Journal of Information and Data Management, 2025, 16:1, doi: 10.5753/jidm.2025.4361

Exploring Evolutionary Patterns: A Jupyter Notebook for Discovering Frequent Subtrees in Phylogenetic Tree Databases

João Vitor Moraes [**Universidade Federal Fluminense** | ***joaovitormoraes@id.uff.br***]
Camila Ferrari [**Universidade Federal Fluminense** | ***camilaferrari@id.uff.br***]
Isabel Rosseti [**Universidade Federal Fluminense** | ***rosseti@ic.uff.br***]
Daniel de Oliveira [**Universidade Federal Fluminense** | ***danielcmo@ic.uff.br***]

Institute of Computing, Universidade Federal Fluminense, Av. Gal. Milton Tavares de Souza, s/n, São Domingos, Niterói, RJ, 24210-590, Brazil.

Received: 8 April 2024 • **Published:** 23 August 2025

Abstract The exploratory analysis of evolutionary information within a phylogenetic tree database is a crucial task in the field of bioinformatics. Phylogenetic trees are constructed by exploring multiple evolutionary and tree construction methods. For instance, methods like Maximum Parsimony, Maximum Likelihood, and Neighbor-Joining may yield slightly different trees due to their distinct approaches to inferring phylogenies (*e.g.*, distance and character-based methods). Therefore, analyzing evolutionary data often entails identifying frequent subtrees within a given set of phylogenetic trees. However, this identification process can be computing-intensive, depending on the size of the input tree database. In this manuscript, we introduce the `NMFSt.P` Notebook, which aims to simplify the comparison of multiple phylogenetic trees for identifying frequent subtrees in the database. Our experiments demonstrate that `NMFSt.P` produces results comparable to the baseline approach while bringing the advantage of flexibility for the scientist.

Keywords: phylogenetics, frequent subtrees, bioinformatics, workflows.

1 Introduction

With the rapid evolution of technologies integrating Biology with Computer Science, *e.g.*, high-throughput DNA sequencing techniques coupled with High-Performance Computing (HPC) environments (*e.g.*, clouds and clusters), the volume of available biological data has increased exponentially [Ocaña and de Oliveira, 2015]. This increase in data availability has had a significant impact on the field of phylogenetics, enabling the generation of large-scale phylogenetic trees [Goloboff *et al.*, 2009] through the execution of standalone applications or complex scientific workflows [Guedes *et al.*, 2017; Ocaña *et al.*, 2011]. Analyzing these phylogenetic trees is crucial for elucidating the evolutionary relationships among organisms, providing essential insights for subsequent tasks.

Several programs, systems, and services can be used for creating phylogenetic trees, each applying a specific evolutionary method, *e.g.*, maximum parsimony, neighbor-joining, and maximum likelihood [Tommy Tsan-Yuk Lam and Tang, 2010]. Since each program/method has its characteristics, the same input phylogenetic tree database can produce different relationships depending on the adopted program/method [Puigbò *et al.*, 2019]. Choosing the most suitable program/method for a given input database of phylogenetic trees may be a challenge, as it depends on various factors such as the number of sequences, sequence sizes, *etc.* Therefore, scientists often explore several programs/methods in their experiments, thus generating multiple phylogenetic trees based on different methods. Extracting practical knowledge from this database of trees is then a top priority.

Some approaches aim to extract information from the phylogenetic tree database by obtaining a consensus tree [Bryant, 2003]. However, this type of tree may not be able to return frequent subtrees, which can indicate the existence of evolutionary patterns. For example, Figure 1 presents two hypothetical phylogenetic trees named (A) and (B) that have a common subtree (C) that may indicate an evolutionary pattern. Despite providing important insights, finding frequent subtrees is an NP-hard problem [Amir and Keselman, 1997] and may involve comparing several hundreds of trees. The identification of frequent subtrees can be considered a large-scale problem. It is often modeled as a scientific workflow and executed using workflow systems [de Oliveira *et al.*, 2019]. However, using workflow systems requires a non-trivial learning curve since each existing system has features that the scientists must understand to model and execute their workflows. Since these workflows often need to be executed in HPC environments, this adds another layer of complexity, which can ultimately limit their efficient usage.

This manuscript introduces the `NMFSt.P` (Notebook for Mining Frequent Subtrees in Parallel) to streamline the process of identifying frequent subtrees without requiring workflow systems or developing complex implementations on the users' part. Our notebook can detect frequent patterns, specifically subtrees, within the input tree topologies. Building upon prior work by our research group, which focused on modeling phylogenetic analysis workflows and subtree mining using existing workflow systems, the `NMFSt.P` extends this framework. Implemented in Python, it leverages the Parsl library [Babuji *et al.*, 2019] to introduce the neces-

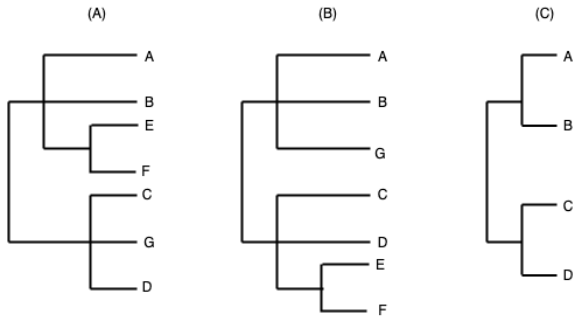

Figure 1. Hypothetical example of two phylogenetic trees and a subtree.

sary parallelism for efficient task execution. By adopting the `NMFSt.P`, scientists can identify, from the generated phylogenetic tree database, which subtrees are shared among different phylogenetic trees. To evaluate the proposed notebook, we used a database comprising 200 phylogenetic trees derived from protozoan genes and constructed using various methods. The findings showed promising results, reinforcing the effectiveness of the approach.

