]. It has been referred to in [1] as likely one of the most implemented algorithms in the world. Intuitive in function, it is also easy to implement, and relatively quick compared to more complex methods, delivering good accuracy in many cases. It has been studied extensively both empirically and theoretically, with particular interest being paid to why NEIGHBOR-JOINING performs so well [25].

In short, NEIGHBOR-JOINING is an agglomeration greedy algorithm, taking as input a distance matrix M_S for a set of taxa S. In each step it greedily chooses the pair of taxa that minimize the selection function given in equation 5, reducing the pair to a single vertex, and then returning the completed tree after $|S| - 3$ iterations.

It was proven in [2] that if $\mathcal{D}_S$ is nearly additive then NEIGHBOR-JOINING is guaranteed to return $\hat{T}$. As previously mentioned, this is referred to as Atteson's convergence radius. This convergence radius is a safety radius, with safety radius defined in [39] as "a radius from a tree metric (a distance matrix realizing a true tree) within which the input distance matrices must all lie in order to satisfy a precise combinatorial condition under which the distance-based method is guaranteed to return a correct tree." It was shown in [14] that NEIGHBOR-JOINING also has a corresponding *stochastic safety radius* for which when $|\mathcal{D}_S - D_T|_\infty$ is within the stochastic safety radius, then NEIGHBOR-JOINING is guaranteed to return $\hat{T}$ (referred to above as the "correct tree") *with a certain probability*.

This likely explains why NEIGHBOR-JOINING has been demonstrated to return $\hat{T}$ even in the case that $\mathcal{D}_S$ is not nearly additive (as mentioned in [1, 14, 39]).

In order to infer at tree for a set of taxa S, the input to NEIGHBOR-JOINING is the distance matrix M_S. At initiation, we have a completely unresolved tree (not phylogenetic tree) which takes the shape of a star; one internal vertex c and $|S|$ leaves corresponding to each taxon in S. The algorithm runs through $|S| - 3$ steps. In each step, two neighbors x and y of c are chosen according to a selection function. A new node z is then created and x and y are disconnected from c and re-connected to z. The node z is then connected to c. Hence the degree of c has decreased by one. This process is repeated until c has only three neighbors. After the last step, the initial star has been transformed into a phylogenetic tree where S is the leaf set and all internal nodes have degree 3.

After the step that disconnects x and y from c and connects the new node z to c, we must update the distance matrix accordingly so that x and y are no longer part of the matrix but z is. That is, we must add the distances between z and every other neighbor of c. These new distances are calculated according to Equation 8 below.

2.5.1 Selecting nodes and updating the distance matrix and edge weights

Here we describe how NEIGHBOR-JOINING selects two nodes to merge and how the distances between nodes are updated. Suppose that the current state of the algorithm is a star with the internal node c and n neighbors $x_1, \ldots, x_n$. There might be other vertices connected to the nodes x_i but they are no longer relevant when selecting two neighbors of c to merge. In the following we will refer to the node x_i simply by writing i. Let d_{ij} denote the distance between i and j. For each pair i, j we define q_{ij} as

$$q_{ij} = (n-2)d_{ij} - \sum_{k=1}^{n} d_{ik} - \sum_{k=1}^{n} d_{jk}. \tag{5}$$

In order two select the two neighbors of c to merge, we choose the i and j for which q_{ij} is smallest. We disconnect i and j from c and connect them to a new node z, which in turn is connected to c. Since we are building a phylogenetic tree we will also assign edge weights to the edges of the tree. That is, we need to assign a weight w_{iz} to the new edge (i, z) and a weight w_{jz} to the new edge (j, z). These weights are calculated as follows:

$$w_{iz} = \frac{1}{2}d_{ij} + \frac{1}{2(n-2)}\left(\sum_{k=1}^{n} d_{ik} - \sum_{k=1}^{n} d_{jk}\right), \tag{6}$$

$$w_{jz} = d_{ij} - w_{iz}. \tag{7}$$

We update the distance matrix by removing i and j and adding z. We therefore need to update the matrix with distances between z and all $n-2$ remaining neighbors of c. For a such a neighbor k, the distance d_{zk} (which is identical to d_{kz}) is calculated as

```
Algorithm 1 NEIGHBOR-JOINING($M_S$)
    1. Create a star shaped tree with one internal node $c$ and $|S|$ leaves
       corresponding to the taxa $S$.
    2. for $|S|-3$ steps:
           (a) Select two neighbors $i$ and $j$ of $c$ such that $q_{ij}$ is minimized
               using Equation 5.
           (b) Create a new node $z$ and connect it to $c$.
           (c) Disconnect $i$ and $j$ from $c$ and re-connect them to $z$.
           (d) Set the edge weights of $(i,z)$ and $(j,z)$ with Equations 6 and 7.
           (e) For every neighbor $k$ of $c$, except $i$ and $j$, set the distance
               $d_{zk}$ with Equation 8.
           (f) Update the distance matrix by removing $i$ and $j$ and adding $z$.
    There are now three neighbors $x_1, x_2, x_3$ of $c$.
    3. Set the edge weights $(x_i, c)$ with Equation 9.
    4. Return the tree with its edge weights.
```

$$d_{zk} = \frac{1}{2}\left(d_{ik} + d_{jk} - d_{ij}\right). \tag{8}$$

After the last iteration we have a tree where the node c has exactly three neighbors x_1, x_2, x_3. This is indeed the final tree, however, in order to conclude the construction of the phylogenetic tree we need to assign weights to the last edges (x_1, c), (x_2, c) and (x_3, c). Let w_{1c}, w_{2c} and w_{3c} denote these weights, respectively. The weights are calculated as follows:

$$\begin{aligned} w_{1c} &= \frac{1}{2}(d_{12} + d_{13} - d_{23}), \quad (9) \\ w_{2c} &= \frac{1}{2}(d_{12} + d_{23} - d_{13}), \\ w_{3c} &= \frac{1}{2}(d_{13} + d_{23} - d_{12}). \end{aligned}$$

For completeness we describe the algorithm with pseudo code in Algorithm 1.

2.6 The DNCTREE Algorithm

The DNCTREE algorithm was developed by Lars Arvestad [1]. It is a heuristic that utilizes NEIGHBOR-JOINING as a sub-routine, offering considerable improvements with the time complexity being in the worst case quadratic, but initial experiments in [1] indicate that in many cases it seems to scale $O(n \lg n)$. In short, DNCTREE randomly selects three sequences from S and partitions the remaining sequences determined by a quartet test outlined below. Arvestad has also shown in [1] that Atteson's convergence radius holds for DNCTREE.

As input the DNCTREE algorithm will take a set of n aligned sequences S and a distance function $\mathcal{D}_S$. Let k be a base case size specified as a parameter, and an outline is

```
Algorithm 2 DNCTREE($S, D_S$)
    1. If $|S| \leq k$ :
        (a) Let $M_S$ be the distance matrix for $S$ using $\mathcal{D}_S$.
        (b) Return NEIGHBOR-JOINING($M_S$).
    2. Randomly select three sequences: $x, y, z \in S$.
    3. Use $\mathcal{D}_v$ to estimate  $d_{xy}, d_{xz}$ and $d_{yz}$.
    4. Introduce a ‘‘center’’ vertex $c$, set $d_{xc}, d_{yc}, d_{zc}$ optimally.
    5. Create subsets $S_x = \{x, c\}, S_y = \{y, c\}, S_z = \{z, c\}$.
    6. For $s \in S \backslash \{x, y, z\}$ :
        (a)For $w \in \{x, y, z\} : d_{sw} = D_S(s, w)$
        (b)Compute a quartet test on $\{x, y, z, s\}$.
        (c)Place $s$ accordingly: in $S_x, Sy$ or $S_z$.
        (d) If $s \in S_x$, set $d_{sc} =$
        $(d_{sy} + d_{sz} - d_{yz})/2$ or adjust accordingly if $s \in S_y$ or $s \in S_z$.
    7. For $i \in \{x, y, z\} : T_i =$ DNCTREE($_i, D_v$).
    8. Return $T = T_x \cup T_y \cup T_z$.
```

provided in Algorithm 2.

