`NMFSt.P`: um *Notebook* para Identificação em Paralelo de Subárvores Frequentes em Conjuntos de Árvores Filogenéticas*

Camila Ferrari[1], **João Vitor Moraes**[1], **Daniel de Oliveira**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)

{camila_ferrari,joaovitormoraes}@id.uff.br, danielcmo@ic.uff.br

Resumo. *A análise exploratória de informações evolutivas em árvores filogenéticas é uma tarefa importante no contexto da bioinformática. Tal análise depende em muitos casos da identificação de subárvores frequentes em um conjunto de árvores filogenéticas de entrada. Essa identificação pode ser uma tarefa computacionalmente intensiva e laboriosa, dependendo do tamanho do conjunto de árvores de entrada. Nesse artigo apresentamos o Notebook* `NMFSt.P`*, que permite a comparação de múltiplas árvores filogenéticas para a identificação de subárvores frequentes. Experimentos realizados mostraram que o* `NMFSt.P` *conseguiu gerar resultados similares a abordagem baseline ao mesmo tempo em que apresentou melhoria de desempenho de até 68,31% no tempo de execução com o uso de um maior número de vCPUs na nuvem.*

Abstract. *The exploratory analysis of evolutionary information in phylogenetic trees is an important task in the context of bioinformatics. This analysis often relies on identifying frequent subtrees within a set of input phylogenetic trees. This identification can be a computing-intensive task, depending on the size of the input tree set. In this paper, we introduce the* `NMFSt.P` *Notebook, which enables the comparison of multiple phylogenetic trees to identify frequent subtrees. Conducted experiments demonstrated that* `NMFSt.P` *is able to generate results similar to the baseline while also improving performance by up to 68.31% in execution time when using a greater number of vCPUs in the cloud.*

1. Introdução

Devido ao rápido avanço de tecnologias que integram a área biológica com a Ciência da Computação (*e.g.*, técnicas de sequenciamento aliadas a computação de alto desempenho), temos presenciado um aumento considerável no volume de dados biológicos disponíveis. Na área da filogenia, tais avanços permitiram a geração de árvores filogenéticas grandes [Goloboff et al. 2009], seja a partir da execuções de aplicações isoladas ou a partir de *workflows* complexos [Guedes et al. 2017]. A análise de tais árvores desempenha um papel fundamental na compreensão das relações evolutivas entre os seres vivos, oferecendo insumos para tarefas posteriores.

Existem diversos programas que podem ser usados para a criação de árvores filogenéticas, onde cada um aplica um método específico *e.g.*, máxima parsimônia e máxima verossimilhança [Felsenstein 1983]. Como cada programa/método apresenta características próprias, um mesmo *dataset* de entrada pode produzir diferentes relações

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Os autores gostariam ainda de agradecer ao CNPq (*grant* 311898/2021-1) e FAPERJ (*grant* E-26/202.806/2019) pelo apoio financeiro.

dependendo do programa/método adotado [Puigbò et al. 2019]. A escolha do programa/método mais adequado para um determinado *dataset* de entrada pode não ser trivial, pois depende de vários fatores como a quantidade de sequências, tamanhos das sequências, *etc*. Dessa forma, os especialistas costumam explorar os programas/métodos existentes em suas análises, gerando assim múltiplas árvores filogenéticas. Extrair conhecimento útil desse conjunto de árvores se faz então necessário.

Existem abordagens que se propõem a obter uma árvore de consenso [Bryant 2003] a partir de um conjunto de árvores filogenéticas. No entanto, esse tipo de árvore pode não ser capaz de identificar subárvores frequentes, que são importantes para revelar padrões evolutivos. Apesar de oferecerem *insights* importantes, encontrar as subárvores frequentes é um problema NP-difícil [Amir and Keselman 1997], e pode envolver a comparação de várias centenas de árvores. A identificação de subárvores frequentes pode ser considerado um problema de larga escala, e em muitos casos é modelada por meio de *workflows* e sistemas de *workflows* [de Oliveira et al. 2019]. Entretanto, a utilização de sistemas de *workflows* requer uma curva de aprendizado não trivial, já que cada sistema existente possui *features* próprias que o usuário deve conhecer para modelar e executar seu *workflow*. Como esses *workflows* muitas vezes precisam executar em ambientes de alto desempenho (*e.g.*, *clusters* e nuvens de computadores), isso acaba inserindo mais uma complexidade, o que pode acabar limitando a sua utilização.

