Implementation of the RapidNJ heuristic using balanced search trees

Francisco (Curro) Campuzano Jiménez

MSc in Bioinformatics at BiRC, Aarhus University

2023-12-13

A bottom-up clustering method by Saitou and Nei (1987)

The most popular method to construct phylogenetic trees from distance matrices.

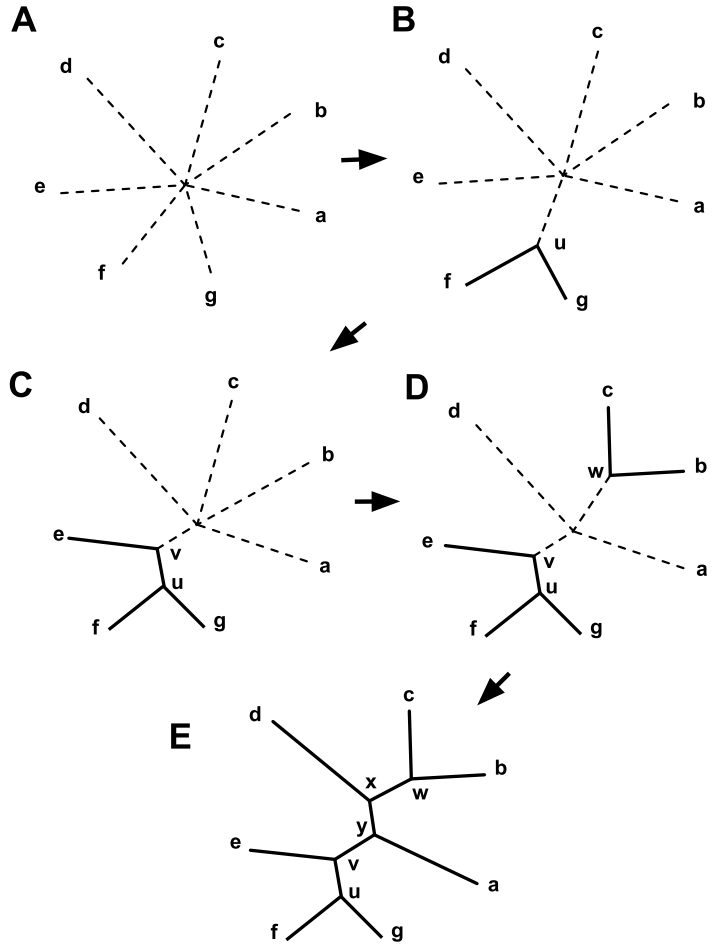

Figure 1: Neighbor-joining diagram

How do you find the pair of neighbors to join?

$$Q(i,j) = (r-2)D(i,j) - \sum_{k=1}^{n} D(i,k) - \sum_{k=1}^{n} D(j,k) \qquad (1)$$

How do you find the pair of neighbors to join?

$$Q(i,j) = (r-2)D(i,j) - \sum_{k=1}^{n} D(i,k) - \sum_{k=1}^{n} D(j,k) \qquad (1)$$

The canonical runtime is $\mathcal{O}(n^3)$:

$$T(n) = \begin{cases} \mathcal{O}(n^2) + T(n-1) & \text{if } n > 3 \\ \mathcal{O}(1) & \text{if } n \leq 3 \end{cases} \qquad (2)$$

Can we avoid searching all pairs?

Number of taxa — Distance — Sum of the i-row — Sum of the j-col

$$Q(i,j) = (r-2)D(i,j) - \sum_{k=1}^{n} D(i,k) - \sum_{k=1}^{n} D(j,k)$$

$$\geq (r-2)D(i,j) - \sum_{k=1}^{n} D(i,k) - u_{\max}$$

$u_{\max}$: Largest sum of all columns

The lower bound of $Q(i,j)$ cannot decrease if we iterate across the i-row in ascending order!

Canonical neighbor-joining

Input : A $r \times r$ distance matrix D
Output: Pair of selected neighbors (i, j)

$q_{\text{min}} \leftarrow \infty$;
neighbors $\leftarrow (-1, -1)$
for $i \leftarrow 1$ **to** r **do**
  for $j \leftarrow i + 1$ **to** r **do**
    if $Q[i, j] < q_{min}$ **then**
      $q_{\text{min}} \leftarrow Q[i, j]$
      neighbors $\leftarrow (i, j)$
    end
  end
end
return neighbors

RapidNJ by Simonsen et al. (2011)

Input : A $r \times r$ distance matrix D, and row-wise sums U
Output: Pair of selected neighbors (i, j)

