Fast NeighborNet: Improving the Speed of the Neighbor-Net Phylogenetic Network Algorithm with Multithreading and a Relaxed Search Strategy

Jacob Porter

Department of Computer Science, Virginia Tech
Blacksburg, VA 24061 USA

Abstract

Fast Neighbor-Net is a command-line Java program that has fast implementations of the popular Neighbor-Net phylogenetic split network approach. This allows more efficiency in computationally intensive tasks such as larger scale data analysis and bootstrapping. The canonical search phase iteratively finds a pair of connected components that minimizes a distance function in $\Theta(n^3)$ time in the input taxa count. A relaxed search strategy has been implemented that has average-case time complexity of $\mathcal{O}(n^2 \log n)$ but with $\Theta(n^3)$ worst-case time complexity. This search strategy sacrifices some accuracy for speed. The original approach's implementation has been improved by using good programming practice. These improvements increased run-time performance by a constant factor of approximately 2 and reduced memory requirements by a constant factor of approximately 6. These search strategies allow multithreading to better use modern CPU hardware. PFAM data of 2000–30,000 taxa were used for testing performance. The canonical implementation with three threads improved average performance by approximately 2.1. The relaxed search has good quality, and the accuracy was tested on a mammal and a eukaryote data set. Kendall tau distance was used as a rough measure of topological similarity for the relaxed and canonical search strategies.

1. Introduction

The popular Neighbor-Net method of Bryant and Moulton is a phylogenetic split network algorithm with good speed and memory requirements [4] and proven statistical consistency [5]. The networks that it produces are planar with outer nodes labeled as taxa; thus, the networks are easy to draw and read. The network can represent recombination, hybridization, gene fusion, sequencing error, lateral gene transfer, and other signals of conflicting evolution that are not well represented by a tree [3]. These phenomena can be represented with a graph where some nodes have multiple parents unlike a tree where nodes have a single parent [11]. Neighbor-Net was inspired by the popular Neighbor-Joining algorithm [20].

Neighbor-Net takes as input a distance matrix and works by agglomerating connected components in an execution graph based on a distance function for each iteration. These terms will be defined in a later section. The original implementation is included in the SplitsTree4 Java program [10].

For n taxa, the Neighbor-Net algorithm has worst-case time complexity $\Theta(n^3)$ and space complexity $\Theta(n^2)$. Thus, it runs

in seconds or minutes for input sizes of only a few thousand taxa but takes hours if not days for input sizes of tens of thousands or hundreds of thousands of taxa. Protein families of this size exist in the PFAM database [9]. Phylogenies of this size are useful in understanding questions in comparative genomics such as tree of life projects [18].

Related Work. Neighbor-Joining and Neighbor-Net are very similar in their implementation details. There are several approaches for Neighbor-Joining [20] that have better average-case run-time. Methods that compute the same Neighbor-Joining tree faster than the canonical approach include QuickTree [13], QuickJoin [19], NINJA [24], and RapidNJ [23]. These methods have the same worst-case time complexity of $\Theta(n^3)$ but require more memory because of their specialized data structures. NINJA, for example, uses an array of priority queues and increases memory requirements by a constant factor. Clearcut [22], a faster implementation of the Relaxed Neighbor-Joining approach [7], uses a relaxed search heuristic to achieve $\mathcal{O}(n^2 \log n)$ average-case time complexity, but it is less accurate than the canonical approach. Another algorithm that computes phylogenetic split networks is split decomposition [1], but networks may not be planar and may be less resolved compared to Neighbor-Net [11]. Another popular split network is median networks [2].

Contributions. To the author's knowledge, there are no implementations of the Neighbor-Net approach that improve the run-time of the original implementation in SplitsTree in any way [10, 4]. Fast Neighbor-Net includes a fast search heuristic called relaxed search inspired by Relaxed Neighbor-Joining [7] and Clearcut [22], and it improves upon the run-time of the canonical approach. These implementations are faster in the following senses.