De forma a estruturar o *workflow* de identificação de subárvores frequentes sem a necessidade do uso de complexos sistemas de *workflow* ou o encadeamento manual de múltiplas aplicações por parte do usuário, esse artigo apresenta o `NMFSt.P` (do inglês *Notebook for Mining Frequent Subtrees in Parallel*), um *Notebook* que visa a identificação de padrões comuns, *i.e.,* subárvores nas topologias de árvores dadas como entrada. O `NMFSt.P` estende trabalhos anteriores do grupo que modelaram *workflows* de análise filogenética e mineração de subárvores [Guedes et al. 2017, Ocaña et al. 2011] usando sistemas de *workflows*. O `NMFSt.P` foi implementado em *Python* e utiliza a biblioteca *Parsl* [Babuji et al. 2019] para prover o paralelismo necessário para a tarefa. Ao utilizar o `NMFSt.P`, o usuário é capaz de identificar a partir das árvores produzidas, quais são as subárvores compartilhadas entre diferentes árvores filogenéticas. Na avaliação da abordagem proposta, analisamos um conjunto de 50 árvores filogenéticas de genes de protozoários que foram construídas com diferentes programas de geração de árvores filogenéticas, e os resultados se mostraram promissores.

Este artigo se encontra organizado em quatro seções além da Introdução. A Seção 2 discute trabalhos relacionados. A Seção 3 detalha o *Notebook* `NMFSt.P`. A Seção 4 apresenta a avaliação experimental, e, finalmente, a Seção 5 conclui o artigo e apresenta trabalhos futuros.

2. Trabalhos Relacionados

Foram encontrados poucos trabalhos na literatura que propõem *workflows* para identificação de subárvores frequentes em conjuntos de árvores filogenéticas. [Guedes et al. 2017] propõem o *SciPhyloMiner*, que é um *workflow* modelado em um sistema de *workflow* paralelo (*i.e.*, SciCumulus) que utiliza a aplicação *Dendropy* para mineração das subárvores. A limitação do *SciPhyloMiner* é justamente em sua capacidade de paralelismo, já que a aplicação *Dendropy* é uma caixa-preta não paralelizável. [Vilella et al. 2009] propõem o *workflow* intitulado *EnsemblCompara GeneTrees* que re-

aliza agrupamento, alinhamento múltiplo de sequências e geração de árvore. O *workflow* identifica subárvores frequentes em grandes famílias de genes, possibilitando o estudo de relações evolutivas entre sequências de genes em diferentes espécies. Diferentemente do `NMFSt.P`, que foi projetado como um *Notebook* de forma a facilitar o seu uso e extensão, nos trabalhos anteriormente citados, o usuário fica restrito aos sistemas de *workflows* utilizados, o que pode limitar a sua utilização em diferentes contextos.

Além dos *workflows*, existem diversos trabalhos na literatura que propõem métodos e ferramentas para identificação de subárvores frequentes em árvores filogenéticas, e que podem ser incorporados ao `NMFSt.P`. [Deepak et al. 2014] propõem o algoritmo *Evominer*, que é um algoritmo paralelo para identificação de subárvores frequentes. Os autores comparam o desempenho do *Evominer* com outras soluções do estado da arte e o mesmo apresentou um desempenho superior em duas ordens de grandeza. [Deepak and Fernández-Baca 2014] propõem o *MfstMiner*, que é uma ferramenta para identificar todas as subárvores frequentes máximas em coleções de árvores filogenéticas. Uma subárvore frequente máxima é aquela que possui o maior número possível de folhas. [Ramu et al. 2012] abordam os desafios envolvendo a identificação de subárvores de acordo frequente máximo (MFASTs) em uma coleção de árvores filogenéticas dadas como entrada. Diferentemente das abordagens existentes, a proposta de [Ramu et al. 2012] consegue realizar o processamento de forma eficiente com conjuntos de dados com mais de 1.000 táxons e centenas de árvores. [Rasmussen and Guo 2022] apresentam métodos para reconciliação de árvores, bem como as florestas de árvores (MAFs) que podem ser combinadas e reestruturadas. [Molloy and Warnow 2019] propõem a abordagem `NJMerge` que tem como foco estimar uma árvore de consenso a partir da análise de diversas árvores e subárvores.