$u_{\text{max}} \leftarrow \max(U)$; $q_{\text{min}} \leftarrow \infty$; neighbors $\leftarrow (-1, -1)$
for $i \leftarrow 1$ **to** r **do**
  for $j \leftarrow i + 1$ **to** r *in ascending sorted order* **do**
    if $D[i, j] \cdot (r - 2) - U[i] - u_{max} > q_{min}$ **then**
      break;
    end
    $q \leftarrow \text{row}[j] \cdot (r - 2) - U[i] - U[j]$
    if $q < q_{min}$ **then**
      $q_{\text{min}} \leftarrow q$
      neighbors $\leftarrow (i, j)$
    end
  end
end
return neighbors

Original data structures

$$D = \begin{bmatrix} 0 & 5 & 9 & 9 & 8 \\ 5 & 0 & 10 & 10 & 9 \\ 9 & 10 & 0 & 8 & 7 \\ 9 & 10 & 8 & 0 & 3 \\ 8 & 9 & 7 & 3 & 0 \end{bmatrix}$$

$$S = \begin{bmatrix} 0 & 5 & 8 & 9 & 9 \\ 0 & 5 & 9 & 10 & 10 \\ 0 & 7 & 8 & 9 & 10 \\ 0 & 3 & 8 & 9 & 10 \\ 0 & 3 & 7 & 8 & 9 \end{bmatrix}$$

$$V = \begin{bmatrix} 1 & 2 & 5 & 3 & 4 \\ 2 & 1 & 5 & 4 & 3 \\ 3 & 5 & 4 & 1 & 2 \\ 4 & 5 & 3 & 1 & 2 \\ 5 & 4 & 3 & 1 & 2 \end{bmatrix}$$

Our approach: B-trees

	a	b	c	d	e
a	0	5	9	9	8
b	5	0	10	10	9
c	9	10	0	8	7
d	9	10	8	0	3
e	8	9	7	3	0

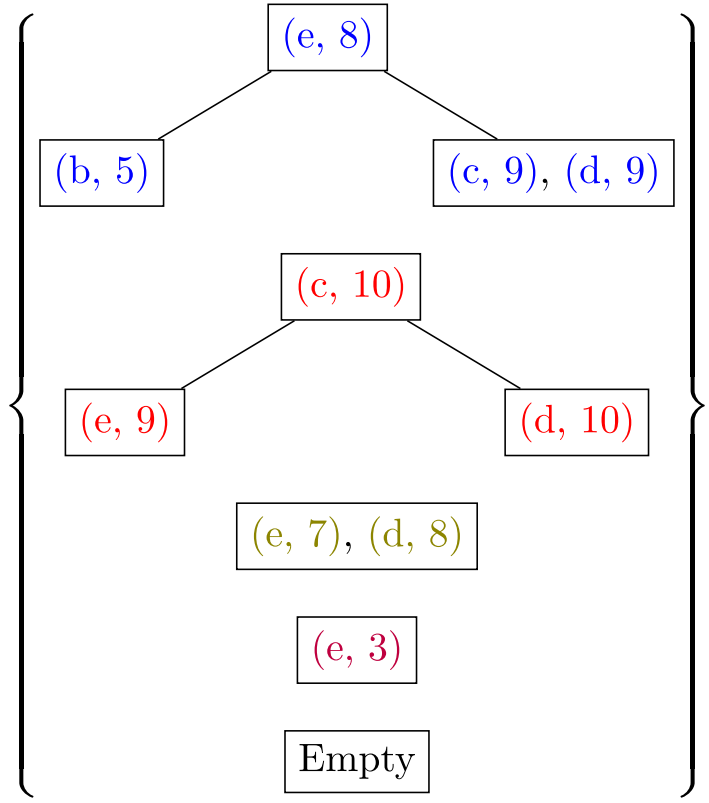

Figure 2: Distance matrix and collection of B-trees at iteration t

Is there any improvement?

Initializing

1. Compute the sum of every row U: $\mathcal{O}(n^2)$
2. Populate the collection of B-trees:
   $\mathcal{O}(\sum_{i=1}^{n} \sum_{j=i}^{n} \log(n-i)) = \mathcal{O}(n^2 \log n)$

Every iteration

1. Find i, j: best-case $\mathcal{O}(n)$, worst-case $\mathcal{O}(n^2)$
2. Update distance matrix and row-sums U: $\mathcal{O}(n)$
3. Update the collection of B-trees: $\mathcal{O}(n \log n)$

Is there any improvement?