The run-time of the implementation of the canonical approach found in SplitsTree [10] was reduced by approximately half by using good programming practice. The memory requirements were reduced by about 9 for the quadratic part of memory. A relaxed search heuristic inspired by Clearcut for Neighbor-Joining was implemented. This gives $\mathcal{O}(n^2 \log n)$ average-case time complexity but $\Theta(n^3)$ worst-case time complexity. The relaxed search strategy reduces the accuracy of the algorithm. The canonical and relaxed search strategies employ multithreading. Computing the circular order faster makes analyzing larger data sizes possible and is useful for bootstrapping. The software “Noisy,” for example, computes circular orderings with Neighbor-Net to search for columns in

a multiple sequence alignment that are phylogenetically less useful [6].

2. The Canonical Neighbor-Net Algorithm

Before discussing the Neighbor-Net algorithm in detail, several terms will be defined and a high level description of the algorithm will be provided.

Definitions. The *taxa* set $\mathcal{X} = \{x_1, \dots, x_n\}$ is a set of strings over an alphabet Σ , where each taxon $x_i \in \mathcal{X}$ represents a group of organisms, such as a species, or an individual. For example, the string x_i could be the consensus sequence of a homologous protein, or each x_i could represent a different mammal such as cow, dog, gorilla, etc. In this example, the alphabet Σ are symbols representing amino acids. The cardinality of $\mathcal{X}$ is n .

A *split* $s = \{A, B\}$ is a partition of the taxa $\mathcal{X}$ into A and B so that $A \cup B = \mathcal{X}$ and $A \cap B = \emptyset$. The *split weight function* $f_S : S \rightarrow \mathbb{R}^+$ assigns a nonnegative real value to each split $s \in S$.

A metric $\delta_{\mathcal{X}}$ on $\mathcal{X}$ gives an estimate of the evolutionary distance between two taxa with respect to some evolutionary model. The metric space is $(\mathcal{X}, \delta_{\mathcal{X}})$. For biological sequence data, this metric can be computed with evolutionary models such as the Jukes-Cantor model [12], the Kimura parameter models [15], and other models [8]. The input to Neighbor-Net is the distance $\delta_{\mathcal{X}}(x, y)$ for each distinct $x, y \in \mathcal{X}$.

During the execution of Neighbor-Net, the algorithm produces a *circular ordering*, $\mathcal{Z} = \{z_1, z_2, \dots, z_n\}$, of the taxa. A *circular ordering* is a free cyclic permutation of the taxa.

High Level Description. The Neighbor-Net algorithm has the following input and output:

Input: A lower triangular matrix $\delta_{\mathcal{X}}^{(0)}$ of pairwise taxon distances.

Output: A circular ordering $\mathcal{Z}$ of the taxa, and a set of splits S together with a split weight function f_S .

The algorithm consists of two phases: a search and agglomeration phase and a split weight estimation phase. The result of the search and agglomeration phase is a *circular ordering* $\mathcal{Z}$ of the taxa. The ordering $\mathcal{Z} = \{z_1, z_2, \dots, z_n\}$ where each $z_i \in \mathcal{X}$ uniquely defines a *split set* S given by

$$S = \{ \{ \{z_p, z_{p+1}, \dots, z_q\}, \mathcal{X} - \{z_p, \dots, z_q\} \} \mid 1 < p \leq q \leq n \}$$

The splits together with their weights are the output of the split weight estimation phase of Neighbor-Net. This phase terminates the Neighbor-Net algorithm.

Data Structures. During the execution of the search and agglomeration phase of Neighbor-Net, the algorithm maintains a set of nodes and two sets of edges on these nodes. This induces two graphs $G_A^{(i)}$, an agglomeration history graph consisting of a directed acyclic graph, and $G_C^{(i)}$, an undirected graph of connected components. As the algorithm proceeds, connected

components from $G_C^{(i)}$ are combined in an operation called agglomeration, and the history of these operations is maintained in $G_A^{(i)}$ and an agglomeration history stack $A_H^{(i)}$. The purpose of this graph and the stack is to construct the circular ordering of the taxa. The two graphs are initially identical and consist of one node for each taxon input into Neighbor-Net and no edges. The agglomeration history stack is initially empty.