3. Abordagem Proposta: o *Notebook* `NMFSt.P`

O *Notebook* `NMFSt.P` realiza a identificação de padrões recorrentes, *i.e.*, subárvores frequentes nas topologias de árvores filogenéticas. O `NMFSt.P` implementa um *workflow* bem definido composto de oito atividades, conforme apresentado na Figura 1. As quatro primeiras atividades são parte do *sub-workflow SciPhy* [Ocaña et al. 2011].

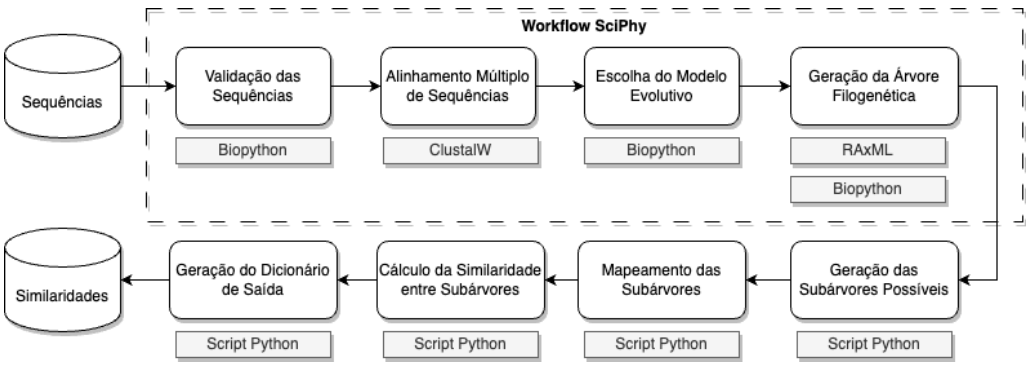

Figura 1. *Workflow* executado pelo *Notebook* `NMFSt.P`.

Inicialmente é executada uma *Validação de Sequências*, por meio da biblioteca *Biopython* (`https://biopython.org/`), de forma a garantir que os arquivos de entrada sejam válidos. Após, a atividade de *Alinhamento Múltiplo de Sequências* é executada. Na versão atual do `NMFSt.P` o *ClustalW* [Thompson et al. 1994] é utilizado

para realizar o alinhamento, mas o *notebook* permite a troca de programa com poucas adaptações por parte do usuário. O resultado do alinhamento é salvo em um arquivo de extensão ALN, que será posteriormente usado para a construção da árvore filogenética. A seguir, a atividade *Escolha do Modelo Evolutivo* define o modelo evolutivo a ser utilizado. Na versão atual, essa escolha é também realizada pela biblioteca *Biopython*. Por padrão, o modelo escolhido é o *Neighbor Joining* (NJ), um método de reconstrução filogenética a partir de sequências de DNA ou proteínas,[Saitou and Nei 1987], mas o usuário pode explorar outros modelos, caso seja necessário. Na atividade *Geração da Árvore Filogenética*, os programas responsáveis pela geração da árvore são invocados. O usuário pode explorar programas conhecidos como o *RAxML* e o *MrBayes*, ou utilizar a biblioteca *Biopython*. A partir das árvores geradas, o processo de identificação das subárvores frequentes pode ser então inciado. A primeira atividade é a *Geração das Subárvores Possíveis*, que identifica em cada árvore filogenética as subárvores possíveis existentes. Cada subárvore é salva em um arquivo, permitindo análises *a posteriori* com granularidade mais fina. O Algoritmo 1 apresenta o processo de geração das subárvores possíveis, e é invocado para cada árvore gerada na etapa anterior. Essa etapa pode ser paralelizada, uma vez que a identificação das subárvores de uma árvore específica não depende das demais (*i.e.*, paralelismo *bag-of-tasks*).