Though following the outline provided for DNCTREE seems very simple, some details regarding the specific steps may be helpful to the reader.

Regarding steps 4 and 5, after the three sequences $x, y, z \in S$ are selected at random, we let c be a new node representing a center vertex for x, y, z and introduce edges $\{c, x\}, \{c, y\}$ and $\{c, z\}$.The corresponding edge weights for d_{xc}, d_{yc} and d_{zc} must then be calculated “optimally” by setting them such that they satisfy

$$d_{xc} + d_{yc} = d_{xy},$$

$$d_{xc} + d_{zc} = d_{xz},$$

$$d_{yc} + d_{zc} = d_{yz}.$$

The quartet test mentioned in step 6b is using $\mathcal{D}_S$ to calculate all pairwise distances on the set $\{s, x, y, z\}$ for some $s \in S \backslash \{x, y, z\}$ forming the matrix M_σ and then constructing a tree we will call here T_σ by using NEIGHBOR-JOINING(M_σ). We will then use T_σ to determine which subset s belongs to by seeing which leaf vertex a was selected as the neighbor for s, and then assign s to subset S_a with $a \in \{x, y, z\}$.

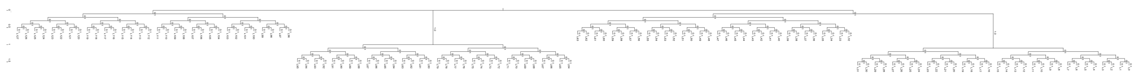

Figure 2: Simple Symmetric Additive Tree with 128 leaves.

2.7 Generating Data used in Experiments

Three experiments were conducted with DNCTREE-K using simulated data. For the first we simulated distances (but not sequences) by constructing a symmetric additive tree with 128 leaves, to which we then added a specified error term to. Our purpose with this was to see how the accuracy of our inferred trees was affected as the errors in our distance estimates (i.e. distance function) increased. The remaining two experiments used simulated sequences, and examined first algorithmic performance (i.e. tree accuracy) as the MSA width varied, and then algorithmic performance (i.e. accuracy and work) on trees with varying leaf-set sizes. See details below as to how the data was generated, and see Section 4 for details on how the experiments were run and their corresponding results.

2.7.1 Creating Noisy Data

The first experiment was designed to investigate how tree inference accuracy is affected as $|\mathcal{D}_S - D_{\hat{T}}|_\infty$ varies. To do so, we will construct data that will be just within, and then also to varying degrees outside the safety radius proven to apply for NEIGHBOR-JOINING and DNCTREE-K. We can control this as described below.

We construct a symmetric additive tree of 128 leaves shown in Figure 2. We now set the weights of the two innermost edges to 1.0, and all remaining edges are given weight 0.1. This will be our $\hat{T}$ and we will refer to the distances in $\hat{T}$ defined by Equation 1 as the additive distances. Given a parameter E, we will now create a $\mathcal{D}_S$ by adding noise, or an error term sampled from a uniform distribution on $[-E, E]$ to the additive distances. This new $\mathcal{D}_S$ will then be used as input to our algorithms DNCTREE, DNCTREE-K and NEIGHBOR-JOINING for this experiment. As can be seen in Section 4.1 we will let E vary in different rounds of the experiment from 0.05 up to a maximum value of 0.2. Note that given that our shortest branch in $\hat{T}$ is 0.1, this means that when $E = 0.05$, if Atteson's convergence radius holds for the algorithm being tested, then $\hat{T}$ should be returned.

2.7.2 Creating Simulated Sequences

For the second test (results shown in Section 4.2),we simulated sequences using the tool AliSim [24], which simulates biologically realistic sequence alignments and is available with IQ-Tree version 2.2.2.6 [17]. In order to perform the simulation, we first generated random trees with a specified number of taxa under the Yule-Harding model with the tree branch lengths initially randomly assigned following an exp(10) distribution. An evolution of these sequences was then simulated following the WAG model [37] Two tests were conducted with simulated sequences, the first looked at tree accuracy as the width of our MSA object varied from 200 up to 1000 columns (i.e. taxa sequence length varied from

200 up to 1000), while keeping the taxa set size to 200. The second, larger experiment, kept the MSA width constant at 500 columns (i.e. each taxa sequence simulated has length 500), while the size of our taxa set simulated varied between 100 up to 3000. In order to produce statistically robust results we replicated each parameter combination 100 times in both tests.

2.8 Putting it All Together: Snakemake

As one may perhaps gather both from Section 2.7 and Section 4, simulating the data required and running the experiments not only generated a large number of files, but running the experiments for some parameter combinations being tested required several days given the limited computational power we were restricted to. The process of running these experiments and collecting the most interesting results in relevant plots would have been a tedious task to do manually, and it was therefore of paramount importance to somehow automate the process. This was done with Snakemake [26], a text-based workflow management system particularly popular in the field of bioinformatics.

The snakemake workflow is defined by rules composed into smaller steps specifying the inputs, outputs, and shell commands. Snakemake is able to automatically determine dependencies between the rules by matching file names. The rules are always stored in a “snakefile” written in the Snakemake language which is an extension of Python with added syntactic structure that enables the rules and additional controls needed for the workflow to be defined.

Snakefiles that Lars Arvestad had produced previously for testing DNCTREE were modified and used to conduct testing on DNCTREE-K.

3 The DNCTREE-K Algorithm

Just as with DNCTREE the input for DNCTREE-K is a taxa sequence set S and a corresponding distance function $\mathcal{D}_S$. Crucially, just as in DNCTREE we will only calculate the distance values between a pair of sequences when we actually need them. All values that we have calculated previously will also be cached so that we may look them up as needed. Though we will not be using matrices in the same way as in NEIGHBOR-JOINING we will still present here the estimated distance between a vertex x and a vertex y as d_{xy}, and in the case that this value has not been calculated yet, assume that we will and can do so when we need to, storing it as we go along for reference later. Details of how this is done in the implementation will be given in Section 3.2.

3.1 DNCTREE-K Walk-Through

The new algorithm DNCTREE-K attempts to address the accuracy issues found in DNCTREE by increasing the quality of the partitions. Similar to DNCTREE, DNCTREE-K is a recursive algorithm where we have the parameter k determining the base case size at which NEIGHBOR-JOINING will be run. That is, if the number of sequences is k or less we run NEIGHBOR-JOINING, otherwise we run DNCTREE-K recursively. Unlike DNCTREE, instead of randomly selecting just three sequences from the sequence set S we randomly select $r \geq 3$ sequences. Let $S_{core} \subseteq S$ represent these sequences, where $|S_{core}| = r$ is referred to as the *core size.*[5]

For the set S_{core} we construct a distance matrix $M_{S_{core}}$ and infer a tree T_{core} with NEIGHBOR-JOINING$(M_{S_{core}})$. For tree T_{core} we identify a center vertex (see definition of center vertex in Section 2.4) and partition T_{core} into three sub-trees T_1, T_2, T_3 by splitting T_{core} at the center vertex. Let S_1, S_2, S_3 denote the leaves of T_1, T_2, T_3, respectively. Hence $S_1 \cup S_2 \cup S_3 = S_{core}$. We refer to the three sets S_1, S_2, S_3 as *clades*.