For each iteration, the graph $G_C^{(i)}$ consists of two kinds of connected components: couples and singles. Couples are two connected nodes, and singles are singleton unconnected nodes. Because of the simplicity of $G_C^{(i)}$, $G_C^{(i)}$ is stored as an array of node objects. The graph $G_A^{(i)}$ consists of node objects with references to sibling and children node objects.

The algorithm maintains an integer $m^{(i)}$ that gives the number of connected components in $G_C^{(i)}$. As the algorithm proceeds, the integer $m^{(i)}$ gets smaller as connected components are combined. The Neighbor-Net algorithm maintains the distance matrix $\delta_{\mathcal{X}}^{(i)}$ that gives the distances between nodes in $G_C^{(i)}$. It is initialized to the distances given as input to the algorithm. The distance $\delta_C^{(i)}$ gives the distances between connected components in $G_C^{(i)}$ and is computed as needed according to a formula given in the next section. Row sums for $\delta_C^{(i)}$ are maintained and stored as a field in a node representing a connected component.

An *agglomeration event* is an operation where two connected components are combined in $G_C^{(i)}$ and replaced by a new connected component. The algorithm chooses two connected components that have the least distance to each other according to a function that will be explained in the next section. The agglomeration history graph, $G_A^{(i)}$, maintains a history of agglomeration events performed on the connected components graph. Because connected components consist of either a single or a couple, there are three possible agglomeration events: single-to-single, single-to-couple, and couple-to-couple. These are shown in Figure 1, and each case will be explained.

For a single-to-single agglomeration, the algorithm defines a new edge between x and y in $G_C^{(i)}$, and a new directed edge from x , the lexicographically least node, to y in $G_A^{(i)}$. For a single-to-couple agglomeration, there is a single node x and two connected nodes y and z . In $G_C^{(i)}$, two new connected nodes u and v are created and x , y , and z are deleted. The node in the couple that is closest to x is found by a procedure described in the next section, and an edge from x to this node is created in the agglomeration history graph, $G_A^{(i)}$. In the agglomeration history graph, the two new nodes u and v are connected with an edge going from u to v . An edge is connected from node u to x and from u to the node closest to x . An edge is connected from node v to the node closest to x and to the node furthest from x . An edge goes from x to the node closest to x . The

Figure 1: The three different agglomeration events performed in the agglomeration phase of Neighbor-Net. (a) Two singles are merged. (b) There is only one possibility, so (c) a couple is produced. (d) A single and a couple are combined, and (e) there are two possibilities to consider (f) resulting in a couple with new nodes. (g) A couple and a couple are combined with four possibilities (not shown) to examine (h) that results in a couple with new nodes.

node u , the lexicographically least node for the two new nodes created, is added to the agglomeration history stack, A_H .

In the last case, a couple consisting of nodes w and x is agglomerated with another couple consisting of z and y . Without loss of generality, suppose that the two nodes that are the closest to each other are x and y , then two single-to-couple agglomeration events are performed. The first agglomeration event is done on w , x , and y to produce s and t and then another agglomeration event is done on s , t , and z to produce u and v .

After an agglomeration event, the distances between the nodes $\delta_{\mathcal{X}}^{(i+1)}$ and the row sums are updated as described next.