Algoritmo 1: SUBTREEGEN - Geração de Subárvores

```
   Input: path ▷ Caminho da Árvore Filogenética
   Output: row_subtree ▷ Lista de caminhos das subárvores geradas
 1 tree ← load(path, "nexus") ▷ Carrega árvore filogenética no formato Nexus
 2 row_subtree ← [] ▷ Lista com caminhos das subárvores
 3 ▷ Iteração sobre todas as subárvores
 4 foreach clade in find_clades(tree) do
 5 |   subtree ← BaseTree.Tree(clade)
 6 |   if count_terminals(subtree) > 1 then
 7 |   |   filepath_out ← write(subtree, "nexus") ▷ Salva a subárvore no formato Nexus e
   |   |     retorna o caminho da subárvore salva
 8 |   |   row_subtree.append(filepath_out)
 9 |   end
10 end
11 return row_subtree ▷ Retorna a lista com os caminhos das subárvores identificadas
```

Uma vez que todas as possíveis subárvores de cada árvore foram identificadas, o `NMFSt.P` classifica cada subávore por tamanho na atividade *Mapeamento das Subárvores*. Além disso, o `NMFSt.P` gera uma matriz de subárvores (cujo procedimento é apresentado no Algoritmo 2), que será utilizada no cálculo de similaridade. É importante ressaltar que a matriz gerada *m_subtree* é uma matriz esparsa e simétrica, já que quando calculamos a matriz transposta $m\_subtree^t$ encontramos a própria matriz *m_subtree*.

Com a classificação das subárvores realizadas, o `NMFSt.P` pode então calcular o grau de similaridade entre cada par de subárvores na atividade *Cálculo de Similaridade entre Subárvores*. A similaridade entre as árvores pode ser representada por valores que variam de 0 a 1 ou de 0 a 100, dependendo da convenção adotada. Quão mais próximo de 1 ou 100, maior a similaridade entre as sequências. No `NMFSt.P` a similaridade, dada por *grau_maf*, é normalizada para o intervalo 0..1, onde 0 é totalmente discordante e 1 totalmente concordante. Essa atividade produz como saída um banco de dados contendo o grau de similaridade entre os pares de subárvores, conforme apresentado no Algoritmo

3. A partir das similaridades obtidas para cada comparação de pares de subárvores, o usuário pode identificar quais são as subárvores frequentes por meio de uma consulta ao banco de dados *fst_db* gerado como saída. É importante ressaltar que é possível salvar as subárvores em formato *Newick/NEXUS* para posterior visualização no *notebook* proposto.

Algoritmo 2: SUBTREEMATRIXGEN - Construção da Matriz de Subárvores

```
Input: path ▷ Caminho das subárvores identificadas
Output: m_subtree ▷ Matriz com todas as subárvores
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;
```

Algoritmo 3: SIMCALCFST - Cálculo da Similaridade entre Subárvores

```
Input: m_subtree ▷ Matriz simétrica contendo as subárvores identificadas
Output: fst_db ▷ Banco de dados de subárvores frequentes (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;
```

Todo o *notebook* foi implementado na linguagem Python e se encontra em processo de disponibilização no GitHub institucional `https://github.com/UFFeScience/NMFSt.P`. Como o processo de identificação das subárvores frequentes pode ser computacionalmente intensivo, o `NMFSt.P` foi adaptado para utilizar a biblioteca *Parsl* de forma a prover a execução paralela do *notebook*. A ideia principal é permitir que o usuário anote onde o *Parsl* deve executar o código *Python* em paralelo. Essa anotação (chamada de *decorator*) é realizada usando Funções Anotadas (*i.e.*, *apps*). Os *Apps* identificados são executados concorrentemente, respeitando as dependências de

dados do *workflow* implementado no *notebook*. Foram adicionados *decorators* do tipo `@python_app` nas atividades de geração das subárvores e no cálculo de similaridade. A versão com recursos de paralelização ainda se encontra em processo de disponibilização.