This manuscript is an extension of work originally reported in the Proceedings of the Brazilian e-Science Workshop (BreSci) [Ferrari *et al.*, 2023] held in Belo Horizonte, MG - Brazil, on September 2023. We enriched the experimental evaluation with a brand-new experiment in this extended version and improved the background and related work sections. We organized the remainder of this manuscript into five sections following this introduction. Section 2 provides background knowledge, while Section 3 delves into related work. Section 4 offers detailed insights into the proposed notebook, `NMFSt.P`. Section 5 presents the experimental results, and finally, Section 6 concludes the manuscript and outlines potential future work.

2 Background Knowledge

This section discusses important concepts regarding this manuscript, *i.e.,* sequence alignment, methods for inferring phylogenies and phylogenetic trees. It is widely acknowledged that DNA and RNA sequences are represented as strings of letters, each corresponding to a nucleotide base. Specifically, these letters denote Adenine (A), Cytosine (C), Guanine (G), Thymine (T), and Uracil (U or Ura) [Setubal and Meidanis, 1997]. While DNA sequences employ A, C, G, and T, RNA sequences use U instead of T during transcription.

In the case of proteins, which are built as chains of amino acids, they are usually labeled as a string of letters, where each amino acid is represented as a letter [Setubal and Meidanis, 1997]. There are 20 standard amino acids found in biological chemistry, coded as A (Ala) for Alanine, C (Cys) for Cysteine, D (Asp) for Aspartic Acid, E (Glu) for Glutamic Acid, F (Phe) for Phenylalanine, G (Gly) for Glycine, H (His) for Histidine, I (Ile) for Isoleucine, K (Lys) for Lysine, L (Leu) for Leucine, M (Met) for Methionine, N (Asn) for Asparagine, P (Pro) for Proline, Q (Gln) for Glutamine, R (Arg) for Arginine, S (Ser) for Serine, T (Thr) for Threonine, V (Val) for Valine, W (Trp) for Tryptophan, and Y (Tyr) for Tyrosine. Typically, text files are employed to represent these sequences in a computational environment, albeit with various possible formats. This manuscript adopts the FASTA format [Markel and Leon, 2003].

In FASTA format, each sequence is preceded by a line starting with the ">" symbol, followed by the sequence name. For instance, in Figure 2, the string represents a fragment of a DNA sequence. Figure 3 represents a fragment of the FASTA file that represents a protein from Plasmodium falciparum, a species of malaria-causing parasite. Additionally, for DNA and RNA sequences, it is noteworthy that there are several letters outside the set $\{A, C, G, T, U\}$. This occurs when the genetic codes cannot be resolved with the desired confidence level, prompting the format to accommodate ambiguous representations. In this context, R denotes either G or A, Y denotes either T or C, K denotes either G or T, M denotes either A or C, S denotes either G or C, W denotes either A or T, B denotes G, T, or C, D denotes G, A, or T, H denotes A, C, or T, V denotes G, C, or A, and N denotes any of the four bases.

```
>gi|186704|Keratin Homo sapiens keratin
CCCAGGGTCCGATGGGAAAGTGTAGCCTGCAGGCCCACACCTCCCC
CTGTGAATCACGCCTGGCGGGACAAGAAAGCCCAAAACACTCCAAA
CAATGAGTTTCCAGTAAAATATGACAGACATGATGAGGCGGATGAG
AGGAGGGACCTGCCTGGGAGTTGGCGCTAGCCTGTGGGTGATGAAA
GCCAAGGGGAATGGAAAGTGCCAGACCCGCCCCCTACCCATGAGTA
TAAAGCACTCGCATCCCTTTGCAATTTACCCGAGCACCTTCTCTTC
ACTCAGCCTTCTGCTCGCTCGCTCACCTCCCTCCTCTGCACCATGA
```

Figure 2. A fragment of a FASTA file containing a DNA sequence.

```
>pfnc|PfNF135_050016300
MLSLKNVKSNDEVNDIRNANDTFNKYSRSIIPMEHNIIVLPCECK
TSIIKNTFLDILSPMKIPFCKINNTNVQNTTNVFSLRKKKKTLRC
ENILNQNKGNTKNDKEQNDNLITCHNNFKSFNSNYLDTYSIIGGT
YKNTYFKNKMDNKYFTIEIERKYDIINEDKNPFDYYTYVAMKNQH
RNYLALKNIPYIEKQIMNCRDLNSVYINKNIVIPEIQYKHNKKTK
ITKRDLIEYNCIKDNANDFFNLNTEISNTLVKDNMISRIINENEL
KKNQSLSLIDDRKKSIARNISEKNNQIIYNSKQHFNIFDDIRPSI
KKNIKKKKKNLGVCYLNLNNLSCQFLMTCDKT
```

Figure 3. A fragment of a sequence in FASTA format representing a protein from Plasmodium falciparum.