Computing Distances. The distance between connected components c_j and c_k in the connected components graph $G_C^{(i)}$ is the average distance between their constituent nodes. Thus, the connected component distance is given by

$$\delta_C^{(i)}(c_j, c_k) = \begin{cases} \frac{1}{|c_j||c_k|} \sum_{x \in c_j} \sum_{y \in c_k} \delta_{\mathcal{X}}^{(i)}(x, y), & c_j \neq c_k \\ 0, & c_j = c_k \end{cases}$$

For a connected component c_k in $G_C^{(i)}$, the row sum $r_{\delta_C}^{(i)}(c_k)$ is the sum of connected component distances between c_k and all the other connected components.

$$r_{\delta_C}^{(i)}(c_k) = \sum_{c_j \in G_C^{(i)}} \delta_C^{(i)}(c_k, c_j)$$

Let $m^{(i)}$ be the number of connected components in the graph $G_C^{(i)}$. The connected component pair c_k and c_j that minimizes the following function, called the Q -criterion, is chosen for agglomeration.

$$Q^{(i)}(c_j, c_k) = (m^{(i)} - 2) \delta_C^{(i)}(c_j, c_k) - r_{\delta_C}^{(i)}(c_j) - r_{\delta_C}^{(i)}(c_k)$$

The Q -criterion has this form so that it is linear, permutation equivariant, and statistically consistent [5]. In the canonical implementation, it is calculated with a doubly nested `for` loop over all existing connected components. This gives a quadratic time complexity for each iteration. It is this part that is the focus of some of this work's enhancements.

Now that two connected components c_j and c_k have been chosen, they must be agglomerated. If one of the connected components is a couple, the agglomeration history graph $G_A^{(i)}$ connects nodes with respect to nodes between connected components that have the least distance according to the $\hat{Q}$ -criterion that will be described now.

The value $\hat{m}^{(i)} = m^{(i)} + |c_j| + |c_k| - 2$ counts all the connected components in $G_C^{(i)}$ minus two plus all the nodes in components c_j and c_k . This is as if all the nodes in c_j and c_k are treated as singleton components. Treating nodes in c_k and nodes in c_j as if they were singleton nodes in the connected components graph, we wish to find the node $n_r \in c_k$ and the node $n_s \in c_j$ that minimizes the following

$$\begin{aligned} \hat{Q}^{(i)}(n_r, n_s) &= \hat{m}^{(i)} \delta_{\mathcal{X}}^{(i)}(n_r, n_s) \\ &- \sum_{c_k \neq n_r, c_k \in G_C^{(i)}} \delta_C^{(i)}(n_r, c_k) \\ &- \sum_{c_k \neq n_s, c_k \in G_C^{(i)}} \delta_C^{(i)}(n_s, c_k) \end{aligned}$$

When agglomeration occurs, the distance matrix $\delta_{\mathcal{X}}^{(i)}$ must be updated and reduced to represent fewer nodes and new distances between nodes. The updated distance, $\delta_{\mathcal{X}}^{(i+1)}$, is computed in the following manner. If a single-to-single agglomeration event occurred, $\delta_{\mathcal{X}}^{(i+1)} = \delta_{\mathcal{X}}^{(i)}$. Suppose a single-to-couple agglomeration event occurred where x and y were a couple and z was the node with the least distance from the single node z . These nodes are replaced by nodes u and v where the distance from these nodes to a node $a \in G_C^{(i+1)}$ is given by the following formulas.

$$\delta_{\mathcal{X}}^{(i+1)}(u, a) = (\alpha + \beta) \delta_{\mathcal{X}}^{(i)}(x, a) + \gamma \delta_{\mathcal{X}}^{(i)}(y, a)$$

$$\delta_{\mathcal{X}}^{(i+1)}(v, a) = \alpha \delta_{\mathcal{X}}^{(i)}(y, a) + (\beta + \gamma) \delta_{\mathcal{X}}^{(i)}(z, a)$$

$$\delta_{\mathcal{X}}^{(i+1)}(u, v) = \alpha \delta_{\mathcal{X}}^{(i)}(x, y) + \beta \delta_{\mathcal{X}}^{(i)}(x, z) + \gamma \delta_{\mathcal{X}}^{(i)}(y, z)$$

The parameters α, β, γ are constants such that $\alpha + \beta + \gamma = 1$. In the canonical implementation, $\alpha = \beta = \gamma = \frac{1}{3}$.