4. Avaliação Experimental do *Notebook* `NMFSt.P`

Nesta seção, apresentamos uma avaliação experimental do *notebook* `NMFSt.P`, analisando os resultados a partir de duas dimensões: (i) biológica e (ii) computacional. Na dimensão biológica, o objetivo do experimento é verificar se o `NMFSt.P` é capaz de identificar as subárvores frequentes a partir de um *dataset* de árvores filogenéticas. Na dimensão computacional, o experimento visa analisar do desempenho da execução paralela do `NMFSt.P` e compará-lo com um *baseline*. No experimento executado, utilizamos o *ClustalW* versão 2.1 como programa de alinhamento e os programas *RAxML* 7.2.8 e *MrBayes* 3.2.6 para geração das árvores filogenéticas, além do *Biopython* 1.81. Como *dataset* de entrada, utilizamos um subconjunto de arquivos multi-fasta de sequências de proteínas de genes ortólogos do *dataset* definido por [Ocaña and Dávila 2011].

A ideia por trás da utilização de tal *dataset* é que assim podemos comparar os resultados obtidos com uma abordagem *baseline* [Guedes et al. 2017], tanto em termos de desempenho computacional quanto do ponto de vista biológico. O *dataset* original era composto de 200 arquivos multi-fasta, porém nos experimentos apresentados nesse artigo, selecionamos 50 arquivos multi-fasta, onde cada arquivo é relacionado a um determinado gene. Os experimentos foram executados na nuvem Amazon AWS (`https://aws.amazon.com/pt/`) utilizando máquinas virtuais dos tipos *c5a.2xlarge* (8 vCPUs e 16 GiB de RAM), *c5a.4xlarge* (16 vCPUs e 32 GiB de RAM), *c5a.8xlarge* (32 vCPUs e 64 GiB de RAM) e a *c5a.16xlarge* (64 vCPUs e 128 GiB de RAM).

Do ponto de vista biológico, o `NMFSt.P` produziu como saída as mesmas árvores geradas pela abordagem *baseline*, diferindo apenas no formato de saída. Enquanto que o tipo de saída gerado pelo `NMFSt.P` é um banco de dados de similaridades, a abordagem *baseline* gera como saída uma lista das subárvores que possuem uma frequência acima de um *threshold* θ definido pelo usuário. Em geral, os genes analisados apresentaram árvores filogenéticas semelhantes tanto usando o *RAxML* quanto o *MrBayes*, o que indica relações consistentes entre os táxons. A mesma conclusão foi alcançada pela abordagem *baseline*.

Do ponto de vista computacional, executamos o *Notebook* proposto no ambiente da Amazon AWS variando o tipo de máquina virtual. Além disso, avaliamos o desempenho da abordagem *baseline* que utiliza um sistema de *workflow* paralelo (*i.e.*, SciCumulus) para executar o *workflow*. A Figura 2 apresenta o *makespan* tanto do `NMFSt.P` quanto do *baseline*, em minutos para cada tipo diferente de máquina virtual utilizado. Observa-se, conforme esperado, uma diminuição do tempo de execução sempre que há a disponibilidade de mais vCPUs na máquina virtual. Por exemplo, utilizando o `NMFSt.P` o *makespan* foi reduzido de 155,40 minutos (utilizando a máquina virtual *c5a.2xlarge*) para 49,24 minutos (utilizando a máquina virtual *c5a.16xlarge*), representando melhorias de desempenho de até 68,31%. Entretanto, apesar da máquina virtual do tipo *c5a.16xlarge* possuir quatro vezes mais vCPUs e memória RAM do que a máquina virtual do tipo *c5a.2xlarge*, o desempenho não melhorou na mesma proporção em que aumentamos a quantidade de recursos. É importante perceber também ao analisar a Figura 2 que o `NMFSt.P` apresentou um *makespan* ligeiramente menor do que a aborda-