However, deriving meaningful insights only from the independent analysis of sequences poses a significant challenge. By comparing sequences, scientists can identify potential evolutionary relationships among them. Consequently, these sequences must undergo alignment. Sequence alignment entails the arrangement of sequences to identify regions of similarity, which can potentially correspond to evolutionary relationships. The basic concept underlying sequence alignment is organizing multiple (and different) sequences in rows, aligning them one above the other so that similar bases are positioned vertically. When sequences are homologous, they show a certain level of similarity. Differences in bases at specific sequence positions denote mutations, insertions, or deletions. Additionally, gaps may be introduced into the sequence to ease alignment.

The sequence alignment task is tedious and error-prone to accomplish manually, especially for large sequences. Thus,

a plethora of software implements sequence alignment tasks, each with specific characteristics, some based on database searches such as BLAST [Altschul *et al.*, 1990] and HMMER [Durbin *et al.*, 1998]. On the other hand, other programs are based on pairwise alignment, such as BLASTZ [Schwartz *et al.*, 2003] and CUDAlign [de O. Sandes *et al.*, 2014]. Finally, some software performs multiple sequence alignment such as MAFFT [Katoh *et al.*, 2017], ClustalW [Chenna *et al.*, 2003], ProbCons [Do *et al.*, 2004], MUSCLE [Edgar, 2004] and T-Coffee [Notredame *et al.*, 2000].

To construct a phylogenetic tree, a fundamental concept in the context of this manuscript, the sequences must be related in some manner. One approach is establishing relationships among sequences based on a shared ancestor, thus identifying a *phylogeny*. The general structure of a phylogenetic tree is depicted in Figure 4. Phylogenetic trees consist of a *root* and numerous *branches*. *Nodes* within the tree represent points where two branches converge, indicating all descendant branches' most recent common ancestor. The scale of the tree denotes the degree of relatedness between branches. At the terminus of the tree, one encounters *leaves*, also referred to as *tips*, representing individual sequences or *taxa*. While trees can be represented textually, they are more effectively visualized using cladograms or phylograms. Phylograms offer the advantage of illustrating both evolutionary direction and genetic distance. The creation of a phylogenetic tree requires the identification of phylogenies between sequences. Various methods exist for inferring phylogenies. Well-known methods include Neighbor-Joining (NJ) [Saitou and Nei, 1987], Maximum Likelihood (ML) [Nixon, 2001], and Maximum Parsimony (MP) [Farris, 1970], all of which can be applied within the notebook proposed in this manuscript. These methods employ distinct algorithms to infer evolutionary relationships, allowing researchers to derive phylogenetic trees from sequence data.

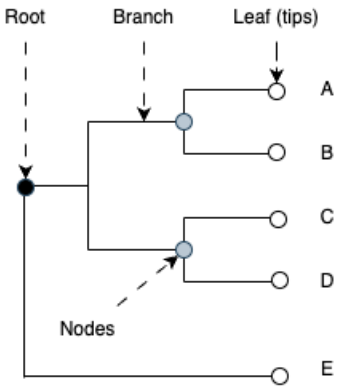

Figure 4. The structure of a phylogenetic tree, with its root, branches, leaves, and nodes.

NJ is a distance-based method that clusters two sequences, often referred to as *neighbors*, according to a distance matrix. Its primary advantage lies in its speed, making it a good choice for large datasets. However, it does not guarantee that the resulting tree accurately represents the evolutionary relationships among the sequences. ML consumes a dataset of aligned sequences and extensively explores all possible trees based on this alignment. While ML offers high accuracy in inferring evolutionary relationships, it is computing-intensive due to the exhaustive exploration of (maybe extensive) tree space.

On the other hand, the MP method assumes that evolutionary changes are infrequent and seeks to minimize the number of such changes in the resulting phylogenetic trees. It selects the tree with the fewest evolutionary changes as the solution. However, a drawback of MP arises when sequences show a high mutation rate, leading to potentially inaccurate tree construction.

3 Related Work

Several papers in the literature have proposed workflows and methodologies for identifying frequent subtrees within phylogenetic tree databases [Guedes *et al.*, 2017; Vilella *et al.*, 2009; Deepak and Fernández-Baca, 2014]. For instance, Guedes *et al.* [2017] introduce `SciPhyloMiner`, a workflow designed within the parallel workflow system `SciCumulus` [de Oliveira *et al.*, 2012]. `SciPhyloMiner` uses the `Dendropy` application [Sukumaran and Holder, 2010] for performing subtree mining. However, despite representing a step forward, `SciPhyloMiner` faces limitations in its parallelism strategy. Since `Dendropy` is non-parallelizable, the most computing-intensive activity of the workflow cannot be distributed across multiple machines.