Circular Ordering Computation and Split Weight Estimation. The Neighbor-Net algorithm agglomerates connected components until there are only three nodes (two connected components) in the graph $G_C^{(i)}$. When this occurs, the algorithm constructs a simple undirected cycle $G_Z^{(0)}$ of the three nodes. The algorithm creates a circular ordering $\mathcal{Z}$ from this, the agglomeration history graph, and the agglomeration stack.

From the circular ordering $\mathcal{Z}$, the set of splits S that respect the circular ordering can be calculated, and their weights can be calculated by using the least squares method. Ordinary least squares can produce negative split weights, which have no evolutionary meaning, so a non-negative constraint is imposed. The circular split network is mathematically motivated and allows the phylogenetic network to be visualized in a plane; however, some biological phenomena may not be able to be visualized in a plane, so the Neighbor-Net approach could miss those phenomena [4, 17]. Further details can be found in [11] and [4].

However, the constrained optimization problem requires substantial computational time. For an input of 2000 taxa, the circular ordering computation took several minutes while the split weight computation took three days. Currently, the split weights are estimated with a method by Lawson and Hanson [16] implemented in Parallel Java 2 [14]. However, this code takes $\theta(n^4)$ memory. A customized solver that takes $\theta(n^2)$ memory is being developed.

Neighbor-Net Algorithm Summary. The algorithm 1 gives a summary of the neighbor net algorithm. The distance matrix $\delta_{\mathcal{X}}$ is the input matrix, and the graphs $G_C^{(0)}$ and $G_A^{(0)}$ are initialized to a set of $m^{(0)} = n$ unconnected nodes. The algorithm then iteratively agglomerates minimizing connected components in $G_C^{(i)}$ according to the Q -criterion storing the agglomeration history in $G_A^{(i)}$ and the stack $A_H^{(i)}$. When there are only three nodes left, the while loop on line 6 terminates and the three remaining nodes are put into a simple cycle $G_Z^{(0)}$. This graph is updated by popping off nodes from $A_H^{(i)}$ and using $G_A^{(i)}$ to find the neighbor and children of the nodes. The children in $G_A^{(i)}$ replace the parents in $G_Z^{(i)}$ until the stack is empty. At this point, the order that the nodes are traversed in the simple cycle in $G_Z^{(i)}$ gives the circular ordering $\mathcal{Z}$ that determines the splits S on lines 14 and 15. Finally, in the split weight estimation phase on line 16, the split weights are calculated. The algorithm returns the circular ordering $\mathcal{Z}$, the splits S , and the split weights f_S . Visualizing the network can be done with the algorithm from section 7.2 of the book Phylogenetic Networks by Huson, Rupp, and Scornavacca [11]. Fast Neighbor-Net outputs a Nexus file that can be visualized in SplitsTree4.

Algorithm 1 Neighbor-Net

```

1: Input:  $\delta_{\mathcal{X}}^{(0)}$  a lower triangular matrix representing pairwise taxa distances
2: procedure NEIGHBOR-NET( $\delta_{\mathcal{X}}^{(0)}$ )
3:   Let  $m^{(0)}$  be the number of rows in  $\delta_{\mathcal{X}}^{(0)}$ 
4:   Initialize an empty stack  $A_H^{(0)}$ 
5:   Let  $G_A^{(0)}$  and  $G_C^{(0)}$  be graphs with  $m^{(0)}$  nodes and no edges
6:   while  $m^{(i)} > 3$  or there are more than 2 connected components in  $G_C^{(i)}$  do
7:     Find the  $Q$  criterion for each connected component pair  $c_k, c_j \in G_C^{(i)}$ 
8:     Agglomerate the minimizers by updating  $G_A^{(i)}$ ,  $G_C^{(i)}$ , and  $A_H^{(i)}$ 
9:     Update  $\delta_{\mathcal{X}}^{(i)}$  with the new node distances and update the row sums  $r_{\delta_C}^{(i)}(c_k)$ .
10:    Reduce  $m^{(i)}$  as appropriate
11:    Put the three nodes in  $G_C^{(i)}$  in a simple cycle  $G_Z^{(0)}$ 
12:    while  $A_H^{(i)}$  is not empty do
13:      Pop  $A_H^{(i)}$  to get node  $n_k$ 
14:      Replace node  $n_k$  and  $n_k$ 's neighbor in  $G_A^{(i)}$  with their children in  $G_Z^{(j)}$ 
15:    Get the circular ordering  $\mathcal{Z}$  from  $G_Z^{(j)}$ 
16:    From  $\mathcal{Z}$ , compute the splits  $S$ 
17:    Compute the split weight function  $f_S$ 
18:  return  $\mathcal{Z}$ ,  $S$ , and  $f_S$ 