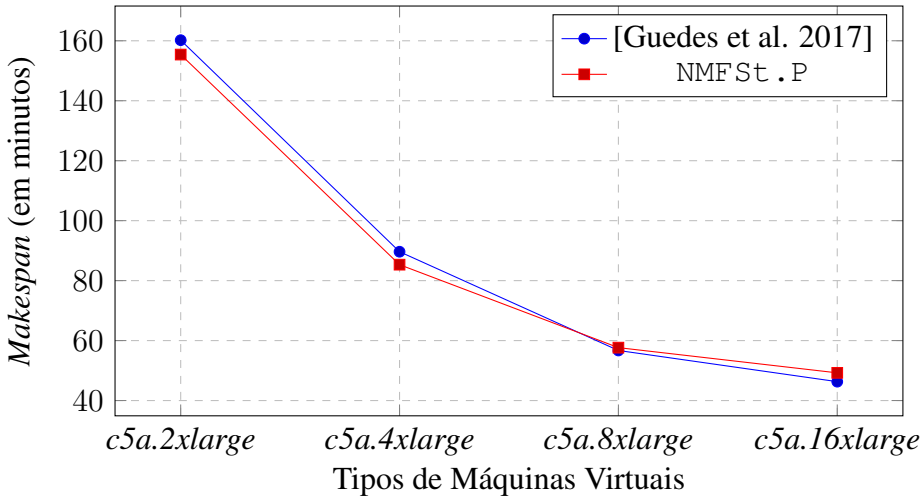

Figura 2. Comparação do *makespan* (em minutos) do `NMFSt.P` e da abordagem *baseline* na Amazon AWS.

gem *baseline* com execuções de até 16 vCPUs, mas com um maior número de vCPUs, houve um acréscimo em relação ao *makespan* da abordagem *baseline*. Tal comportamento está sendo investigado no momento. Vale ressaltar que o `NMFSt.P` apresentou uma variação de desempenho em comparação com a abordagem de *baseline* de -2,9% na máquina 2xlarge, -4,8% na máquina 4xlarge, +1,6% na máquina 8xlarge e +6,3% na máquina 16xlarge.

5. Conclusão

Atualmente, muitos experimentos na área de bioinformática produzem grandes conjuntos de árvores filogenéticas. Analisar tais árvores e identificar subárvores frequentes pode indicar a existência de padrões evolutivos. Entretanto, essa não é uma tarefa simples, pois pode demandar a comparação de centenas de árvores. Neste artigo, apresentamos o *Notebook Python* `NMFSt.P` que tem como objetivo prover um *workflow* de fácil utilização por parte do usuário para identificar subárvores frequentes em conjuntos de árvores filogenéticas. O `NMFSt.P` foi avaliado por meio de um estudo de caso que identificou as subárvores frequentes em um conjunto de árvores geradas a partir de 50 arquivos multifasta. Os resultados se mostraram equivalentes aos da abordagem *baseline* tanto em termos biológicos quanto em termos computacionais, com a facilidade do acesso por parte do usuário. Trabalhos futuros incluem o acoplamento de ferramentas de captura de proveniência ao `NMFSt.P` e a execução de experimentos com *datasets* maiores.

Referências

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.

Babuji, Y. N. et al. (2019). Parsl: Pervasive parallel programming in python. In Weissman, J. B., Butt, A. R., and Smirni, E., editors, *Proc. of the 28th HPDC*, pages 25–36. ACM.

Bryant, D. (2003). *A classification of consensus methods for phylogenetics*, pages 163–183.

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.

Deepak, A. et al. (2014). Evominer: frequent subtree mining in phylogenetic databases. *Knowledge and Information Systems*, 41(3):559–590.

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

Felsenstein, J. (1983). Statistical inference of phylogenies. *Journal of the Royal Statistical Society. Series A (General)*, 146(3):246–272.

Goloboff, P. A. et al. (2009). Phylogenetic analysis of 73 060 taxa corroborates major eukaryotic groups. *Cladistics*, 25(3):211–230.

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.

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.

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.

Ocaña, K. A. and Dávila, A. M. (2011). Phylogenomics-based reconstruction of protozoan species tree. *Evol Bioinform Online*, 7:107–121.

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.

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.

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

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

Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). ClustalW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680.

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.