Initializing

1. Compute the sum of every row U: $\mathcal{O}(n^2)$
2. Populate the collection of B-trees: $\mathcal{O}(\sum_{i=1}^{n} \sum_{j=i}^{n} \log(n-i)) = \mathcal{O}(n^2 \log n)$

Every iteration

1. Find i, j: best-case $\mathcal{O}(n)$, worst-case $\mathcal{O}(n^2)$
2. Update distance matrix and row-sums U: $\mathcal{O}(n)$
3. Update the collection of B-trees: $\mathcal{O}(n \log n)$

RapidNJ overall runtime

- Best-case of $\mathcal{O}(n^2 \log n)$ if we estimate $\hat{q}_{\min}$ correctly every time.
- Worst-case of $\mathcal{O}(n^3)$, as in the canonical algorithm.

Methods and implementation

- We implemented the canonical algorithm, the RapidNJ heuristic using B-Trees, and a hybrid strategy using Rust.
- Command line application: read from stdin and print in Newick to stdout.
- Extensive testing (unit tests, E2E, property-based and GitHub actions)
- Reproducible benchmark using PFAM alignments

Canonical NJ: A nice surprise

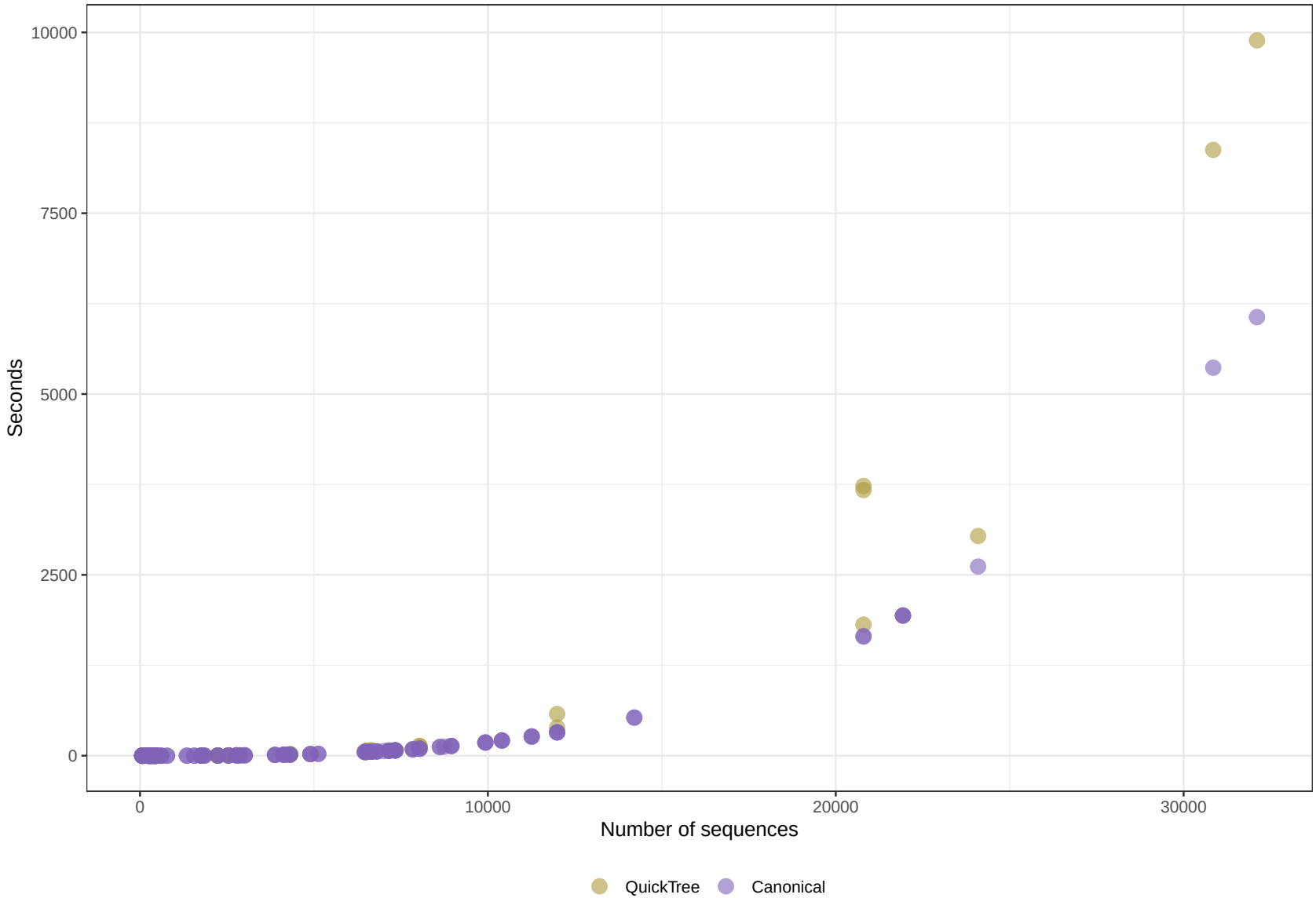

Figure 3: Canonical neighbor-joining implementations.