Similarly, Vilella *et al.* [2009] propose the `EnsemblCompara GeneTrees` workflow, which encompasses multiple activities, including multiple sequence alignment, tree construction, and subtree clustering. This workflow identifies frequent subtrees within large gene families, enabling the exploration of evolutionary relationships among gene sequences across different species. Unlike `NMFSt.P`, designed as a Jupyter Notebook for ease of use and extension, the approaches mentioned earlier constrain users to specific workflow systems. This constraint may limit their applicability across different contexts.

In addition to workflows, numerous papers in the literature propose methods for identifying frequent subtrees within phylogenetic trees, which can be integrated into `NMFSt.P`. For instance, Deepak *et al.* [2014] introduce the `Evominer` algorithm, a parallel algorithm designed to identify frequent subtrees. The authors compared performance with other state-of-the-art solutions, demonstrating that `Evominer` outperforms them by two orders of magnitude. Unlike `NMFSt.P`, `Evominer` assumes that the trees are already available and the sequences are aligned. Similarly, Deepak and Fernández-Baca [2014] propose *MfstMiner*, a tool tailored for identifying all maximal frequent subtrees within databases of phylogenetic trees. A maximal frequent subtree is one with the largest possible number of leaves (tips).

Addressing the challenges surrounding the identification of Maximum Frequent Subtrees (MFASTs) in phylogenetic tree databases, Ramu *et al.* [2012] propose a method that efficiently processes datasets with over 1,000 taxa and hundreds of trees, setting it apart from existing approaches. Furthermore, Rasmussen and Guo [2022] present methods for tree reconciliation, along with tree forests (MAFs), that can be combined and restructured. Additionally, Molloy and Warnow [2019] introduce the `NJMerge` approach, estimat-

ing a consensus tree by analyzing various trees and subtrees. These methods offer several approaches to subtree identification, each with its strengths and weaknesses, providing valuable options for integration into `NMFSt.P`.

4 The Proposed Approach: `NMFst.P`

The `NMFSt.P` Notebook is designed for identifying recurring patterns in phylogenetic tree databases, *e.g.*, frequent subtrees in the topologies of phylogenetic trees. The `NMFSt.P` implements a well-defined workflow composed of seven activities, as depicted in Figure 5. The first four activities are part of the `SciPhy` sub-workflow, previously defined by Ocaña *et al.* [2011].

Initially, a *Sequence Validation* is performed using the Biopython library (`https://biopython.org/`) to ensure that the input files are in the expected format, *i.e.*, FASTA format. After that, the *Multiple Sequence Alignment* activity is invoked. In the current version of `NMFSt.P`, `ClustalW` is defined as the standard tool for performing the alignment, but the notebook allows for the program to be replaced by another by the scientist. The alignment result is saved in a file with ALN extension, which will be later used to construct the phylogenetic tree. Next, the *Evolutionary Model Selection* activity defines the evolutionary model to be used. In the current version, this choice is also made by the Biopython library. By default, the chosen model is Neighbor Joining (NJ), a method for phylogenetic reconstruction from DNA or protein sequences Saitou and Nei [1987], but the user can explore other models if necessary.

In the *Phylogenetic Tree Construction* activity, programs responsible for tree generation are invoked. The user can explore well-known programs such as `RAxML` and `MrBayes` or use the Biopython library. From the generated trees, the process of identifying frequent subtrees can then be initiated. The first activity is *Subtree Generation*, which identifies all possible subtrees in each phylogenetic tree. Each subtree is saved in a file, allowing for finer-grained post-analysis. Algorithm 1 presents the process of generating possible subtrees and is invoked for each phylogenetic tree generated in the *Tree Generation* activity. Algorithm 1 iterates over a set of phylogenetic trees and identifies the possible subtrees in this set. This activity can be parallelized since the identification of subtrees in a specific tree does not depend on others (*e.g.*, bag-of-tasks parallelism [Zhang *et al.*, 2019]).

Once all possible subtrees of each tree have been generated, `NMFSt.P` classifies each subtree by size in the *Subtree Mapping* activity. Additionally, `NMFSt.P` generates a subtree matrix (the procedure of which is presented in Algorithm 2), which will be used in *Similarity Calculation* activity. It is important to emphasize that the generated matrix, *m_subtree*, is a sparse and symmetric matrix, since when we calculate the transpose matrix *m_subtree*, we obtain the matrix *m_subtree* itself.