The DNCTREE-K algorithm proceeds by recursively calling itself on the three smaller sets defined by the clades. Before recursively building trees based on the clades we need to consider the sequences in $S \setminus S_{core}$ that have not yet been processed. The idea is to go though all sequences in $S \setminus S_{core}$ and allocate each one to exactly one of the sets S_1, S_2, S_3. Once all sequences have been allocated to its set S_i we are more or less ready to perform the recursive calls. The only part that remains is to also include the center vertex in each of the three sets. The reason for including the center vertex is that we need a distinguished vertex on which the three trees returned by the recursive calls will be joined on.

Before getting into the details of how the vertices of $S \setminus S_{core}$ are allocated to the clades, let us introduce a bit of notation. Let $S'_i \subseteq S \setminus S_{core}$ be the set of sequences that are allocated to the ith clade S_i. Let c denote the center vertex of T_{core}. If there are two candidates for the center vertex we choose one arbitrarily. The three recursive calls of DNCTREE-K will be on the sets

$$S_i \cup S'_i \cup \{c\}$$

for $i = 1, 2, 3$. Here we are facing a potential issue: the vertex c is not a sequence of S, hence there is no well defined distance between c and a sequence of S. To mitigate this issue we will regard c as a new, or pseudo sequence and find appropriate distances between c and other sequences of S. Once we have established these distances we can call DNCTREE-K on the three sets. Subsequently, the three returned trees will all contain a leaf that represents the sequence c. We now join the trees by merging them on the leaf c, resulting in a tree with one leaf for each sequence in S, where each internal vertex has degree three. This is the tree returned by DNCTREE-K. Next we explain how sequences are allocated to the clades, as well as how the distance from c to a sequence in S is calculated.

[5]In the testing conducted as part of this project, we will let the core size r range from 50 to 200.

3.1.1 Allocating Sequences to Clades

In order to allocate a sequence $s \in S \setminus S_{core}$ to one of the clades S_1, S_2, S_3 we will identify its "friend" $f \in S_{core}$. If f belongs to the clade S_i then s will be allocated to S_i. That is, a sequence is allocated to the same clade as its friend. The friend of s is identified by using the neighbor joining selection function and selecting the sequence that minimizes the q-value. The set of sequences we consider for this step is therefore $S_{core} \cup \{s\}$. Suppose $m = r + 1$ denotes the size of this set. Using the standard formula for the neighbor joining selection function, we determine the friend f from

$$\arg\min_{f \in S_{core}} q(s, f) = (m-2)d_{sf} - \sum_{x=1}^{m} d_{sx} - \sum_{x=1}^{m} d_{fx}. \tag{10}$$

We may rewrite the formula for $q(s, f)$ as follows:

$$\begin{aligned} q(s,f) &= (m-2)d_{sf} - \sum_{x=1}^{m} d_{sx} - \sum_{x=1}^{m} d_{fx} \\ &= ((r+1)-2)d_{sf} - \sum_{x \in S_{core} \cup \{s\}} d_{sx} - \sum_{x \in S_{core} \cup \{s\}} d_{fx} \\ &= (r-1)d_{sf} - \sum_{x \in S_{core}} d_{sx} - d_{ss} - \sum_{x \in S_{core}} d_{fx} - d_{fs} \\ &= (r-2)d_{sf} - \sum_{x \in S_{core}} d_{sx} - \sum_{x \in S_{core}} d_{fx} \end{aligned} \tag{11}$$

since $d_{ss} = 0$ and $d_{fs} = d_{sf}$. Finding the friend f of s is now straightforward. Given a sequence s we utilize Equation 11 by iterating over the sequences of S_{core} and selecting the f that minimizes $q(s, f)$. For the sake of implementing this step efficiently, we calculate the values of

$$\sum_{x \in S_{core}} d_{yx}$$

for each $y \in S$ once and reuse these values where needed.

3.1.2 Calculating the Distance Between the Center Vertex and a Sequence

Calling DNCTREE-K recursively requires a well defined distance between the center vertex c and sequences of S. For a sequence $s \in S_{core}$ we set the distance d_{sc} between s and c to be the sum of the edge weights on the path from s to c in the tree T_{core}. Since T_{core} was obtained by running neighbor joining on S_{core}, the edge weights are readily available.

For a vertex $s \in S \setminus S_{core}$ it is less obvious what the distance d_{sc} should be. Given a vertex $w \in S_{core}$, where w belongs to a clade different from the clade that s belongs, an intuitive estimate of the distance d_{sc}, parameterized by w, would be

$$d_{sw} - d_{wc}.$$

To see this we may think of s sitting in one of the sub-trees T_1, T_2, T_3 and w in another sub-tree. A path from s to w must therefore go via the center vertex c. Thus, the distance from s to w may be split into two parts:

$$d_{sw} = d_{sc} + d_{cw}.$$

Since we already have an estimate for the distance d_{wc} we can solve for d_{sc}. This particular estimate of d_{sc} was parameterized by a sequence w. By considering all possible sequences w we get many estimates of d_{sc}. Taking the average of all these estimates gives us a final estimate of d_{sc}. For example, suppose that $s \in T_1$. For each $w \in T_2 \cup T_3$ we have an estimate. The distance d_{sc} is then the arithmetic mean of these $|S_2| + |S_3|$ estimates. Namely,

$$\begin{aligned} d_{sc} &= \frac{1}{|S_2| + |S_3|} \sum_{w \in S_2 \cup S_3} (d_{sw} - d_{wc}) \\ &= \frac{1}{|S_2| + |S_3|} \left(\sum_{w \in S_2 \cup S_3} d_{sw} - \sum_{w \in S_2} d_{wc} - \sum_{w \in S_3} d_{wc} \right). \end{aligned} \tag{12}$$

The last step, breaking the summation into three sums, is illustrated here for the purpose of matching the details of the implementation of DNCTREE-K where we calculate the three sums separately. Note that the last two sums are independent of s. The estimates d_{sc} for s in T_2 and T_3 are calculated similarly. This concludes the description of DNCTREE-K. A summary in pseudocode is given in Algorithm 3.

3.1.3 Complexity Analysis of DNCTREE-K

We analyze the runtime of DNCTREE-K for two cases: on data where the allocation of taxa into the three clades is as unbalanced as possible, and on data where the three clades are of the same size. In the former scenario we will see that the runtime is $O(n^2)$, and in the latter case the runtime is $O(n \log n)$. We start by analyzing the first scenario.