RapidNJ: As expected

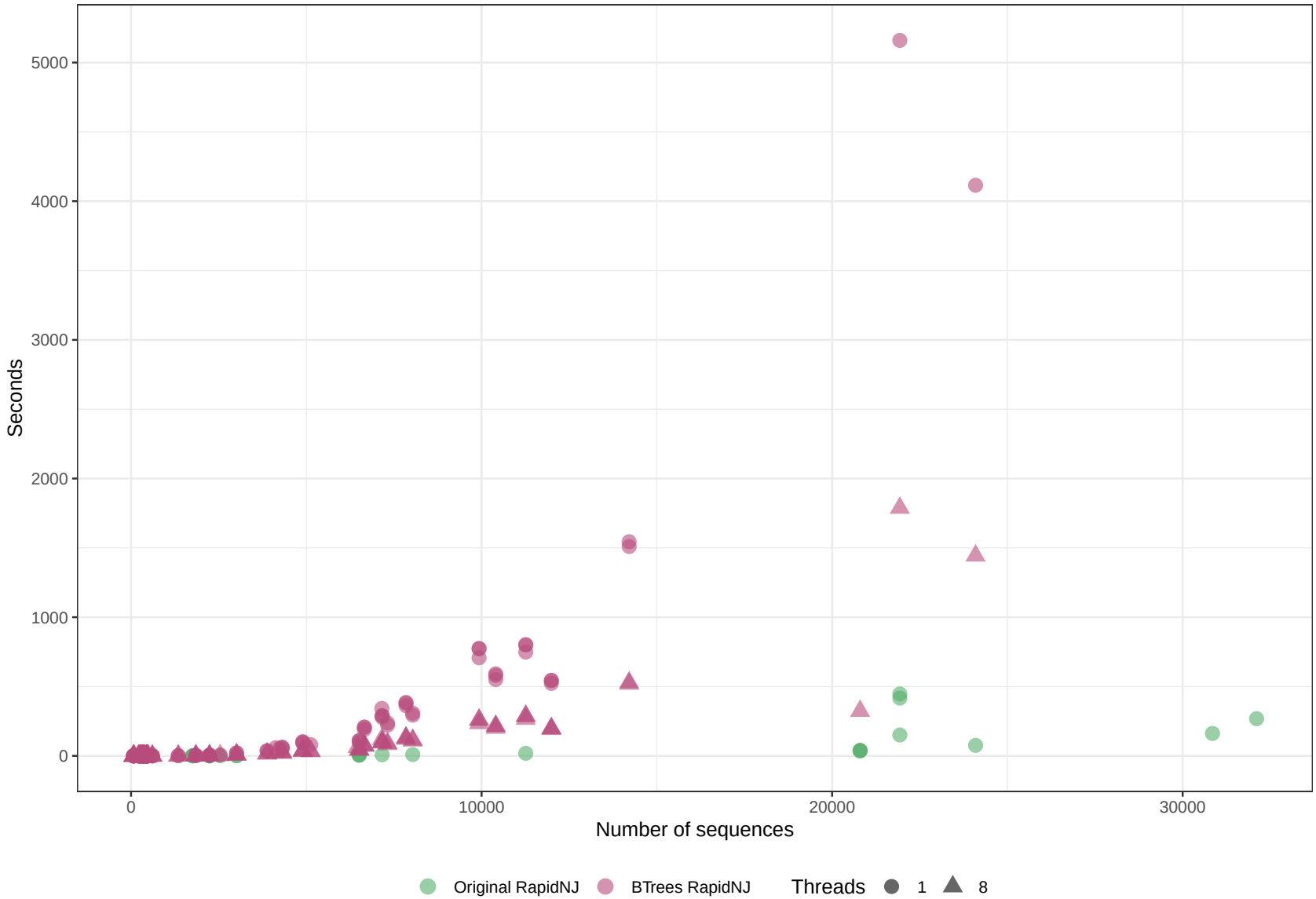

Figure 4: RapidNJ heuristic implementations.

Hybrid strategy: too simplistic?

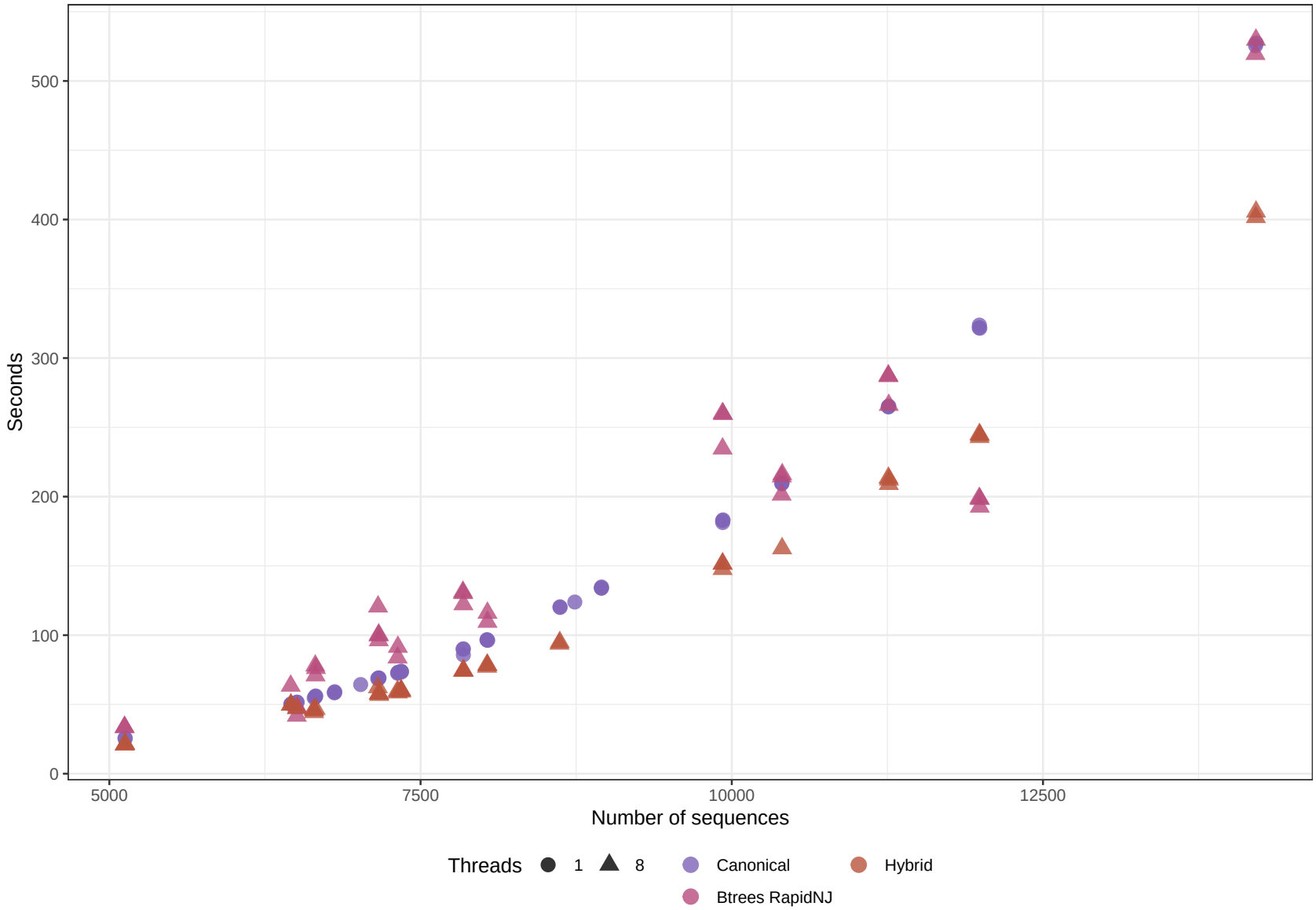

Figure 5: Runtime in seconds of different strategies.

Asymtotic behaviour

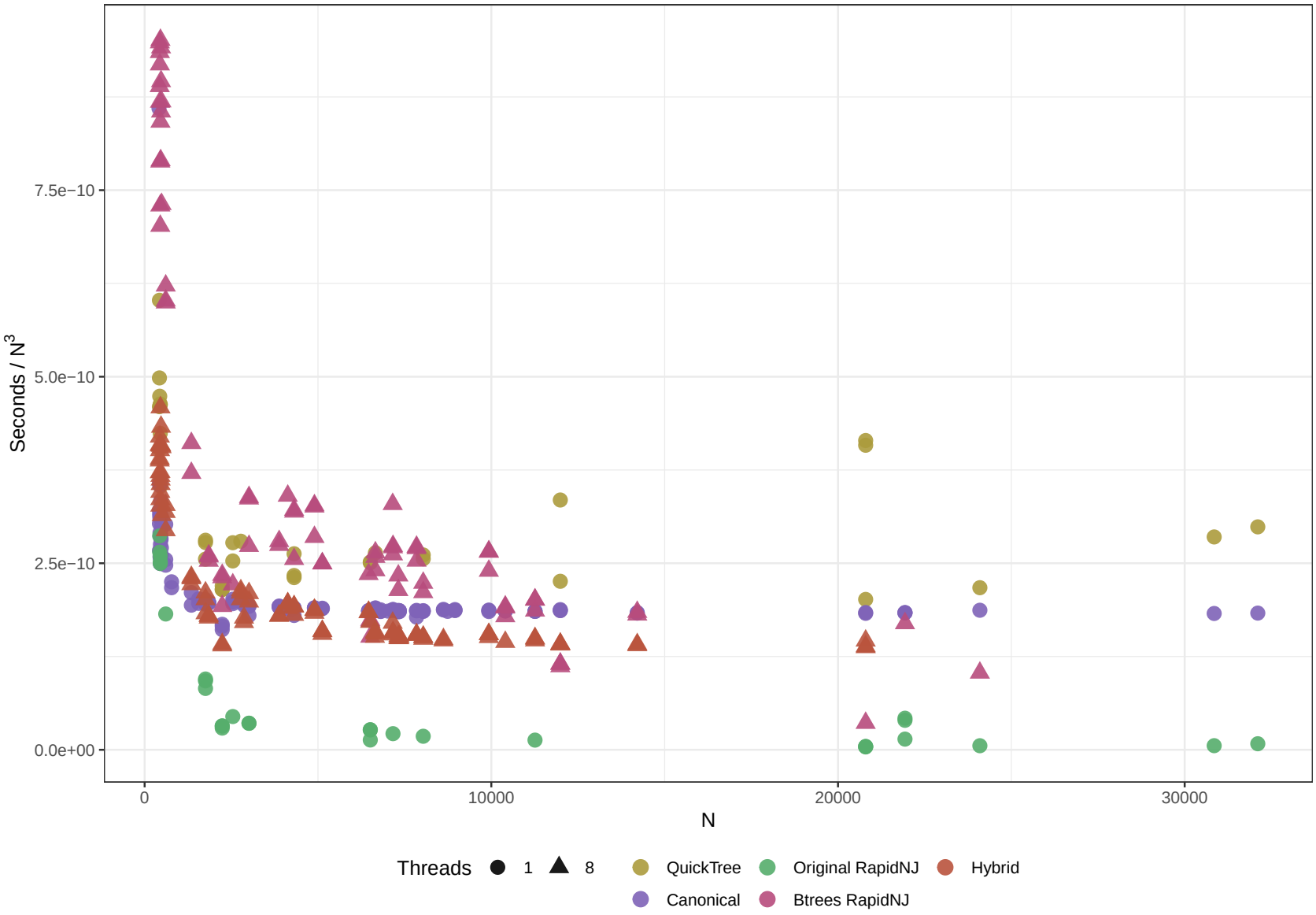

Figure 6: Ratio of runtime to N^3.

Future work

1. Dig into the QuickTree source code. Why is it slower than our canonical implementation?
2. Implement custom balanced tree structures to use in the RapidNJ heuristic.
3. Implement additional optimization tricks from the original RapidNJ program.
4. Determine a set of decision rules to use with the hybrid strategy

Conclussions

- Balanced search trees are suitable data structures to implement the RapidNJ heuristic.
- Rust is an appropriate language alternative to C or C++ to implement bioinformatics algorithms with a rich tooling ecosystem.
- Further optimization of my implementation requires custom data structures and/or optimization tricks that would obscure the implementation.

Thanks!

Non-optimized memory usage

There is much room for optimization, but we chose to stick with the Standard library.

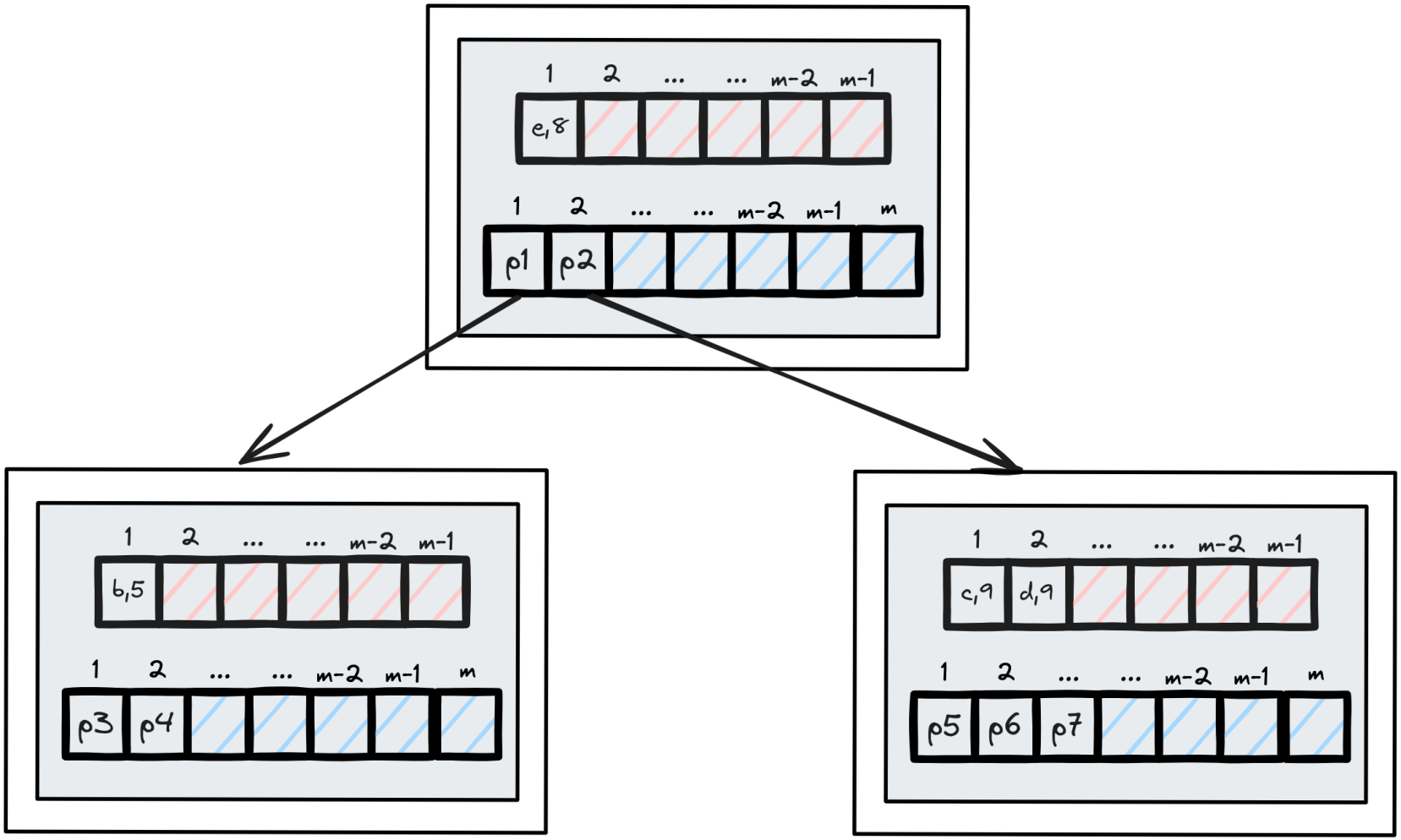

Figure 7: Schematic internal representation of a B-tree

Lookups

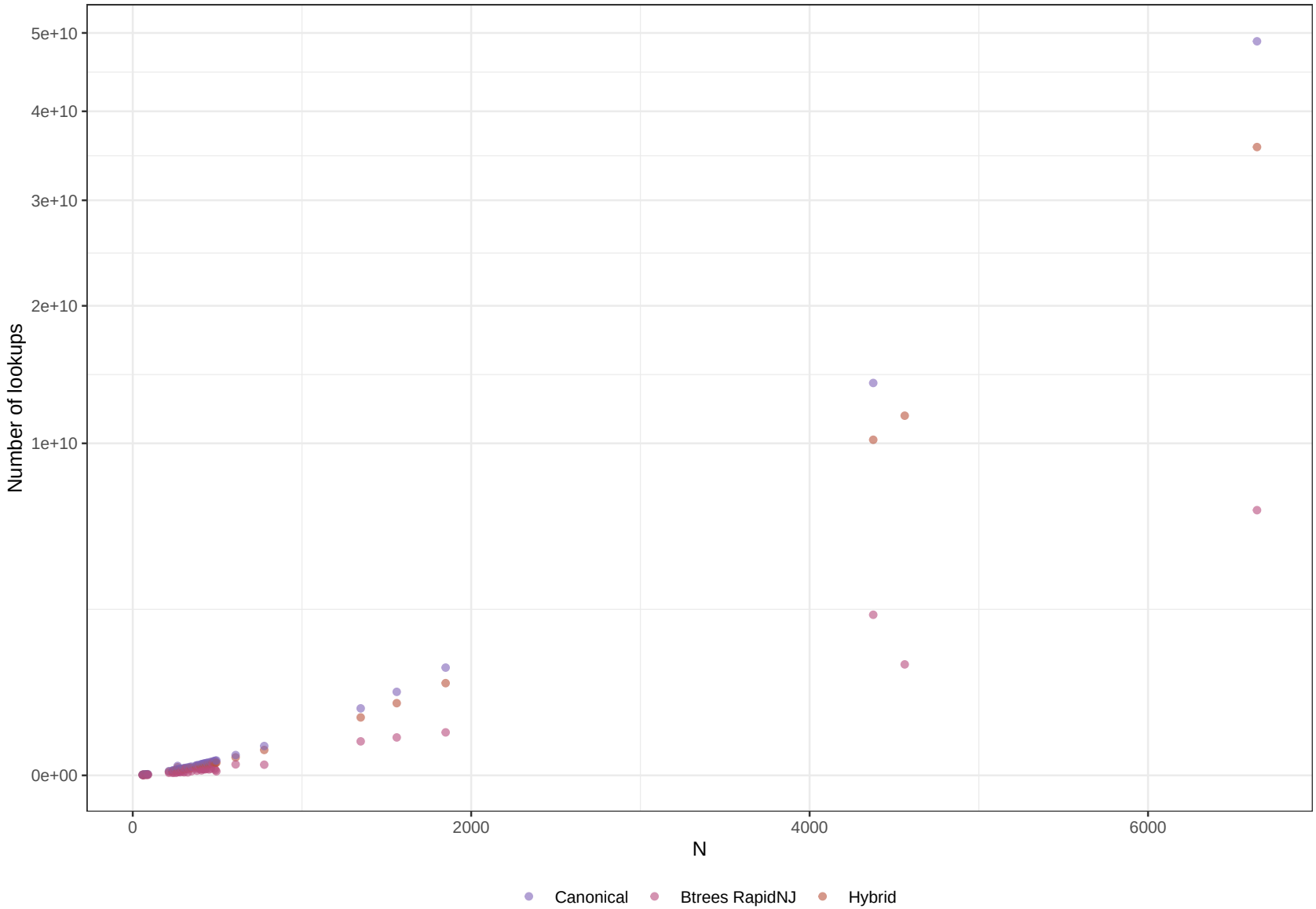

Figure 8: Number of lookups to the Q

Testing

- Unit-test of the critical use cases (convert the phylogenetic tree into Newick, find the pair of neighbors for a small matrix ...)
- End-to-end test with small examples from Wikipedia and Algorithms in Bioinformatics slides.
- Property-based testing.
- Continuous Testing with GitHub Actions

Input: Minimum number of taxa $n_{\min}$, maximum number of taxa $n_{\max}$, and tolerance ϵ

$n \leftarrow \texttt{RandomInteger}\ (n_{\min}, n_{\max})$
$T \leftarrow \texttt{GenerateRandomTree}\ (n)$
$D \leftarrow \texttt{CalculateDistanceMatrix}\ (T)$

return $\texttt{BranchScore}\ (T, \texttt{NeighborJoining}\ (D)) < \epsilon$

Reconstructing the distance matrix (1/2)

$$D(f,u) = \frac{1}{2}D(f,g) + \frac{1}{2(r-2)}\left[\sum_{k=1}^{n} D(f,k) - \sum_{k=1}^{n} D(g,k)\right] \tag{3}$$

$$D(g,u) = D(f,g) - D(f,u) \tag{4}$$

$$D(u,k) = \frac{1}{2}[D(f,k) + D(g,k) - D(f,g)] \tag{5}$$

Termination step (2/2)

$$
\begin{aligned}
D(v,i) &= \frac{D(i,j) + D(i,m) - D(j,m)}{2} \\
D(v,j) &= \frac{D(i,j) + D(j,m) - D(i,m)}{2} \\
D(v,m) &= \frac{D(i,m) + D(j,m) - D(i,j)}{2}
\end{aligned}
\tag{6}
$$

Why B-tree's height is logarithmic

Let us consider a B-tree of order m. This implies that every node can have at most m children and contain at most $m-1$ elements. Let us denote $b = [\frac{m}{2}]$, the minimum number of children a node (except the root) can have. This implies that every non-root element must contain at least $b-1$ elements.

Height	n(h)
1	1
2	2
3	$2 \cdot b \cdot (b-1) + 2 + 1$
h	$2b^{h-1} - 1$

$$n = 2b^{h-1} - 1$$

$$h \in \mathcal{O}(\log n)$$

References

Saitou, N., and Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4, 406–425. doi:10.1093/oxfordjournals.molbev.a040454.

Simonsen, M., Mailund, T., and Pedersen, C. N. S. (2011). Inference of large phylogenies using neighbour-joining. in *Biomedical engineering systems and technologies*, eds. A. Fred, J. Filipe, and H. Gamboa (Berlin, Heidelberg: Springer Berlin Heidelberg), 334–344.