```

The time complexity of the search and agglomeration phase is $\Theta(n^3)$. It can be shown that there are $\theta(n)$ iterations of the while loop on line 6, and there are $\theta(n^2)$ pairs to calculate the Q -criterion on line 7. Lines 8 and 10 take constant time, and line 9, updating the distance matrix and row sums, takes linear time. Storing the distances takes $\theta(n^2)$ memory.

3. The Relaxed Search Strategy

The canonical search strategy chooses a pair of connected components among all connected components in $G_C^{(i)}$ that minimizes the Q -criterion in line 7 of Algorithm 1. The relaxed search strategy replaces this approach by choosing a pair that minimize the Q -criterion with respect to each other.

More formally,

1. Until an agglomeration event occurs, select the next connected component c_m from a permutation $P_C^{(i)}$ of the list of connected components in $G_C^{(i)}$. Do the following.
 - (a) Calculate $c_k = \operatorname{argmin}\{Q(c_m, c_j) \mid \forall c_j \neq c_m \in G_C^{(i)}\}$
 - (b) Calculate $c_l = \operatorname{argmin}\{Q(c_k, c_j) \mid \forall c_j \neq c_k \in G_C^{(i)}\}$
 - (c) If $c_m = c_l$ agglomerate c_m and c_k .

This heuristic has worst-case time-complexity $\Theta(n^3)$ since

all connected component pairs may need to be examined. In the best-case, step 1(a) will have an immediate minimizing neighbor and will not need to do more than one iteration of the `for` loop of line 1. According to the Relaxed Neighbor-Joining paper [7], this occurs with probability ranging from $4/n$ to 1. The probability is higher for perfectly balanced trees and lower for pectinate “comb-like” trees. This gives an average-case time complexity of $\mathcal{O}(n^2 \log n)$. The heuristic is nondeterministic, and calculating the permutation uses the unbiased Fisher-Yates procedure.

4. Results and Discussion

The correctness and accuracy of the canonical approach need not be tested since it has been tested in the existing literature [3, 4, 5], but the speed enhancements of this approach have been studied. The quality of the relaxed search strategy will be discussed.

4.1. Speed

Performance tests were performed on an Intel Core i7-5820k 6-core CPU @ 3.30 GhZ with 16 GB RAM and a 12GB Java heap in server mode on OpenSuse 13.2 and Java 8. Arbitrary protein family taxa were taken from the PFAM database for performance testing. There were around 50 files with between 2000–10000 taxa. The sequence data was converted to Phylip distance format using QuickJoin [19] and the process in [24].

The original Neighbor-Net code from SplitsTree4 was modified and integrated into Fast Neighbor-Net. This allows a fair comparison of speed. Fast Neighbor-Net calculates the row sums on line 9 in Algorithm 1 in linear time by subtracting the old distances from each connected component and then adding the distance to the new connected component to each connected component. The original implementation recreated the row sums from scratch, a quadratic process. Therefore, the single-threaded run-time of Fast Neighbor-Net is approximately twice that of the original. The distance matrix, $\delta_{\mathcal{X}}^{(i)}$, in Neighbor-Net used a row for each new node, but Fast Neighbor-Net reuses rows when new nodes are created. This reduced memory needed for the array by approximately 9.

Fast Neighbor-Net implements parallelism by giving each thread an equal-sized block of connected component pairs to compute the Q -criterion. Figure 2 shows how run-time scaled with 1, 3, and 5 threads. (Executions with 2 and 4 threads were excluded for readability.) Using three threads improved performance by an average of approximately 2.1. The relaxed approach was implemented in parallel by giving each thread a block of taxa pairs to compute the Q -criterion. The relaxed approach only showed performance gains with 10,000+ taxa.

4.2. Quality

Mammals and Branching Eukaryotes Data. Two real data sets were used to test the quality of the relaxed search approach. A 30 taxa set for mammals was taken from the

Figure 2: Multithreaded performance of the canonical approach.

Figure 3: A thirty mammal phylogeny from data from SplitsTree4 computed with the relaxed search strategy. All the major clades are resolved accurately.

SplitsTree4 software. Figure 3 shows that the major mammal clades are nicely resolved.

The evolution of early eukaryotes is an important part of evolutionary theory. Analyzing data with mitochondrial genes is challenging because of the few genes on the mitochondria, but it has led to a better understanding of evolutionary signal. The approach of [21] analyzed gene order data of mitochondria for 18 taxa using the normalized breakpoint distance. Biologists often use uncorrected distances like this when the evolutionary model is uncertain. Figure 4 shows a nice resolution of major clades. It resolved the green algae and red algae, and it grouped the alveolates with the stramenopiles. This is consistent with recent hypotheses. Neighbor-Joining and split decomposition did not do this [3]. The local search strategy produced the same result as the canonical strategy.

Figure 4: Early branching eukaryotes

Topology Distance. The circular ordering gives a general topology for the split network, so comparing the distance between the circular ordering computed by the canonical approach and the circular ordering compute by the relaxed heuristic gives an idea of whether the relaxed heuristic is better than a random topology. The normalized Kendall tau distance between the canonical approach's ordering and the relaxed approach's ordering was computed on 37 PFAM data sets of 2000–13000 taxa and averaged across five executions of the relaxed heuristic. The average was 0.379, and all distances were below 0.5, which is the average distance to a random permutation. This suggests that the topologies inferred are reasonably good. The least distance was 0.10, which is quite close.

4.3. Correctness

Input distances are *additive* when they correspond to the unique distances between taxa defined by the edges of a tree. The canonical search strategy recovers this tree when the input distances are additive [4]. Relaxed Neighbor-Joining recovers the correct tree when the distances are additive and an additive check is performed [7]. Therefore, the relaxed search heuristic for Neighbor-Net should recover the correct tree when the input distances are additive and an additive check is performed.

5. Conclusion and Future Work

Fast Neighbor-Net is a fast implementation of the Neighbor-Net algorithm with a new relaxed search strategy that has good accuracy and performance qualities. Its input is a Phylip distance file, and it produces a Nexus file that can be rendered in SplitsTree4. It has a command-line interface for deployment on institutional compute clusters. The algorithm was modernized with multithreading. It is available at jacobporter.com/software/. Other search strategies such as a less accurate but faster random search strategy has been implemented. The memory requirements of the split weight estimation can be optimized. Further analysis is planned.

Acknowledgements. David Bryant provided the original

Java implementation of Neighbor-Net, and he provided the eukaryote data. Travis Wheeler provided the PFAM data. Layne Watson of Virginia Tech gave helpful comments about terminology and mathematical notation.