The entire notebook was implemented in Python and is available on the institutional GitHub repository at the following URL `https://github.com/UFFeScience/NMFSt.P`. Since identifying frequent subtrees can be computing-intensive, `NMFSt.P` has been adapted to leverage the Parsl

Algorithm 1: SubTreeGen - Subtree Generation

```
Input: path ▷ Path of the Phylogenetic Tree
Output: row_subtree ▷ List of generated subtrees paths
tree ← load(path, "nexus") ▷ Load phylogenetic tree in Nexus format
row_subtree ← [] ▷ List with subtree paths
▷ Iterating over all subtrees
foreach clade in find_clades(tree) do
    subtree ← BaseTree.Tree(clade)
    if count_terminals(subtree) > 1 then
        filepath_out ← write(subtree, "nexus") ▷ Saves the subtree in Nexus format and returns the saved subtree path
        row_subtree.append(filepath_out)
    end
end
return row_subtree ▷Returns the list with the paths of the identified subtrees
```

Algorithm 2: SubTreeMatrixGen - Construction of the Subtree Matrix

```
Input: path ▷ Path of identified subtrees
Output: m_subtree ▷ Array with all subtrees
m_subtree ← []
foreach file in path do
    if file.name != nil then
        m_subtree.append(sub_tree(file.path, file.name));
    end
end
return m_subtree;
```

Algorithm 3: SimCalcFST - Similarity Calculation

```
Input: m_subtree ▷ Symmetric array containing the identified subtrees
Output: fst_db ▷ Frequent Subtree Database (FST)
fst_db ← {}
n_column ← num_columns(m_subtree)
n_row ← num_rows(m_subtree)
max_fst ← 0
g_fst ← 0
for i ← 1 to n_row do
    for j ← to n_column do
        for k ← 1 to n_row do
            for l ← to n_column do
                if i != k then
                    if max_fst ≤ sim(m_subtree[i][j], m_subtree[k][l]) then
                        max_fst ← sim(m_subtree[i][j], m_subtree[k][l])
                    end
                    sim_fst ← sim(m_subtree[i][j], m_subtree[k][l])
                    if sim_fst ≥ 1 then
                        fst_db[g_fst][m_subtree[i][j].append(m_subtree[k][l])
                        g_fst ← g_fst + 1
                    end
                end
            end
        end
    end
end
return fst_db;
```

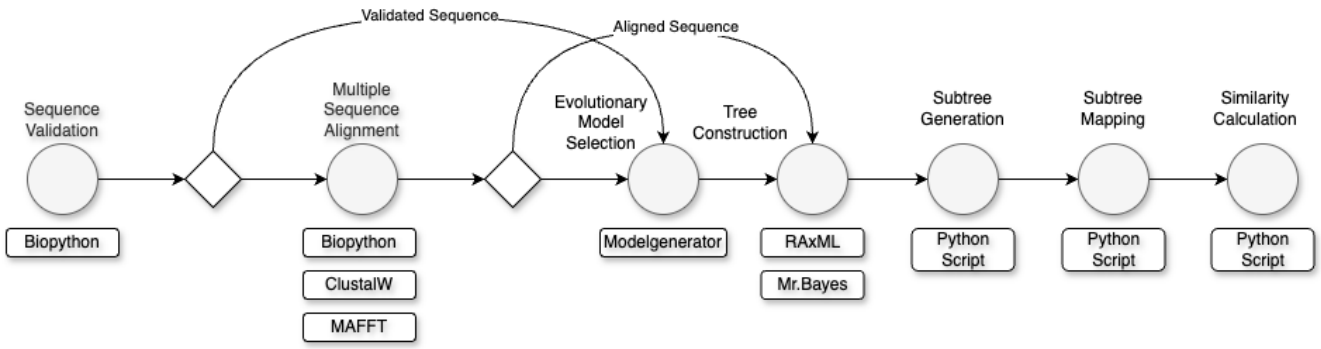

Figure 5. The Workflow executed by `NMFSt.P`.

library, enabling parallel execution of the notebook. It is important to note that the public version of `NMFSt.P` does not yet include parallel capabilities, as we are currently adapting the script to seamlessly function across various environments, including local computers, clouds, and clusters. Using Parsl, scientists can indicate where the code should be executed in parallel using annotations, known as *decorators*. These annotations, implemented as Annotated Functions (*i.e.*, *apps*), allow for the concurrent execution of identified apps while respecting the data dependencies within the workflow implemented in the notebook. For instance, `@python_app` decorators can be added to subtree generation and similarity calculation activities. This approach enhances the efficiency of `NMFSt.P` by leveraging parallel computing capabilities to speed up the analysis process.

5 Experimental Evaluation

In this section, we provide an experimental evaluation of `NMFSt.P`, analyzing the results from two perspectives: (i) biological and (ii) computational. From a biological standpoint, the experiment aims to verify whether `NMFSt.P` can effectively identify frequent subtrees from a database of phylogenetic trees. On the computational front, the experiment aims to evaluate the performance of parallel execution in `NMFSt.P` and compare it against a baseline.

5.1 Environment Setup

For the experiments conducted in this manuscript, both `NMFSt.P` and the approach proposed by Guedes *et al.* [2017], *i.e.*, `SciPhyloMiner`, were executed on Google Cloud Platform (GCP). GCP stands out as one of the leading cloud providers in the market. Among the various types of virtual machines offered by GCP, we opted to use the `c3d-standard-60` type. This virtual machine configuration has 60 vCPUs, 120 GB of RAM, and 20 Gbps of bandwidth, and it costs US$ 1.0651 per hour in the spot market. Such specifications suit CPU-bound workloads that do not require large memory allocations, aligning with the workflow requirements executed by `NMFSt.P`.