In the most extreme situation T_{core} consists of two clades with only one taxa each and one clade containing the other $k - 2$ taxa. This scenario is technically not possible since the center vertex would not sit this close to two leaves, but for the sake of analyzing the time complexity we may assume that two clades contain just one taxa each. Further suppose that all taxa that are not part of T_{core} are allocated to the largest clade. The recursive step of the algorithm would then involve two subproblems of size 2 (a leaf and the center vertex) and one subproblem of size $n - 1$. Let $T(n)$ be the runtime of the algorithm on n taxa. Hence

$$T(n) = 2T(2) + T(n-1) + k^3 + rkn,$$

where the term k^3 is the runtime of NEIGHBOR-JOINING on k taxa and r is a constant. The last term rkn covers the part of the algorithm that calculates the center vertex, distances

```
Algorithm 3 DNCTREE-K
    Input: A set of sequences $S$ and a distance function $\mathcal{D}_S$.
           Base case size $k$ and core size $r$.

    1. If $|S| \le k$ :
        (a) Compute $M_S$ using $\mathcal{D}_S$ on S.
        (b) Return NEIGHBOR-JOINING($M_S$).
    2. Form $S_{core}$ by randomly selecting $r$ sequences from $S$.
    3. Compute $M_{S_{core}}$ using $\mathcal{D}_S$ on $S_{core}$.
    4. Let $T_{core}$ = NEIGHBOR-JOINING($M_{S_{core}}$).
    5. Identify a center vertex $c$ in $T_{core}$.
    7. Use $c$ to partition $T_{core}$ into sub-trees $T_1, T_2, T_3$. Let $S_1, S_2, S_3$ denote
       the sets of sequences (clades) of the three trees, respectively.
    Sequences $s \in S \backslash S_{core}$ will be allocated to one of the three
       sets $S'_1, S'_2, S'_3$ as follows:
    8. For each $s \in S \backslash S_{core}$:
        (a) Identify a "friend" $f \in S_{core}$ using Equation 10.
        (b) If $f \in S_i$ then assign $s$ to $S'_i$.
    9. For each $s \in S_{core}$, let $d_{sc}$ be the distance induced by $T_{core}$
       (see Section 3.1.2).
    10. For each $s \in S \backslash S_{core}$, calculate $d_{sc}$ according to Equation 12.
    11. For $i = 1, 2, 3$, recursively call DNCTREE-K with the sequence
        sets $S_i \cup S'_i \cup \{c\}$ and the same values of $k$ and $r$.
        The distance function is augmented with the estimated distances $d_{sc}$.
    12. Return the tree obtained by connecting the three trees on $c$.
```

and allocates taxa to the three clades. It follows immediately that

$$T(n) \le (2T(2) + k^3)n + rkn^2,$$

hence the time complexity is $O(n^2)$ if we keep k fixed. Thus, if the subproblems in each step of the recursion are as unbalanced as in this scenario we would have quadratic runtime of DNCTREE-K.

Suppose next that the subproblems are of equal size. That is, on n taxa the three subcases are of size $n/3$ each (ignoring the center vertex). The runtime $T(n)$ is therefore

$$T(n) = 3T\left(\frac{n}{3}\right) + k^3 + rkn.$$

To solve this equation we use the Master theorem. By identifying the applicable case of the Master theorem we conclude that the runtime $T(n)$ is $O(n \log n)$.

3.2 Implementation of DNCTREE-K in Python

The DNCTREE-K algorithm has been implemented in Python closely following the steps outlined in Section 3.1. In this section we will highlight some of the details of the implementation. The code for DNCTREE-K is an addition to the extensive Python package by Lars Arvestad implementing the DNCTREE algorithm. The package is publicly available at `https://pypi.org/project/dnctree/` and contains plenty of features and auxiliary functions for handling sequences, distances and running the algorithm on various data. The package also contains an implementation of neighbor joining.

The contribution by this project to the DNCTREE package is an implementation of the DNCTREE-K algorithm. The code has been added as a single module that utilizes functions and classes from the implementation of DNCTREE. We also provide a notebook for unit testing some of the functionality of the implementation of DNCTREE-K. Efforts have been made to include succinct, yet relevant comments, as well as informative docstrings in the functions. The implementation of DNCTREE-K can be divided into the following parts.

- A slightly modified version of the implementation of NEIGHBOR-JOINING. The implementation used in the DNCTREE package does not include edge weights for the tree returned by the NEIGHBOR-JOINING function. Since we need edge weights when calculating the distance from a sequence to the center vertex in T_{core} we had to augment the code for NEIGHBOR-JOINING.
- Code for calculating the center vertex of a tree.
- Code for splitting the tree T_{core} at the center vertex c and extracting the corresponding clades, as well as calculating the distances d_{sc} for $s \in S_{core}$.
- The DNCTREE-K algorithm, making use of above functions. The implementation follows tightly the procedure of Algorithm 3. As mentioned previously in Section 3.1 we directly utilize Equations 11 and 12.
- Unit tests of the new code.

The extension of NEIGHBOR-JOINING is relatively straightforward. Next we describe some of the other parts of the implementation.

3.3 Calculating the Center Vertex

We use the `Tree` class of the DNCTREE package to represent trees. The internals of this class uses a dictionary to map a vertex v to a list containing the neighbors of v. Calculating the center vertex, or center vertices, is done through a helper function that returns the length of a longest path from a vertex v to a leaf (via a specific edge starting at v). This function uses recursion combined with memoization; a dictionary containing path lengths is passed with each recursive call in order to avoid calculating the same quantity more than once. In the case of two candidates for the center vertex, the code returns the one that happens to be encountered first.

3.4 Obtaining Clades and Distances to the Center Vertex

Here we split a tree at the center vertex c and return three dictionaries: a mapping of a leaf to its clade (a number in $\{0, 1, 2\}$), a mapping from a clade number to the list of leaves in the clade, as well as a mapping from each taxa s (leaf) to the distance d_{sc}. The distances d_{sc} are calculated recursively, starting from c and branching out towards the leaves.

In order keep track of distances, in general, we utilize the `PartialDistanceMatrix` class of the DNCTREE package. This class uses dictionaries to store distances between taxa, or any vertices for that matter. When asking the `PartialDistanceMatrix` object for the distance between two taxa x and y, the distance will be returned immediately if it has already been calculated. If the distance has not yet been calculated, an appropriate distance function will be called (which depends on the input sequences). The distance is then cached for fast retrieval. The `PartialDistanceMatrix` object is central to the code and is passed between functions. This means that the number of cached distances is constantly increasing. In particular, the `PartialDistanceMatrix` object is included in the NEIGHBOR-JOINING call and is responsible for storing the edge weights. As new taxa, or rather pseudotaxa or vertices are added, the `PartialDistanceMatrix` object ensures that each new entry is given a unique identifier. Thus, the `PartialDistanceMatrix` object will contain the edge weight between two vertices x and y of T_{core} long after T_{core} has being discarded and served its purpose. The distances d_{sc} that we calculate are added to the `PartialDistanceMatrix` object. When we call DNCTREE-K recursively with the center vertex c as part of the taxa, the distances d_{sc} are therefore available.

3.5 Unit tests

We provide simple unit tests for the implemented functions. The input is a set of eight taxa, `T_0`,...,`T_7`, with handcrafted pairwise distances as well as a complete tree with edge weights. The edge weights are given by

```
taxa_distances = [[  0, 20, 50, 50, 100, 60, 50, 60],
                  [ 20,  0, 50, 50,  80, 40, 70, 40],
                  [ 50, 50,  0, 40,  50, 10, 80, 30],
                  [ 50, 50, 40,  0,  50, 50, 40, 50],
                  [100, 80, 50, 50,   0, 40, 50, 40],
                  [ 60, 40, 10, 50,  40,  0, 90, 20],
                  [ 50, 70, 80, 40,  50, 90,  0, 90],
                  [ 60, 40, 30, 50,  40, 20, 90,  0]]
```

The tree is the one illustrated in Figure 1. The diagram was generated with `https://csacademy.com/app/graph_editor`, which has been an invaluable website for quickly drawing trees during the implementation of DNCTREE-K. From the distances above we see that the distance between `T_0` and `T_4` is 100, but the sum of the edge weights between the leaves `T_0` and `T_4` of the tree is only 71. Using the distance matrix above and the edge weights of the tree, we run several of our functions and compare the output with the manually expected output.