References

- [1] H.-J. Bandelt and A. W. Dress, "Split decomposition: A new and useful approach to phylogenetic analysis of distance data," *Molecular phylogenetics and evolution*, vol. 1, no. 3, pp. 242–252, 1992.
- [2] H.-J. Bandelt, V. Macaulay, and M. Richards, "Median networks: Speedy construction and greedy reduction, one simulation, and two case studies from human mtDNA," *Molecular phylogenetics and evolution*, vol. 16, no. 1, pp. 8–28, 2000.
- [3] D. Bryant and V. Moulton, "NeighborNet: An agglomerative method for the construction of planar phylogenetic networks," in *Proceedings of the 2nd International Workshop on Algorithms in Bioinformatics*. Springer, 2002, pp. 375–391.
- [4] D. Bryant and V. Moulton, "Neighbor-net: An agglomerative method for the construction of phylogenetic networks," *Mol Biol Evol.*, vol. 21, pp. 255–65, 2004.
- [5] D. Bryant, V. Moulton, and A. Spillner, "Consistency of the neighbor-net algorithm," *Algorithms for Molecular Biology*, vol. 2, pp. 8–20, 2007.
- [6] A. W. Dress *et al.*, "Noisy: Identification of problematic columns in multiple sequence alignments," *Algorithms Mol Biol*, vol. 3, no. 7, pp. 7188–3, 2008.
- [7] J. Evans, L. Sheneman, and J. Foster, "Relaxed neighbor joining: A fast distance-based phylogenetic tree construction method," *Journal of molecular evolution*, vol. 62, no. 6, pp. 785–792, 2006.
- [8] J. Felsenstein, *Inferring Phylogenies*. Sunderland, Mass.: Sinauer Associates, 2004.
- [9] R. Finn *et al.*, "The pfam protein families database," *Nucleic Acids Research*, vol. 38, pp. D211–222, 2010.
- [10] D. Huson and D. Bryant, "Application of phylogenetic networks in evolutionary studies," *Molecular Biology and Evolution*, vol. 23, no. 2, pp. 254–267, 2006.
- [11] D. H. Huson, R. Rupp, and C. Scornavacca, *Phylogenetic Networks*. Cambridge, United Kingdom: Cambridge University Press, 2010.
- [12] T. H. Jukes and C. R. Cantor, "Evolution of protein molecules," *Mammalian protein metabolism*, vol. 3, pp. 21–132, 1969.
- [13] A. B. K. Howe and R. Durbin, "Quicktree: Building huge neighbour-joining trees of protein sequences," *Bioinformatics*, vol. 11, pp. 1546–7, 2002.
- [14] A. Kaminsky, "Parallel java: A unified api for shared memory and cluster parallel programming in 100% java," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–8.
- [15] M. Kimura, "A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences," *Journal of molecular evolution*, vol. 16, no. 2, pp. 111–120, 1980.
- [16] C. L. Lawson and R. J. Hanson, *Solving least squares problems*. SIAM, 1974, vol. 161.
- [17] D. Levy and L. Pachter, "The neighbor-net algorithm," *Advances in Applied Mathematics*, vol. 47, no. 2, pp. 240–258, 2011.
- [18] D. R. Maddison, K.-S. Schulz, and W. P. Maddison, "The tree of life web project," *Zootaxa*, vol. 1668, no. Linnaeus Tercentenary: Progress in Invertebrate Taxonomy, pp. 19–40, 2007.
- [19] T. Mailund and C. Pedersen, "Quickjoin - fast neighbor-joining tree reconstruction," *Bioinformatics*, vol. 20, pp. 3261–2, 2004.
- [20] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees," *Mol Biol Evol.*, vol. 4, pp. 406–25, 1987.
- [21] D. Sankoff *et al.*, "Early eukaryote evolution based on mitochondrial gene order breakpoints," *Journal of Computational Biology*, vol. 7, no. 3–4, pp. 521–535, 2000.
- [22] L. Sheneman, J. Evans, and J. Foster, "Clearcut: A fast implementation of relaxed neighbor joining," *Bioinformatics*, vol. 22, pp. 2823–4, 2006.
- [23] M. Simonsen, T. Mailund, and C. Petersen, "Rapid neighbor-joining," in *Proceedings of the 8th International Workshop on Algorithms in Bioinformatics*. WABI, 2008, pp. 113–122.
- [24] T. Wheeler, "Large-scale neighbor-joining with ninja," in *Proceedings of the 9th Workshop on Algorithms in Bioinformatics*. WABI, 2009, pp. 375–389.