Regarding software, the virtual machine is configured with the following versions: ClustalW version 2.1 for sequence alignment and programs RAxML 7.2.8 and MrBayes 3.2.6 for phylogenetic tree generation. Additionally, Biopython version 1.81 is included to support various bioinformatics tasks.

5.2 Experiment Setup

The dataset used in our experiments comprises 200 multi-fasta files. We selected a subset of multi-fasta files containing protein sequences from orthologous genes found in malaria-causing parasites for our input dataset. This dataset, obtained in fasta format, was sourced from `RefSeq` (`ftp://ftp.ncbi.nih.gov/refseq/release/protozoa/`), as defined by Ocaña and Dávila [2011].

5.3 Results Discussion

From a biological standpoint, `NMFSt.P` produced the same trees as generated by the baseline approach, *i.e.*, `SciPhyloMiner` differing only in the output format. While the output type generated by `NMFSt.P` is a database of similarities, the baseline approach outputs a list of subtrees with a frequency above a user-defined threshold θ. Overall, the analyzed genes presented similar phylogenetic trees using both `RAxML` and `MrBayes` as presented in Figure 6, indicating consistent relationships among taxa, achieving the same conclusion obtained by Guedes *et al.* [2017], the baseline approach.

From a computational perspective, we ran `NMFSt.P` on a `c3d-standard-60` virtual machine, varying the number of cores used in each execution. Additionally, we evaluated the baseline approach's performance, which was run in the same environment but used a parallel workflow system, *i.e.*, `SciCumulus`, to execute the workflow.

Figure 7(a) illustrates the makespan of both `NMFSt.P` and the baseline, measured in hours for each different configuration used. It is evident that for both approaches, the total makespan decreases as more cores are added to the execution, as expected. However, `SciPhylominer` demonstrates superior performance compared to `NMFSt.P`. This difference in performance can be attributed to the different execution engines used by each approach. While `SciPhylominer` executes on top of `SciCumulus`, a workflow system optimized for cloud environments, `NMFSt.P` runs on top of the Jupyter Notebook engine. As noted by Colonnelli *et al.* [2022], the overhead introduced by the interactive execution mode in notebooks may become non-negligible in specific scenarios.

We analyzed the speedup (Figure 7(b)) and the parallel efficiency (Figure 7(c)) for both `NMFSt.P` and

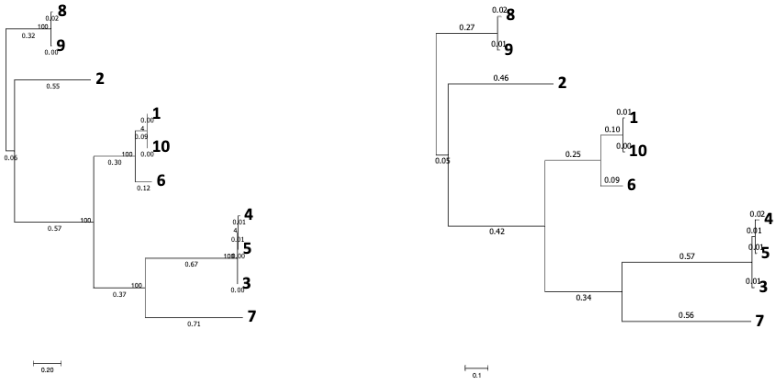

Figure 6. Phylogenetic trees created by `RAxML` and `Mr.Bayes` within `NMFSt.P`.

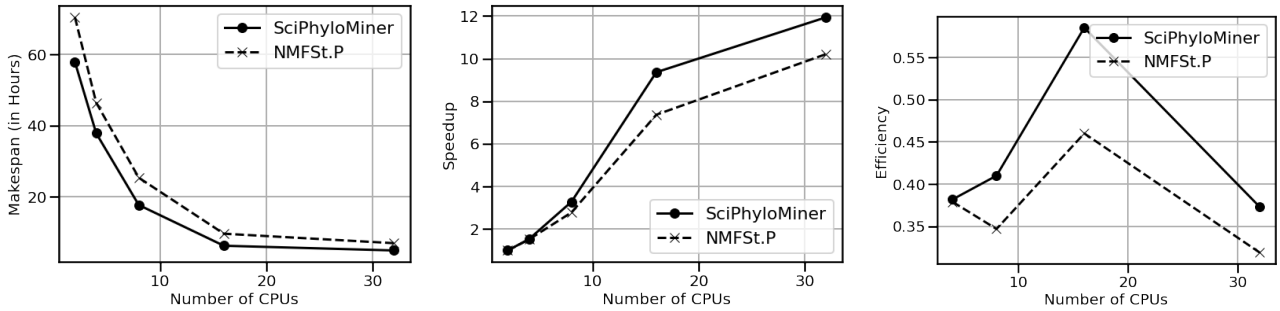

Figure 7. Analysis of (a) Makespan, (b) Speedup and (c) Efficiency for both `NMFSt.P` and `SciPhylomics`.

`SciPhylominer`. We observed that the speedup, defined as the ratio of computing time for sequential execution to that for parallel execution, deviates significantly from linear speedup for both approaches. This deviation is primarily attributed to segments of the workflow that are not parallelizable, commonly referred to as blocking activities, requiring sequential execution even with multiple available cores. Additionally, the optimal efficiency, calculated as the speedup divided by the number of execution units (in this context, vCPUs) for both approaches, was attained using 16 vCPUs. This aspect highlights an area for potential improvement in future versions of `NMFSt.P`.

6 Conclusions and Future Work

Currently, many experiments in the field of bioinformatics generate large databases of phylogenetic trees. Analyzing such phylogenetic trees and identifying frequent subtrees can indicate the presence of evolutionary patterns. However, this task is not straightforward as it may require comparing hundreds of trees. In this manuscript, we introduce a Jupyter Notebook called `NMFSt.P`, which aims to provide a user-friendly workflow for identifying frequent subtrees in a database of phylogenetic trees. `NMFSt.P` was evaluated through a case study that identified frequent subtrees in trees generated from 200 multi-fasta files of malaria-causing parasites. The results were equivalent to those of the baseline approach, both in biological and computational terms, with the advantage of user-friendly accessibility. Future work includes integrating provenance capture tools into `NMFSt.P` using `DfAnalyzer` provenance library [Silva *et al.*, 2020] and conducting experiments with larger datasets to evaluate scalability.

Acknowledgements

We acknowledge using the AI-based tools Grammarly and ChatGPT for grammar checking. We carefully examined and often corrected AI suggestions, whereby we took full responsibility for the form and content of the manuscript.

Funding

This work was supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. The authors would also like to thank CNPq (grants 311898/2021-1 and 316625/2021-3) and FAPERJ (grant E-26/202.806/2019) for their financial support.

Authors' Contributions

All authors contributed to the design, writing, and revision. João Moraes and Camila Ferrari were responsible for implementing the current version of `NMFSt.P` and conducting the experiments.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

The datasets generated and analyzed during the current study are available in https://github.com/UFFeScience/NMFSt.P.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410. DOI: https://doi.org/10.1016/S0022-2836(05)80360-2.

Amir, A. and Keselman, D. (1997). Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms. *SIAM Journal on Computing*, 26(6):1656–1669. DOI: 10.1137/S0097539794269461.

Babuji, Y., Woodard, A., Li, Z., Katz, D. S., Clifford, B., Kumar, R., Lacinski, L., Chard, R., Wozniak, J. M., Foster, I., Wilde, M., and Chard, K. (2019). Parsl: Pervasive parallel programming in python. In *28th ACM International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. babuji19parsl.pdf. DOI: 10.1145/3307681.3325400.

Bryant, D. (2003). *A classification of consensus methods for phylogenetics*, pages 163–183. DOI: 10.1090/dimacs/061/11.

Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T. J., Higgins, D. G., and Thompson, J. D. (2003). Multiple sequence alignment with the Clustal series of programs. *Nucleic Acids Research*, 31(13):3497–3500. DOI: 10.1093/nar/gkg500.

Colonnelli, I., Aldinucci, M., Cantalupo, B., Padovani, L., Rabellino, S., Spampinato, C., Morelli, R., Di Carlo, R., Magini, N., and Cavazzoni, C. (2022). Distributed workflows with jupyter. *Future Generation Computer Systems*, 128:282–298. DOI: https://doi.org/10.1016/j.future.2021.10.007.

de O. Sandes, E. F., Miranda, G., Melo, A. C., Martorell, X., and Ayguade, E. (2014). Fine-grain parallel megabase sequence comparison with multiple heterogeneous gpus. *SIGPLAN Not.*, 49(8):383–384. DOI: 10.1145/2692916.2555280.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. (2012). A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552. DOI: 10.1007/S10723-012-9227-2.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers. DOI: 10.2200/S00915ED1V01Y201904DTM060.

Deepak, A. and Fernández-Baca, D. (2014). Enumerating all maximal frequent subtrees in collections of phylogenetic trees. *Algorithms for Molecular Biology*, 9(1):16. DOI: 10.1186/1748-7188-9-16.

Deepak, A. *et al*. (2014). Evominer: frequent subtree mining in phylogenetic databases. *Knowledge and Information Systems*, 41(3):559–590. DOI: 10.1007/s10115-013-0676-0.

Do, C. B., Brudno, M., and Batzoglou, S. (2004). PROBCONS: probabilistic consistency-based multiple alignment of amino acid sequences. In McGuinness, D. L. and Ferguson, G., editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 703–708. AAAI Press / The MIT Press.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. J. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press. DOI: 10.1017/CBO9780511790492.

Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797. DOI: 10.1093/nar/gkh340.

Farris, J. S. (1970). Methods for Computing Wagner Trees. *Systematic Biology*, 19(1):83–92. DOI: 10.1093/sysbio/19.1.83.

Ferrari, C., Moraes, J. V., and de Oliveira, D. (2023). Nmfst.p: um notebook para identificação em paralelo de subárvores frequentes em conjuntos de Árvores filogenéticas. In *Anais do XVII Brazilian e-Science Workshop*, pages 1–8, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/bresci.2023.234110.

Goloboff, P. A. *et al*. (2009). Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics*, 25(3):211–230. DOI: https://doi.org/10.1111/j.1096-0031.2009.00255.x.

Guedes, T., Ocaña, K., and de Oliveira, D. (2017). Sciphylominer: um workflow para mineração de dados filogemônicos de protozoários. In *Anais do XI Brazilian e-Science Workshop*, pages 69–76, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/bresci.2017.9924.

Katoh, K., Rozewicki, J., and Yamada, K. D. (2017). MAFFT online service: multiple sequence alignment, interactive sequence choice and visualization. *Briefings in Bioinformatics*, 20(4):1160–1166. DOI: 10.1093/bib/bbx108.

Markel, S. and Leon, D. (2003). *Sequence analysis in a nutshell - a guide to common tools and databases: covers EMBOSS 2.5.0*. O'Reilly.

Molloy, E. K. and Warnow, T. (2019). TreeMerge: a new method for improving the scalability of species tree estimation methods. *Bioinformatics*, 35(14):i417–i426. DOI: 10.1093/bioinformatics/btz344.

Nixon, K. C. (2001). Phylogeny. In Levin, S. A., editor, *Encyclopedia of Biodiversity (Second Edition)*, pages 16–23. Academic Press, Waltham, second edition edition. DOI: https://doi.org/10.1016/B978-0-12-384719-5.00108-8.

Notredame, C., Higgins, D. G., and Heringa, J. (2000). T-coffee: a novel method for fast and accurate multiple sequence alignment11edited by j. thornton. *Journal of Molecular Biology*, 302(1):205–217. DOI: https://doi.org/10.1006/jmbi.2000.4042.

Ocaña, K. and de Oliveira, D. (2015). Parallel computing in genomic research: advances and applications. *Adv. Appl. Bioinform. Chem.*, page 23.

Ocaña, K. A. C. S. *et al*. (2011). Sciphy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Proc. of the 6th Brazilian Symposium on Bioinformatics*, pages 66–70. Springer. DOI: 10.1007/978-3-642-22825-4_9.

Ocaña, K. A. and Dávila, A. M. (2011). Phylogenomics-based reconstruction of protozoan species tree. *Evolutionary Bioinformatics*, 7:EBO.S6861. PMID: 21863127. DOI: 10.4137/EBO.S6861.

Puigbò, P., Wolf, Y. I., and Koonin, E. V. (2019). *Genome-Wide Comparative Analysis of Phylogenetic Trees: The Prokaryotic Forest of Life*, pages 241–269. Springer New York, New York, NY. DOI: 10.1007/978-1-4939-9074-0_8.

Ramu, A., Kahveci, T., and Burleigh, J. G. (2012). A scalable method for identifying frequent subtrees in sets of large phylogenetic trees. *BMC Bioinformatics*, 13(1):256. DOI: 10.1186/1471-2105-13-256.

Rasmussen, D. A. and Guo, F. (2022). Espalier: Efficient tree reconciliation and arg reconstruction using maximum agreement forests. *bioRxiv*. DOI: 10.1101/2022.01.17.476639.

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

Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler, D., and Miller, W. (2003). Human–mouse alignments with BLASTZ. *Genome Res.*, 13(1):103–107.

Setubal, J. C. and Meidanis, J. (1997). *Introduction to computational molecular biology*. PWS Publishing Company.

Silva, V., Campos, V., Guedes, T., Camata, J. J., de Oliveira, D., Coutinho, A. L. G. A., Valduriez, P., and Mattoso, M. (2020). Dfanalyzer: Runtime dataflow analysis tool for computational science and engineering applications. *SoftwareX*, 12:100592. DOI: 10.1016/J.SOFTX.2020.100592.

Sukumaran, J. and Holder, M. T. (2010). DendroPy: a python library for phylogenetic computing. *Bioinformatics*, 26(12):1569–1571.

Tommy Tsan-Yuk Lam, C.-C. H. and Tang, J. W. (2010). Use of phylogenetics in the molecular epidemiology and evolutionary studies of viral infections. *Critical Reviews in Clinical Laboratory Sciences*, 47(1):5–49. DOI: 10.3109/10408361003633318.

Vilella, A. J., Severin, J., Ureta-Vidal, A., Heng, L., Durbin, R., and Birney, E. (2009). Ensemblcompara genetrees: Complete, duplication-aware phylogenetic trees in vertebrates. *Genome research*, 19 2:327–35.

Zhang, Y., Zhou, J., and Sun, J. (2019). Scheduling bag-of-tasks applications on hybrid clouds under due date constraints. *Journal of Systems Architecture*, 101:101654. DOI: https://doi.org/10.1016/j.sysarc.2019.101654.