4 Evaluating DNCTREE-K

We present here an oversight of the results obtained through initial experiments designed to examine the performance of DNCTREE-K when working with simulated data. In order to exhibit a comparison between DNCTREE-K and DNCTREE-K, we have presented graphs side-by-side with the results of DNCTREE-K consistently on the left-hand side, and DNCTREE on the right. All results have been plotted using seaborn [38]. The K value noted in all graphs represents in the graphs with DNCTREE-K both base case size and core size throughout all tests (i.e. base case size = core size = K). In the case of DNCTREE, K represents the base case size. In order to provide a measure on tree similarity, we use either the relative RF distance and/or TMD (see Section 2.3). Please see Section 2.7 for details on how the data for the different experiments was actually simulated.

4.1 Experimenting with Noisy Distance Data

We have described in section 2 how we constructed $\hat{T}$ (with $|S| = 128$) shown in Figure 2. For this experiment, we will let the parameter E vary from 0.05 up until 0.2, and let the value of K for both DNCTREE-K and DNCTREE vary from just 5 up to 35. Again, K represents the base case size for DNCTREE and both the core size and base case size for DNCTREE-K The results on the same data achieved with running NEIGHBOR-JOINING are also presented in the same plot as NJ. We compare the inferred trees to the original topology of $\hat{T}$ using the relative RF distance in Figure 3 and using the more robust TMD in Figure 4.

Both Figures are presented as box plots indicating quartiles, and with the outliers removed. These outliers were determined by seaborn using a method that is a function of the inter-quartile range. Given that the shortest branch length in $\hat{T}$ is 0.1, we know that if Atteson's holds for the algorithm, when $E \leq 0.05$, then the true tree $\hat{T}$ should be returned (i.e. both the RF-distance and TMD between $\hat{T}$ and an inferred tree should be 0). We can see indeed that when $E = 0.05$,DNCTREE returns $\hat{T}$ for all base case sizes (K) tested, and NEIGHBOR-JOINING returns $\hat{T}$. However, we see that DNCTREE-K is not returning $\hat{T}$ consistently in any case, though we are more likely to return $\hat{T}$ as K increases. We can therefore conclude that, given that we have correctly implemented DNCTREE-K, Atteson's convergence radius does *not* hold for DNCTREE-K. However, what is interesting to note, however, is that as E increases, DNCTREE-K demonstrates a level of tree accuracy very much on par with NEIGHBOR-JOINING when $K = 25$ and $K = 35$. This is confirmed by both plots showing the RF distance, and also the more robust TMD. The original DNCTREE, however, suffers significantly, with accuracy deteriorating rapidly compared to NEIGHBOR-JOINING (and DNCTREE-K) as E increases.

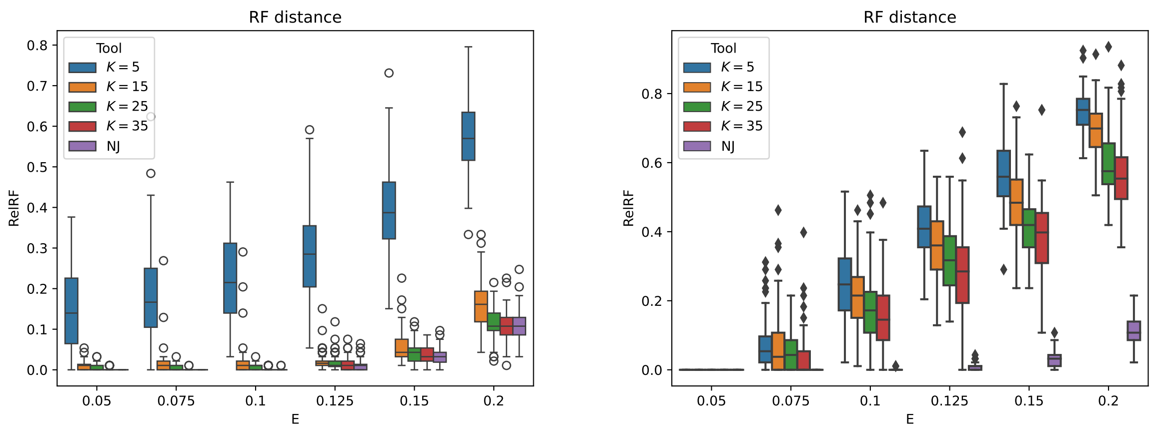

Figure 3: Testing on data generated from simple trees (see Section 4.1) with an added and varied error term E and using the popular but easily saturated relative RF distance for tree comparison. Note that DNCTREE-K is on the left hand side, and DNCTREE on the right.

4.2 Experimenting with Varying MSA Width

In the second test, we looked at how the accuracy of our inferred trees depended on the length of our taxa sequences in S. For this experiment we simulated sequences of varying length (from 200 until 1000), using the process described in Section 2.7.2 We then vary base case and (for DNCTREE-K) core size and compare our inferred trees to the relevant $\hat{T}$ using just the relative RF distance this time. For each parameter combination we made 100 replicates, again presenting the results using the box plots in Figure 5. Note the plot with DNCTREE-K had issues with a bug resulting in consistent abnormal coloring, but given the clarity of results for all values of K tested we nonetheless have included it. In general, as can be expected, our inferred trees have greater accuracy when based on sequences that are longer (i.e. we have a greater MSA width). We can also see that with DNCTREE-K for *all* values of K, the inferred trees are strikingly more accurate than those inferred with DNCTREE. In fact, the results of the experiment indicates that for all values of K we achieve a level of tree inference accuracy on par with NEIGHBOR-JOINING.

4.3 Experimenting Varying Sequence Set Sizes

In the final experiment, we once again simulated sequences following the process described in Section 2.7.2. This time, however, we kept the MSA width consistently at 500, but let the leaf size range from 100 up to a maximum size of 3000. For this experiment we were interested not noly in seeing how our inferred tree accuracy would be for larger sequence set sizes (and varying base case and core sizes) but also in seeing how the number of pair-wise distances calculated would scale as $|S|$and K increased. Again, just as with the previous experiments, we let K represent the base case and core size for DNCTREE-K and the base case size for DNCTREE. Results of the experiment are presented in Figure 6 which

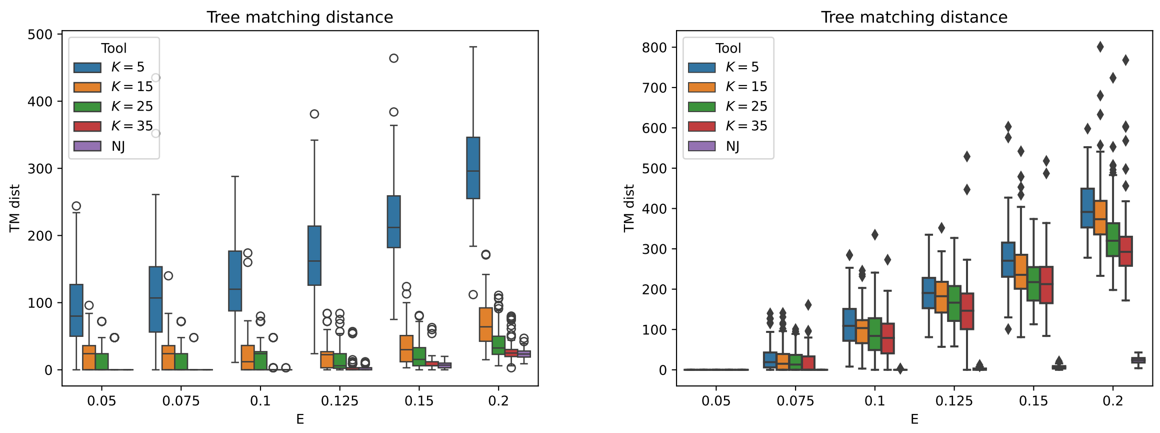

Figure 4: Testing on data generated from simple trees (see Section 4.1) with an added and varied error term E but this time using the more robust TMD to compare our inferred trees to $\hat{T}$. Note that DNCTREE-K is on the left hand side, and DNCTREE on the right. Quartiles are indicated, and outliers have been removed.

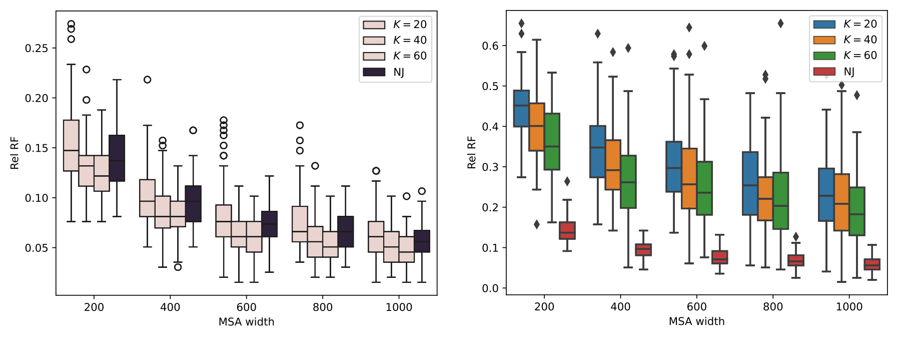

Figure 5: Testing on simulated sequences with MSA width varying from 200 up to 1000. Results from DNCTREE-K on left, and DNCTREE on the right. K indicates both base case size and core size for DNCTREE-K and the base case size for DNCTREE. Quartiles are indicated, and outliers have been removed.

uses the relative RF distance (denoted RF in the plot) as a measure of the accuracy of our inferred trees, and the number of pairwise distances calculated is presented in Figure 7. Once again, the trees inferred with the new DNCTREE-K are considerably more accurate compared to DNCTREE even at very low values of K. Interestingly, we can also note that whereas in order to achieve greater accuracy when using DNCTREE one requires a great value of K, our experiments do not necessarily indicate that this holds for DNCTREE-K. In fact, in Figure 6 we see that that we reach very good accuracy for all of our sequence set sizes with DNCTREE-K and a base case/core size of $K = 100$ indicating that this may be the value for which, given our data, we are able to create partitions of a higher quality leading to inferred trees of good accuracy. Unsurprisingly, the number of pairwise distances calculated is considerably higher for DNCTREE-K, yet as we showed in Section 3.1.3 we have the same asymptotic time complexity as the original DNCTREE, meaning we are still doing

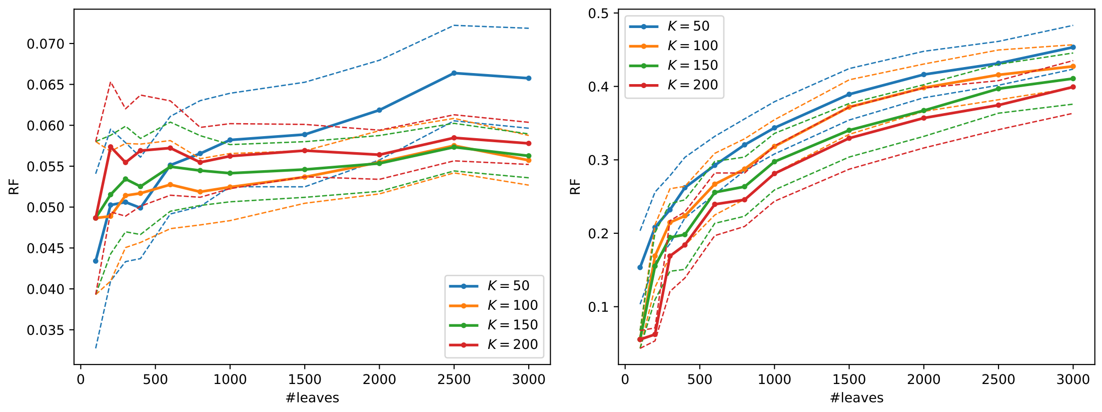

Figure 6: Testing on simulated sequences with MSA width of 500 and leaf-count varying from 100 up to 3000. Results from DNCTREE-K (shown on left) and DNCTREE (shown on right). Note the RF value is in fact the relative RF value described in Section 2.3. The mean relative RF distance for each combination of K and leaf count (represented as a specific color in key) is shown in bold, with the corresponding dotted lines indicating the standard error.

considerably less work than required for NEIGHBOR-JOINING, but, as demonstrated here, capable of inferring trees with comparable accuracy.

5 Discussion

In section 5.1 we will offer a general summary of the conclusions reached in this thesis regarding the performance of DNCTREE-K and how it compares to both NEIGHBOR-JOINING and DNCTREE. We will then conclude with Section 5.2 in which we give a brief overview of a few aspects and extensions of the ideas laid out here that may be interesting to investigate in the future.

5.1 General Summary of Results

The primary intention of this thesis was to develop, implement, and investigate a randomized divide-and-conquer algorithm for phylogenetic tree inference that was based on, but more accurate, than Arvestad's original DNCTREE. Considering the results of the experiments run here with simulated data, we have indeed done this. Not only did we show, given the data we have worked with, that DNCTREE-K infers trees with more accuracy than DNCTREE but, even at a relatively low K value (representing both base case and core size) our trees inferred with DNCTREE-K have a similar accuracy to those inferred with NEIGHBOR-JOINING even when working with noisy data, across varying MSA widths, and for larger data sets. And while the number of pairwise distances required is more than for DNCTREE, we have also shown in Section 3.1.3 that DNCTREE-K has the same asymptotic time complexity as DNCTREE.

Perhaps one of the most interesting results shown in Figure 6 is that we seem to

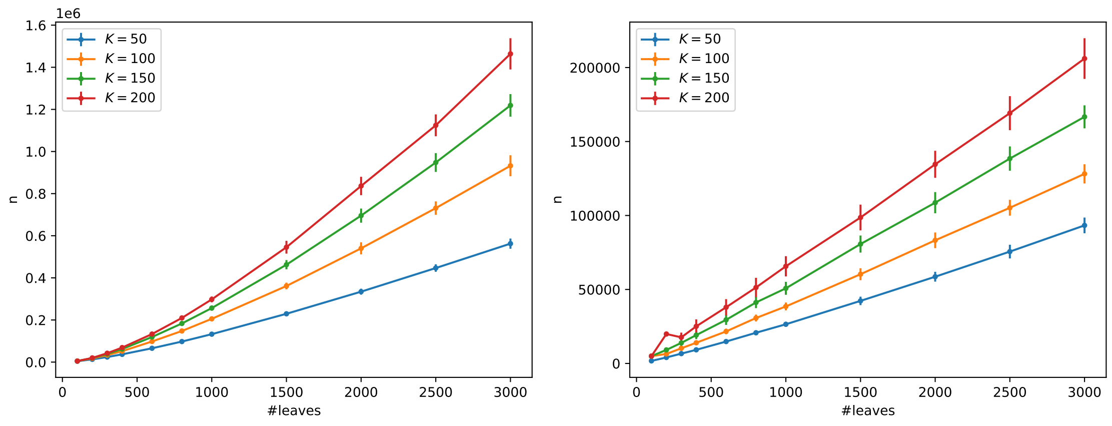

Figure 7: Side by side comparison of the number of pairwise distances computed during tree inferences over varying leaf sizes. Again, we have DNCTREE-K on the left with $K =$ base case size $=$ core size and DNCTREE on the left with $K =$ base case size. The vertical bars indicate standard error.

have extremely good performance with a base case and core size of $K = 100$ indicating that the mantra "bigger is better" which was certainly the case in the original DNCTREE, does not necessarily hold for DNCTREE-K. It may be that this is the core size value at which our partitioning method is capable of creating subsets of high quality, but to investigate this would require much more extensive testing.

Unfortunately, what also doesn't seem to hold for DNCTREE-K based on Figures 3 and 4, is the Atteson's convergence radius.[6] We see that when $E = 0.05$, $\hat{T}$ is recovered by NEIGHBOR-JOINING and DNCTREE (regardless of base case size K). Disregarding this, however, given the otherwise similar accuracy results when compared to NEIGHBOR-JOINING despite considering we are getting away with only a fraction of the work, we must conclude that DNCTREE-K in initial testing on simulated data seems to exhibit exciting potential to be a scalable alternative to NEIGHBOR-JOINING without sacrificing too much in terms of accuracy.

5.2 Continued Investigation

Undoubtedly, clever improvements can be introduced to the straight-forward Python implementation of DNCTREE-K that we present here.

It was mentioned in [15] that in the case that substitutions rates were higher, and more varied across lineages for a given taxa set, BIONJ (mentioned briefly in our introduction) had considerably more topological accuracy compared to NEIGHBOR-JOINING. Perhaps one could replace the standard NEIGHBOR-JOINING functions utilized in DNCTREE-K with those modified versions that take variance into account used in BIONJ and see how the performance of this modified DNCTREE-K compares to our original DNCTREE-K presented here.

[6]We assume, based on careful review, that we have implemented DNCTREE-K correctly.

It would also be of interest to conduct more extensive experiments than those run for this thesis. For example, a good starting point for a closer and more comprehensive comparison of NEIGHBOR-JOINING and DNCTREE-K could be to run NEIGHBOR-JOINING on the simulated data created for the experiment in Section 4.3. This would of course however require considerably more computational power than what was available during the writing of this thesis, or at the very least a considerably greater capacity for patience.

The fact that DNCTREE-K greatly reduces the number of pairwise distances calculated, just as DNCTREE does, would also mean that, as Arvestad suggested for DNCTREE in [1] we would be able to consider more computationally heavy but accurate distance functions. And, if one is more concerned with speeding up things, such as would be required for working with data sets of extreme size, perhaps it would be interesting to look into finding a way to modify and combine DNCTREE-K so that it instead works with alignment-free estimations such as the ones mentioned in [20] or [3]. An experiment similar to the one in Section 4.3 (but with even larger sequence set sizes) could then be used to compare this modification with a modified version of NEIGHBOR-JOINING also working with alignment-free estimations. Such a modification of NEIGHBOR-JOINING has already been initially studied in [20].

6 Availability

The source code for DNCTREE-K is available in Lars Arvestad's github repository and found in the branch AMY at https://github.com/arvestad/dnctree. Please note that this repository may be private.

7 Acknowledgments

Immense thanks is due to Lars Arvestad, not only for his patience and expert guidance during the development of the project, but also for the use of modules he developed for testing his original DNCTREE algorithm, the permission to reuse his plots showing results form his original testing with DNCTREE, as well as the allowance of hijacking his snakefiles originally written for testing.

References

[1] Arvestad, Lars. Scalable distance-based phylogeny inference using divide-and-conquer, October 16, 2023. DOI:10.1101/2023.10.11.561902

[2] Atteson, K. The Performance of Neighbor-Joining Methods of Phylogenetic Reconstruction . Algorithmica 25, 251–278 (1999). DOI: 10.1007/PL00008277

[3] Marcin Bogusz, Whelan, S. Phylogenetic Tree Estimation With and Without Alignment: New Distance Methods and Benchmarking, Systematic Biology, Volume 66, Issue 2, March 2017, Pages 218–231. DOI: 10.1093/sysbio/syw074

[4] Bryant, D. On the Uniqueness of the Selection Criterion in Neighbor-Joining. Journal of Classification 22, 3–15 (2005). DOI: 10.1007/s00357-005-0003-x

[5] Clausen, Philip T L C. Scaling neighbor joining to one million taxa with dynamic and heuristic neighbor joining, Bioinformatics, Volume 39, Issue 1, January 2023, btac774, https://doi.org/10.1093/bioinformatics/btac774

[6] Conant, G.C. and Wolfe, K.H. (2008) Turning a Hobby into a Job: How Duplicated Genes Find New Functions. Nature Reviews Genetics, 9, 938-950. DOI: 10.1038/nrg2482

[7] Chatzou M, Magis C, Chang JM, Kemena C, Bussotti G, Erb I, Notredame C. Multiple sequence alignment modeling: methods and applications. Brief Bioinform. 2016 Nov;17(6):1009-1023. DOI: 10.1093/bib/bbv099.

[8] da Fonseca, R.R.; Albrechtsen, A.; Espregueira Themudo, J.G.; Ramos-Madrigal, J.; Sibbesen, J.A.; Maretty, L.; Zepeda-Mendoza, M.L.; Campos, P.F.; Heller, R. and Pereira, R.J. (2016) Next-generation biology: Sequencing and data analysis approaches for non-model organisms. Marine Genomics, 6, pp.3-13. DOI: 10.1016/j.margen.2016.04.012

[9] Edgar, RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 2004 Mar 19;32(5):1792-7. DOI: 10.1093/nar/gkh340

[10] Efron B, Halloran E, Holmes S. Bootstrap confidence levels for phylogenetic trees. Proc Natl Acad Sci U S A. 1996 Nov 12;93(23):13429-34. DOI: 10.1073/pnas.93.23.13429

[11] Elias I, Lagergren J. Fast computation of distance estimators. BMC Bioinformatics. 2007 Mar 13;8:89. DOI: 10.1186/1471-2105-8-89

[12] Huerta-Cepas J, Serra F, Bork P. ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. Mol Biol Evol. 2016 Jun;33(6):1635-8. DOI: 10.1093/molbev/msw046

[13] Gascuel, O. and Steel, M. Neighbor-Joining Revealed, Molecular Biology and Evolution, Volume 23, Issue 11, November 2006, Pages 1997–2000, DOI: 10.1093/molbev/msl072

[14] Gascuel, O. and Steel, M. A 'stochastic safety radius' for distance-based tree reconstruction. *Algorithmica,* 74:1386-1403, 2016. DOI: 10.48550/arXiv.1411.4106

[15] Gascuel, O. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. Mol Biol Evol. 1997 Jul;14(7):685-95. DOI: 10.1093/oxfordjournals.molbev.a025808

[16] Guindon S, Gascuel O. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. Syst Biol. 2003 Oct;52(5):696-704. DOI: 10.1080/10635150390235520

[17] Minh BQ, Schmidt HA, Chernomor O, Schrempf D, Woodhams MD, von Haeseler A, Lanfear R. IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era. Mol Biol Evol. 2020 May 1;37(5):1530-1534. DOI: 10.1093/molbev/msaa015

[18] Kannan, Sampath K., Eugene L. Lawler, Tandy J. Warnow, Determining the Evolutionary Tree Using Experiments, Journal of Algorithms, Volume 21, Issue 1, 1996, Pages 26-50. DOI: 10.1006/jagm.1996.0035

[19] Khan MA, Elias I, Sjölund E, Nylander K, Guimera RV, Schobesberger R, Schmitzberger P, Lagergren J, Arvestad L. Fastphylo: fast tools for phylogenetics. BMC Bioinformatics. 2013 Nov 20;14:334. DOI: 10.1186/1471-2105-14-334

[20] Kolekar P, Kale M, Kulkarni-Kale U. Alignment-free distance measure based on return time distribution for sequence analysis: applications to clustering, molecular phylogeny and subtyping. Mol Phylogenet Evol. 2012 Nov;65(2):510-22. DOI: 10.1016/j.ympev.2012.07.003. Epub 2012 Jul 20. PMID: 22820020

[21] Shuying Li, Dennis K. Pearl and Hani Doss (2000) Phylogenetic Tree Construction Using Markov Chain Monte Carlo, Journal of the American Statistical Association, 95:450, 493-508, DOI: 10.1080/01621459.2000.10474227

[22] Lin, Yu, Fei, Hu Tang, Jijun and Moret, Bernard, M.E. Maximum Likelihood Phylogenetic Reconstruction from High-Resolution Whole-Genome Data and a Tree of 68 Eukaryotes. Biocomputing 2013: 285-296. DOI: 10.1142/9789814447973_0028

[23] Lin, Y., Rajan, V., Moret, B.M.E. (2011). A Metric for Phylogenetic Trees Based on Matching. In: Chen, J., Wang, J., Zelikovsky, A. (eds) Bioinformatics Research and Applications. ISBRA 2011. Lecture Notes in Computer Science(), vol 6674. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-21260-4_21

[24] Ly-Trong, Nhan, Suha Naser-Khdour, Robert Lanfear, Bui Quang Minh, AliSim: A Fast and Versatile Phylogenetic Sequence Simulator for the Genomic Era, Molecular Biology and Evolution, Volume 39, Issue 5, May 2022, msac092. DOI: 10.1093/molbev/msac092

[25] Mihaescu, R., Levy, D. & Pachter, L. Why Neighbor-Joining Works. Algorithmica 54, 1–24 (2009).DOI: 10.1007/s00453-007-9116-4

[26] Mölder F, Jablonski KP, Letcher B, Hall MB, Tomkins-Tinch CH, Sochat V, Forster J, Lee S, Twardziok SO, Kanitz A, Wilm A, Holtgrewe M, Rahmann S, Nahnsen S, Köster J. Sustainable data analysis with Snakemake. F1000Res. 2021 Jan 18;10:33. DOI: 10.12688/f1000research.29032.2

[27] Pardi, F., Gascuel, O. Distance-based methods in phylogenetics. Richard M. Kliman. Ency- clopedia of Evolutionary Biology, Elsevier, pp.458-465, 2016, 1st Edition, 978-0-12-800426-5. lirmm- 01386569

[28] Pardi, F., Guillemot, S. and Gascuel, O. Robustness of Phylogenetic Inference Based on Minimum Evolution. Bull. Math. Biol. 72, 1820–1839 (2010). DOI: 10.1007/s11538-010-9510-y

[29] Price, Morgan N., Paramvir S. Dehal, Adam P. Arkin, FastTree: Computing Large Minimum Evolution Trees with Profiles instead of a Distance Matrix, Molecular Biology and Evolution, Volume 26, Issue 7, July 2009, Pages 1641–1650, DOI: 10.1093/molbev/msp077

[30] Robinson, D. F. and Foulds, L.R. Comparison of phylogenetic trees. *Molecular Biology and Evolution*, 26(7):1641-50, 1981.

[31] Saitou, N, and Nei, M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol. 1987 Jul;4(4):406-25. DOI: 10.1093/oxfordjournals.molbev.a040454

[32] Steel, Mike and Penny, David. Distributions of Tree Comparison Metrics–Some New Results. June 1993 Systematic Biology 42(2):126-141. DOI:10.1093/sysbio/42.2.126

[33] Studier, J. A.; Keppler, K. J. (November 1988). "A note on the neighbor-joining algorithm of Saitou and Nei". Molecular Biology and Evolution. 5 (6): 729–31. DOI:10.1093/oxfordjournals.molbev.a040527

[34] Turakhia, Y., Thornlow, B., Hinrichs, A. et al. Pandemic-scale phylogenomics reveals the SARS-CoV-2 recombination landscape. Nature 609, 994–997 (2022). DOI: 10.1038/s41586-022-05189-9

[35] Van de Peer Y, Salemi M. Phylogenetic inference based on distance methods. In: Lemey P, Salemi M, Vandamme A-M, eds. The Phylogenetic Handbook: A Practical Approach to Phylogenetic Analysis and Hypothesis Testing. 2nd ed. Cambridge: Cambridge University Press; 2009:142-180. DOI:10.1017/CBO9780511819049.007

[36] Weiwen, James Barbetti, Thomas Wong, Bryan Thornlow, Russ Corbett-Detig, Yatish Turakhia, Robert Lanfear, Bui Quang Minh, DecentTree: scalable Neighbour-Joining for the genomic era, Bioinformatics, Volume 39, Issue 9, September 2023, btad536, DOI: 10.1093/bioinformatics/btad536

[37] Whelan, Simon and Goldman, Nick. A General Empirical Model of Protein Evolution Derived from Multiple Protein Families Using a Maximum-Likelihood Approach, Molecular Biology and Evolution, Volume 18, Issue 5, May 2001, Pages 691–699. DOI: 10.1093/oxfordjournals.molbev.a003851

[38] Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, DOI: 10.21105/joss.03021

[39] Xi, Jing & Xie, Jin & Yoshida, Ruriko & Forcey, Stefan. (2015). Stochastic safety radius on Neighbor-Joining method and Balanced Minimal Evolution on small trees. DOI: 10.48550/arXiv.1507.08734

[40] Yang Z, Rannala B. Molecular phylogenetics: principles and practice. Nat Rev Genet. 2012 Mar 28;13(5):303-14. DOI: 10.1038/nrg3186. PMID: 22456349

[41] Yang S, Yang H, Grisafi P, Sanchatjate S, Fink GR, Sun Q, Hua J. The BON/CPN gene family represses cell death and promotes cell growth in Arabidopsis. Plant J. 2006 Jan;45(2):166-79. DOI: 10.1111/j.1365-313X.2005.02585.x

Datalogi
www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm