Bioinformatics
doi.10.1093/bioinformatics/xxxxxx
Advance Access Publication Date: Day Month Year
Manuscript Category

OXFORD

Library of efficient algorithms for phylogenetic analysis

Luana Silva [1,2,*]

[1] Instituto Superior Técnico, Lisbon, 1049-001, Portugal, and [2] INESC-ID, Lisbon, 1000-029, Portugal

* To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

Motivation: With growing exchanges of people and merchandise between countries, epidemics have become an issue of increasing importance, thus epidemiological surveillance is now a global procedure rather than a country-based one. Combining information of country-specific datasets can now reveal epidemic spreading patterns that were not possible to detect before, but phylogenetic algorithms are often hard to use and integrate in analysis frameworks and tools.
Results: The developed library is a command line application that conforms to the phylogenetic analysis workflow and is highly performant, extensible, reusable, and portable. It differs from other existing tools as it was built to be continuously extended and not just serve a single purpose. It enables reading datasets, distance matrices, and phylogenetic trees from files, calculating and correcting distance matrices, inferring and locally optimizing phylogenetic trees, and writing distance matrices and phylogenetic trees to files.
Availability: The code of the developed library is publicly available at https://github.com/Luanab/phylolib along with its documentation, and for testing purposes, this library is hosted in a server as a Docker image. A fully extended version of this article is available at http://arxiv.org/abs/2012.12697.
Contact: luana.silva@tecnico.ulisboa.pt
Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Evolutionary relationships between species are usually inferred through phylogenetic analysis, which provides phylogenetic trees computed from allelic profiles built by sequencing specific regions of the sequences and abstracting them to categorical indexes. The process of phylogenetic analysis consists of parsing, assembling, and profiling the sequences, so that they can then be processed by a distance calculation metric and an optional distance correction metric, followed by an inference algorithm and multiple optional local optimizations [1]. This process is exposed in projects like INNUENDO [2], which performs these operations in High Performance Computing (HPC) [3] pipelines. However, most of them, like INNUENDO, do not compute the steps after the parsing, assembling and profiling of sequences in the pipelines, because there is not yet a library that can be integrated to compute those parts.

With growing exchanges of people and merchandise, epidemics have become increasingly important, and combining information of country-specific datasets can now reveal unknown spreading patterns that were not possible to detect before, but phylogenetic algorithms are often hard to use and integrate in analysis frameworks and tools. There are hundreds of computational phylogenetics tools out there that are commonly used to address this problem. And, although they all try to achieve the same goal, which is to build a phylogenetic tree, they all differ widely in the way they operate, the formats they support, and the criteria and algorithms they implement, as shown in Supplementary Table S1. Due to those differences, the use of different tools may result in different phylogenetic trees [4] from the same algorithm. There is not yet a library that tries to integrate all of the algorithms into just one library that works on all platforms and can be integrated with other tools.

The PhyloLib project was created to address these problems by implementing the last four steps of the phylogenetic analysis workflow, namely the distance calculation, distance correction, inference algorithm, and local optimization steps, in one single tool. The distance calculation step consists of producing a distance matrix from a dataset, including several sequences, through a distance calculation method, such as Hamming [5], GrapeTree [6], or Kimura, that calculates the distances between each pair of sequences. The dataset can be represented in several formats, including MLST, MLVA, FASTA, and SNP. The distance correction step is optional and it takes a distance matrix and corrects each distance using a correction formula like the Jukes-Cantor model [7]. The inference algorithm step infers a phylogenetic tree from a distance matrix by running a clustering algorithm, like goeBURST [8], GrapeTree [6], UPGMA, or NJ by Studier and Keppler [9]. The phylogenetic tree can be

represented in several formats, including Newick [10] and Nexus [11]. The local optimization step is optional and tries to locally optimize a phylogenetic tree through an algorithm, such as LBR [6].

2 Approach

This project boils down to the development of a command line application in Java, that conforms to the phylogenetic analysis workflow and is highly performant, extensible, reusable, and portable. It is different from other existing tools (Supplementary Table S1) in the sense that it was built to be continuously extended and not just serve a single purpose. It enables reading datasets, distance matrices, and phylogenetic trees from files, calculating and correcting a distance matrix, inferring and locally optimizing a phylogenetic tree, and writing distance matrices and phylogenetic trees to files. The formats and algorithms available in it are the ones previously mentioned plus some other similar inference algorithms. Additionally, it provides the capabilities of executing only certain steps of the workflow as well as outputting the results of each step, which can be used as a way to stop and resume the workflow whenever the user desires. This is another thing that other tools do not offer, yet is particularly useful in certain scenarios, such as when the user intends to run several inference algorithms over the same input data, but does not wish to waste time or resources computing the same distance matrix for all of them. Being a command line application, each call to the library receives its commands through the CLI arguments. All commands and options are optional and case insensitive, and the order in which they are declared is irrelevant. Each command is represented by its name, type, and options, separated by a space, and is separated from other commands by a colon. Each option is represented by its name preceded by two dashes, or by a letter preceded by a dash, and followed by an equals sign and its corresponding value. The available commands are respectively `distance`, `correction`, `algorithm`, and `optimization`, each respectively defining a step in the workflow. The type and options declared in a command specify its execution, namely the type identifies the implementation for that command, and the options specify details for that implementation. For example, if the command is `algorithm` then the type may be `goeburst` and an option may be `--lvs`. Besides the custom options that may be used by each command and specific type, such as `--lvs` or `-l`, an option that may be declared for every command is `--out` or `-o`, and it defines the output file for the command. Other options that a command may or may not use, depending on its input needs, are `--dataset`, `--matrix`, and `--tree`, or `-d`, `-m`, and `-t`, each respectively defining a data type file input in the workflow. All of these file options, including `--out`, are represented by a format name followed by a colon and a file location. An example use case is to compute the goeBURST algorithm with three locus variants and the output in Newick to a file `tree.txt`, using the Hamming distance, with the dataset as input in SNP format from a file `dataset.txt`, and the LBR optimization, with the output in Newick to a file `out.txt`, as follows:

```
phylolib algorithm goeburst -l=3 -o=newick:tree.txt
      : distance hamming -d=snp:dataset.txt
      : optimization lbr -o=newick:out.txt
```

The architecture of this project takes into account reusability, thus the commands and data types concentrate as much reusable code as possible in common hierarchical classes. As a result, it also improves its extensibility as it becomes easier to extend commands and data types, since most of the code necessary to implement a command or data type is already written. Nonetheless, performance is also taken into account in the implementation, as some code that could be reused is not, due to optimizations that can be made specifically to some algorithm implementations.

3 Evaluation

The results of the time and memory benchmarks developed for the experimental evaluation are represented as a function of the number of profiles n and can be seen in Supplemental Figures S1 through S8. The time performance benchmarks show that the implementations of the algorithms conform to their theoretical time complexity, namely $\mathcal{O}(n^3)$ for NJ algorithms and $\mathcal{O}(n^2)$ for MST and GCP algorithms. However, the implementation of the GrapeTree algorithm was shown to have a considerable overhead compared to other MST and GCP algorithms, despite having the same time complexity. Meanwhile, the memory performance benchmarks lead to the conclusion that the implementations of the MST and GCP algorithms have a memory complexity of $\mathcal{O}(n^2)$, while the implementations of the NJ algorithms tend more towards a memory complexity of $\mathcal{O}(n)$. However, despite showing that the implementations of the NJ algorithms have a lower memory complexity, it is also shown that they seem to have an overhead great enough to still require more memory than the implementations of the MST and GCP algorithms with smaller datasets.

4 Conclusion

The aim of this project has been the development of a library that is efficient, reusable, extensible and portable. However, it can still be further extended to include more distance and correction metrics, inference and local optimization algorithms, and dataset, distance matrix and phylogenetic tree formats. Furthermore, it can still be extended in other ways, namely by including other optional steps in the phylogenetic analysis workflow, such as the dynamic addition of relationships between the inference algorithm and local optimization steps, and the calculation of visualization coordinates after all other steps. Also, despite it already being efficient, its time and memory performances can still be improved upon by, for example, introducing parallelization in the algorithms and a cache system in the distance matrix.

Funding

This project was partly supported by national funds through FCT – Fundação para a Ciência e Tecnologia, under projects PTDC/CCI-BIO/29676/2017 and UIDB/50021/2020.

References

[1]Cátia Vaz, Marta Nascimento, João A Carriço, Tatiana Rocher, and Alexandre P Francisco. Distance-based phylogenetic inference from typing data: a unifying view. *Briefings in Bioinformatics*, 07 2020. bbaa147.

[2]Ann-Katrin Llarena, Bruno Filipe Ribeiro-Gonçalves, Diogo Nuno Silva, Jani Halkilahti, Miguel Paulo Machado, Mickael Santos Da Silva, Anniina Jaakkonen, Joana Isidro, Crista Hämäläinen, Jasmin Joenperä, et al. Innuendo: A cross-sectoral platform for the integration of genomics in the surveillance of food-borne pathogens. *EFSA Supporting Publications*, 15(11):1498E, 2018.

[3]Kevin Dowd and Charles Severance. High performance computing. 2010.

[4]Andreia Sofia Teixeira, Pedro T Monteiro, João A Carriço, Mário Ramirez, and Alexandre P Francisco. Not seeing the forest for the trees: size of the minimum spanning trees (msts) forest and branch significance in mst-based phylogenetic analysis. *Plos one*, 10(3):e0119315, 2015.

[5]Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[6]Zhemin Zhou, Nabil-Fareed Alikhan, Martin J Sergeant, Nina Luhmann, Cátia Vaz, Alexandre P Francisco, João André Carriço, and Mark Achtman. Grapetree: visualization of core genomic relationships among 100,000 bacterial pathogens. *Genome research*, 28(9):1395–1404, 2018.

[7]Keith Erickson. The jukes-cantor model of molecular evolution. *Primus*, 20(5):438–445, 2010.

[8]Alexandre P Francisco, Miguel Bugalho, Mário Ramirez, and João A Carriço. Global optimal eburst analysis of multilocus typing data using a graphic matroid approach. *BMC bioinformatics*, 10(1):152, 2009.

[9]James A Studier and Karl J Keppler. A note on the neighbor-joining algorithm of saitou and nei. *Molecular biology and evolution*, 5(6):729–731, 1988.

[10]Joseph Felsenstein, J Archie, W Day, W Maddison, C Meacham, F Rohlf, and D Swofford. The newick tree format, 1986.

[11]David R Maddison, David L Swofford, and Wayne P Maddison. Nexus: an extensible file format for systematic information. *Systematic biology*, 46(4):590–621, 1997.

Library of efficient algorithms for phylogenetic analysis: Supplementary Data

RELATED WORK

There are hundreds of computational phylogenetics tools out there that are commonly used in comparative genomics, cladistics, and bioinformatics. Although they all try to achieve the same goal, which is to build a phylogenetic tree, they all differ widely in multiple aspects.

These tools may represent phylogenetic trees in different formats, such as Newick, Nexus, or even a format of their own. They may even deal with different input formats, such as FASTA, SNP, MLST, MLVA, among others. They may also have different implementations, producing different phylogenetic trees by relying on a set of algorithms for estimating phylogenies, such as Neighbour Joining, Maximum Parsimony, Globally Closest Pairs, Bayesian phylogenetic inference, and Maximum Likelihood. They can also rely on several different distance calculation and correction formulas, such as Hamming, Jukes-Cantor, Kimura, and so on. Some tools may provide local optimization algorithms, such as LBR, SPR, NNI, and TBR, while others may not provide any.

These tools can have two different purposes. Some are meant to be used by other tools, in the format of a library or command-line application. While others are created to be used directly by the final user, as a desktop or web application, which may be free or paid. They can be implemented in different languages and used in different platforms or even be specific to one platform. However, besides differing in many aspects, these tools may also share some unappealing aspects, like not being easily integrated into existing phylogenetic analysis workflows, not supporting a common API between algorithms, and not always providing efficient implementations.

Some examples of the most well-known libraries are PHYLIP [1], PhyML [2], RAxML [3], PAUP* [4], MrBayes [5] and MEGA [6]. And, some examples of the most frequently used desktop and web applications, that are free, are PHYLOViZ and PHYLOViZ Online. The major differences between these tools, regarding the phylogenetic analysis workflow for distance matrix based algorithms, can be seen in Table S1. In this table, the first and last columns represent the input and output formats supported by each tool, the columns with a number between parentheses represent the four steps of the phylogenetic analysis workflow discussed in this chapter, and the second to last column refers the other four columns and represents the steps of the workflow that provide an output file.

EXPERIMENTAL EVALUATION

All components of this library were tested in terms of their functionality, through unit testing implemented using the TestNG [7] framework, except for the local optimization which does not have another implementation to compare its results to. However, only the algorithm component was tested in terms of time and memory performances, since it is the core operation of the workflow and the one that requires the most time and memory to execute. The experimental evaluation will focus itself on the time and memory performances, comparing both the performance of each algorithm against each other, as well as the performance of each algorithm with an implicit matrix versus with an explicit matrix.

The time and memory performances were tested through the implementation and execution of benchmarks with 10 warmups and 20 iterations, over the first ten to one thousand profiles of the Streptococcus pneumoniae dataset [8], using the Hamming distance as the distance calculation method. The results are represented as a function of the number of profiles n. The same dataset and distance calculation method were used throughout the benchmarks to provide an equal and fair evaluation to all algorithms. For that same reason, all tests were performed in the same machine, in this case with a 2.6 GHz 6-Core Intel Core i7 processor and a 16 GB 2667 MHz DDR4 memory.

Time

The average running time that each algorithm took to execute in the eager version, over the increasing number of profiles of the given dataset, is represented in a plot graph in Figure S1, in milliseconds. From this graph it is possible to see the difference in time complexity that exists between the NJ algorithms and all the others.

Despite having the same time complexity as other MST and GCP algorithms, it is possible to see from the previous plot graph that the GrapeTree algorithm is much slower than the others.

Tool \ Feature	Input Format	Distance Calculation (1)	Distance Correction (2)	Inference Algorithm (3)	Local Optimization (4)	Output Processing	Output Format
PHYLIP	PHYLIP	Hamming	Fitch-Margoliash	UPGMA NJ by Saitou & Nei	Robinson-Foulds	(1) (2) (3) (4)	Newick
PhyML	PHYLIP Nexus	Hamming	JC69 K80 F81 F84 HKY85 TN93 GTR	BioNJ	NNI SPR	(3) (4)	Newick
RAxML	PHYLIP FASTA	Hamming	JC69 K80	-	NNI	(3) (4)	Newick
PAUP*	Nexus	Hamming	HKY85	UPGMA NJ by Saitou & Nei	-	(3)	Nexus
MrBayes	Nexus	Hamming	JC69 K80 F81 HKY85 GTR	BioNJ NJ by Saitou & Nei	-	(3)	Nexus
MEGA	MEGA	Hamming	Jukes-Cantor Tajima-Nei Kimura 2-Parameter Tamura 3-Parameter Tamura-Nei Log-Det	UPGMA NJ by Saitou & Nei	-	(3)	Newick
PHYLOViZ	MLST MLVA SNP	Hamming	-	goeBURST CL SL UPGMA WPGMA NJ by Saitou & Nei NJ by Studier & Keppler	-	(3)	Custom JSON
PHYLOViZ Online	MLST MLVA FASTA Newick	Hamming	-	goeBURST	-	(3)	Custom JSON
PhyloLib	MLST MLVA FASTA SNP Symmetric Matrix Asymmetric Matrix Newick Nexus	Hamming GrapeTree Kimura	Jukes-Cantor	goeBURST GrapeTree CL SL UPGMA WPGMA UPGMC WPGMC NJ by Saitou & Nei NJ by Studier & Keppler UNJ	LBR	(1) (2) (3) (4)	Symmetric Matrix Asymmetric Matrix Newick Nexus

Table S1. Major differences between some of the most well-know phylogenetic tools, regarding the phylogenetic analysis workflow.

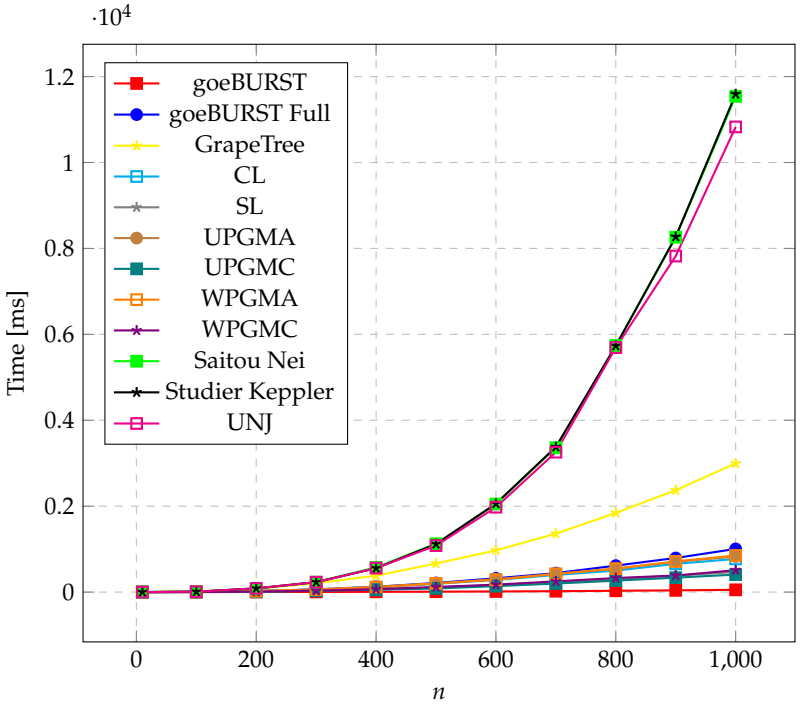

Fig. S1. Running times in milliseconds for ten to one thousand profiles using the eager version.

The difference in the results of the eager and lazy versions can be better compared in individual plot graphs. Figures S2, S3 and S4 represent these differences for MST, GCP and NJ algorithms respectively.

From these individual plot graphs it is possible to see that the implementations of the algorithms conform to their theoretical time complexities, namely $\mathcal{O}(n^3)$ for NJ algorithms and $\mathcal{O}(n^2)$ for MST and GCP algorithms. It is also possible to see a difference in running time between the eager and lazy versions of the algorithms. However, these differences are hardly noticeable due to the small impact that the distance calculation has on the workflow, compared to the inference algorithm, except when compared to goeBURST, as its running times are small enough to notice a difference. Thus, the benefit of storing the distance matrix in a file and reusing it is almost insignificant in terms of running time for most algorithms.

Memory

The average memory usage that each algorithm took to execute in the eager version, over the increasing number of profiles of the given dataset, is represented in a plot graph in Figure S5, in megabytes. From this graph it is possible to see that the goeBURST algorithms require the lesser memory between all algorithms, while the GrapeTree algorithm requires the most.

The difference in the results of the eager and lazy versions can be better compared in individual plot graphs. Figures S6, S7 and S8 represent these differences for MST, GCP and NJ algorithms respectively.

From these individual plot graphs it is possible to see that the implementations of the MST and GCP algorithms have a memory complexity of $\mathcal{O}(n^2)$, while the implementations of the NJ algorithms tend more towards a memory complexity of $\mathcal{O}(n)$. However, despite that, the implementations of the NJ algorithms seem to have an overhead great enough to still require more memory than the implementations of the MST and GCP algorithms with smaller datasets. It is also possible to see that the memory results do not follow a clear pattern and the difference between the lazy and eager versions is almost inexistent. Thus, reusing a distance matrix stored in a file is almost insignificant in terms of memory for all implemented algorithms.

REFERENCES

1. J. D. Retief, "Phylogenetic analysis using phylip," in *Bioinformatics methods and protocols*, (Springer, 2000), pp. 243–258.

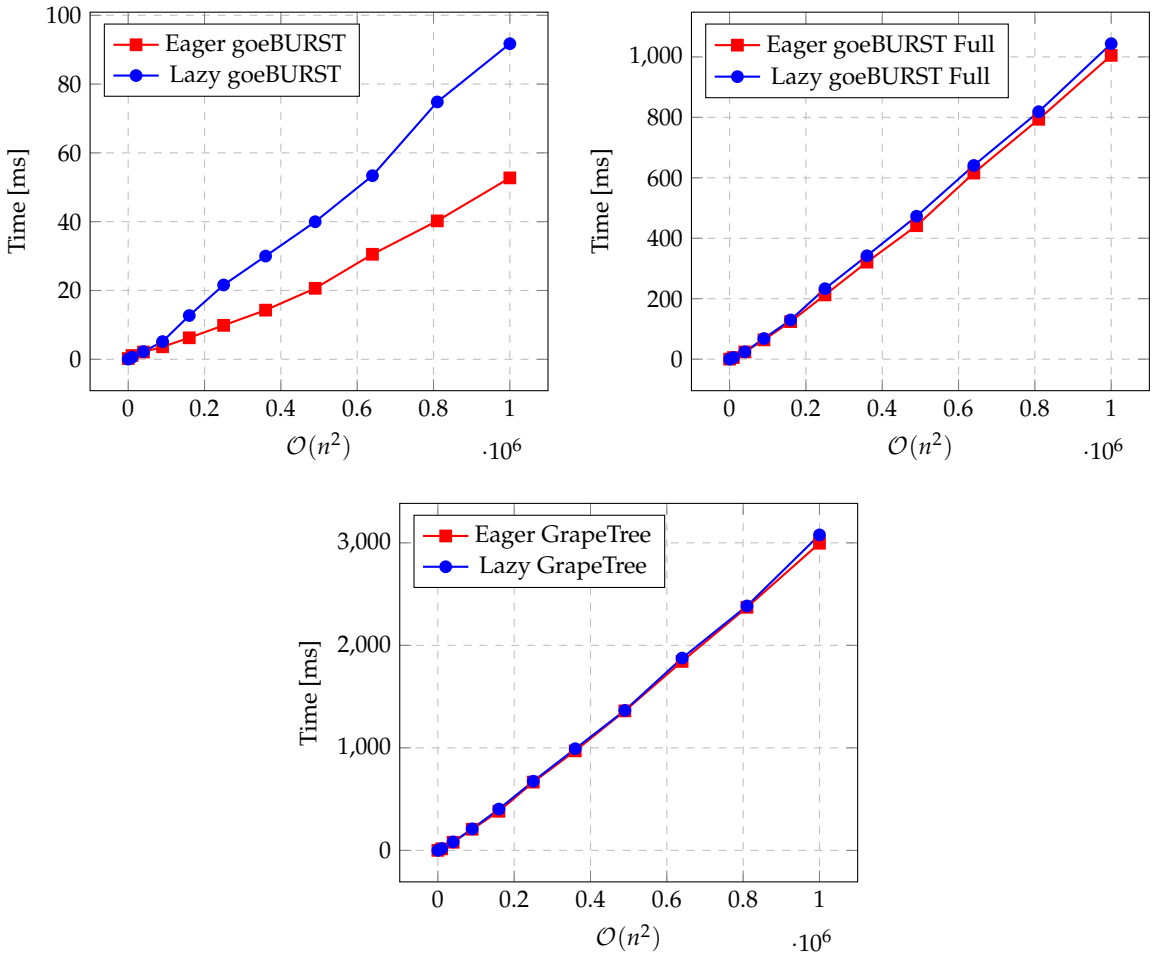

Fig. S2. Running times in milliseconds for MST algorithms compared to their time complexity.

2. S. Guindon, J.-F. Dufayard, V. Lefort, M. Anisimova, W. Hordijk, and O. Gascuel, "New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0," Syst. biology **59**, 307–321 (2010).
3. A. Stamatakis, "Raxml version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies," Bioinformatics **30**, 1312–1313 (2014).
4. D. L. Swofford, "Paup*: Phylogenetic analysis using parsimony (and other methods) 4.0. b5," (2001).
5. J. P. Huelsenbeck and F. Ronquist, "Mrbayes: Bayesian inference of phylogenetic trees," Bioinformatics **17**, 754–755 (2001).
6. S. Kumar, M. Nei, J. Dudley, and K. Tamura, "Mega: a biologist-centric software for evolutionary analysis of dna and protein sequences," Briefings bioinformatics **9**, 299–306 (2008).
7. C. Beust and H. Suleiman, *Next generation Java testing: TestNG and advanced concepts* (Pearson Education, 2007).
8. K. Jolley, "Pubmlst," https://pubmlst.org/. [Online; accessed 15 December 2019].

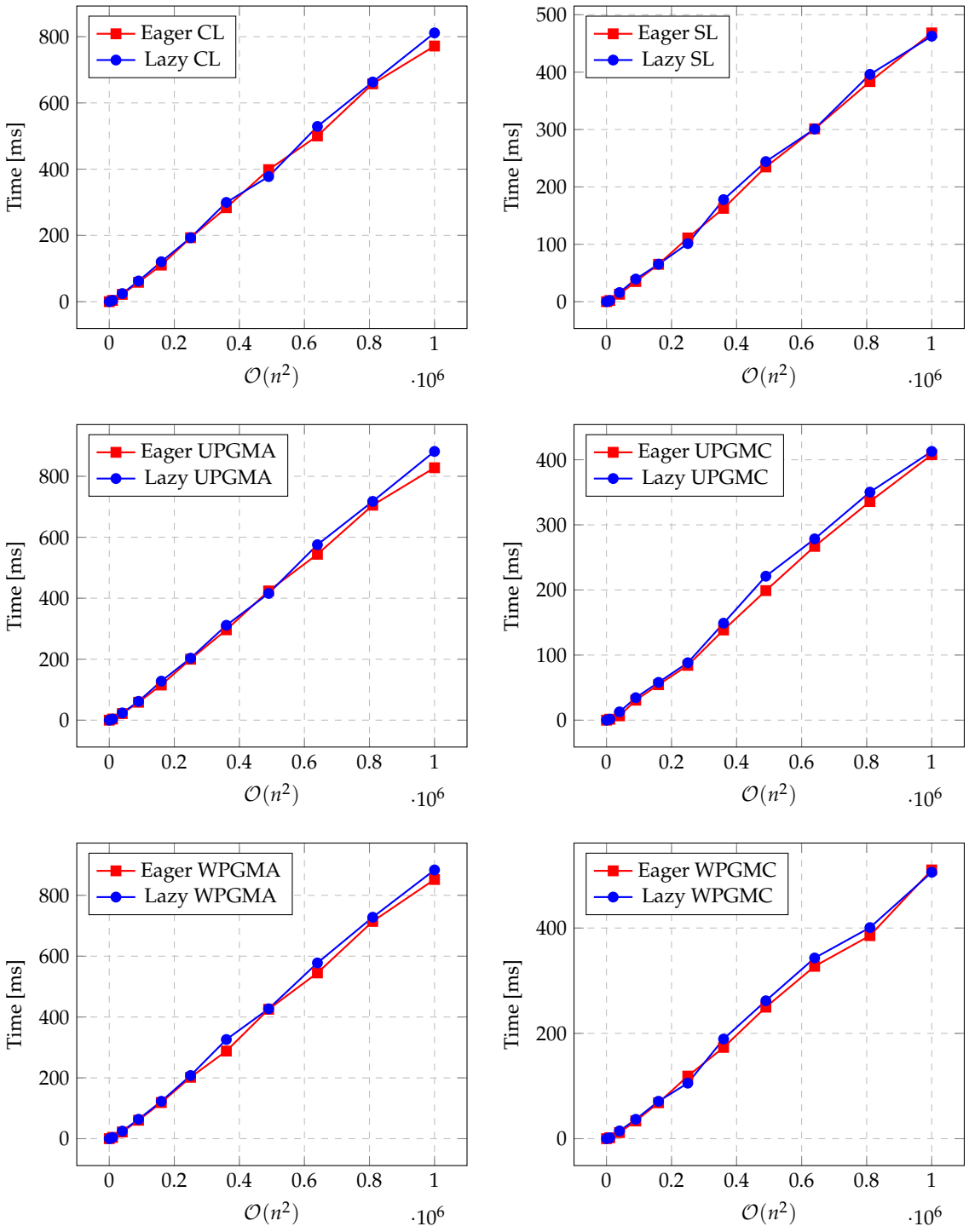

Fig. S3. Running times in milliseconds for GCP algorithms compared to their time complexity.

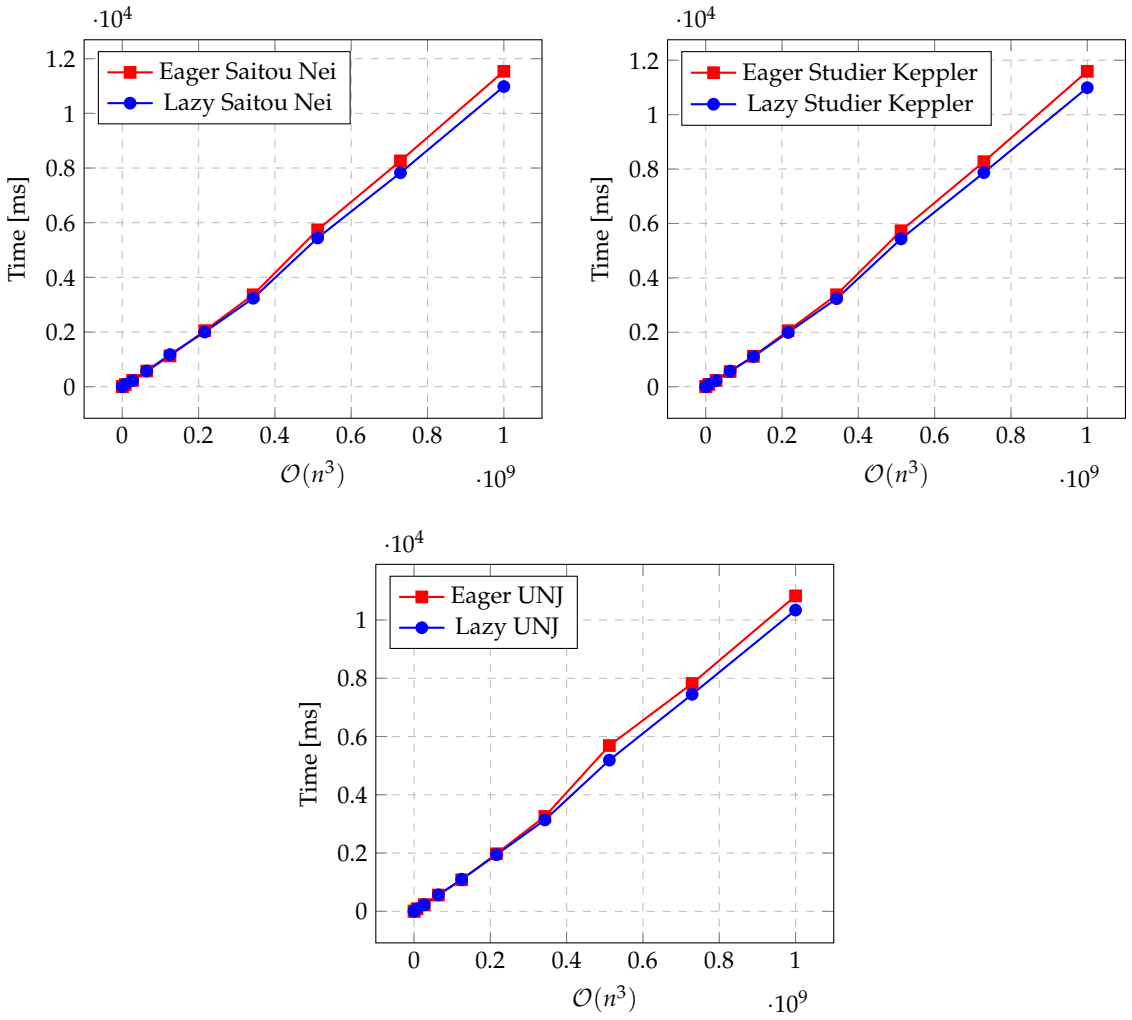

Fig. S4. Running times in milliseconds for NJ algorithms compared to their time complexity.

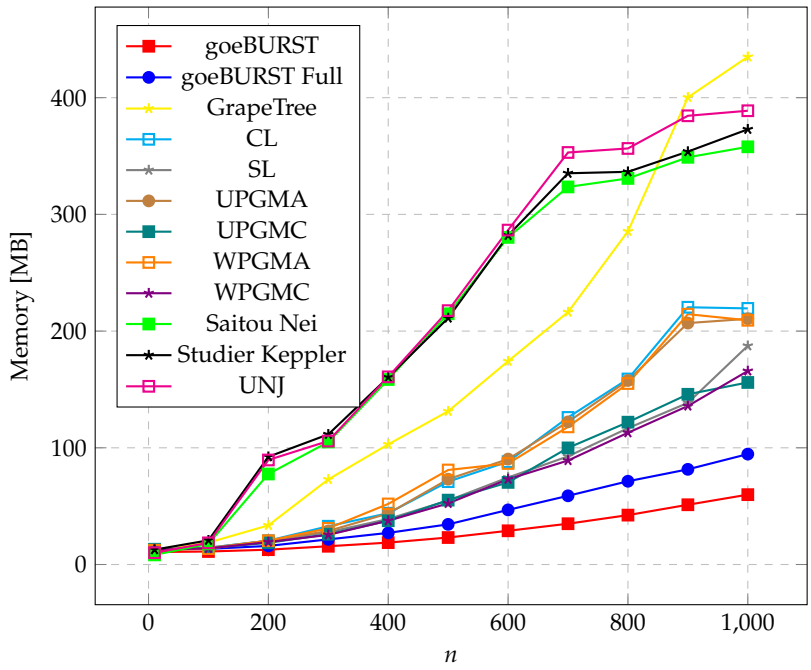

Fig. S5. Peak memory usage in megabytes for ten to one thousand profiles using the eager version.

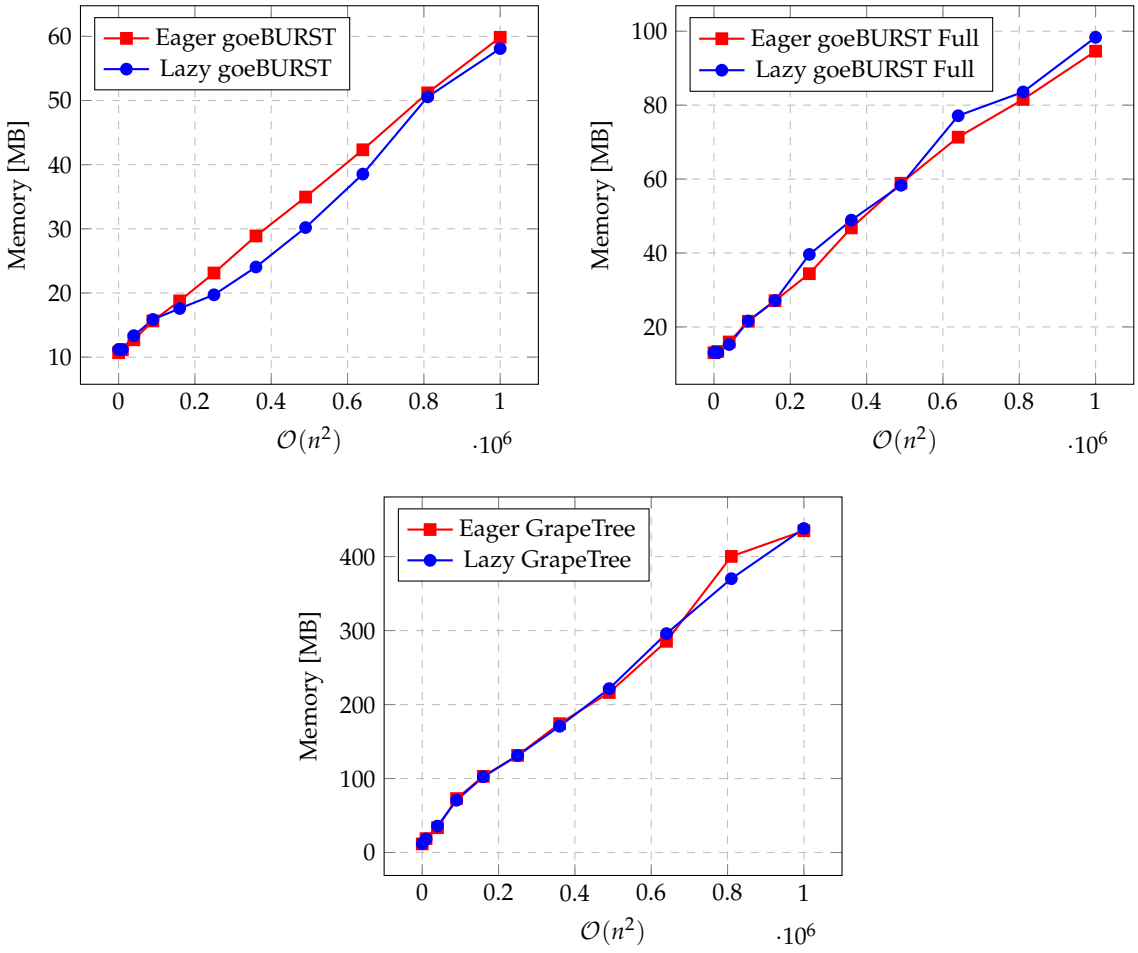

Fig. S6. Peak memory usage in megabytes for MST algorithms compared to their memory complexity.

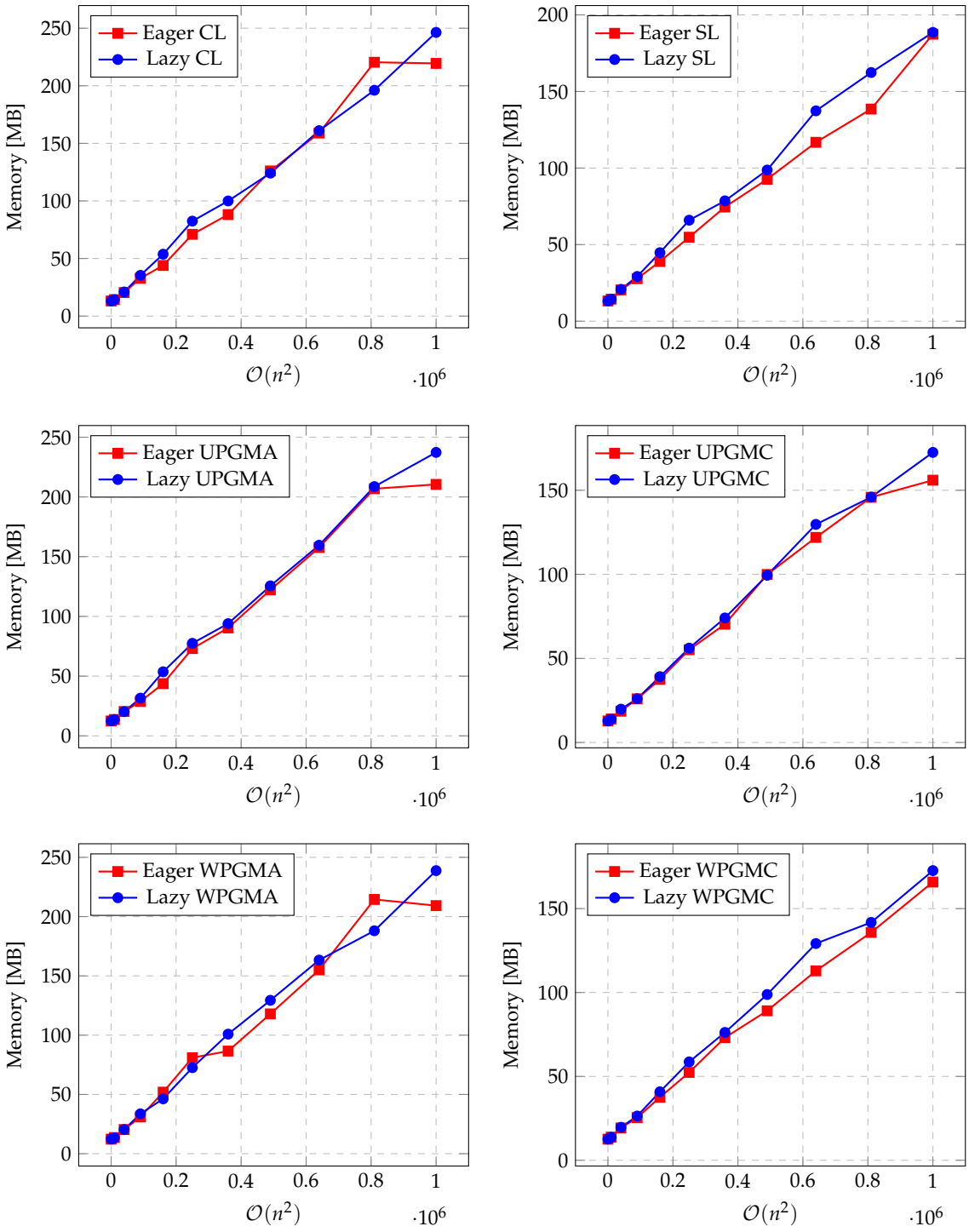

Fig. S7. Peak memory usage in megabytes for GCP algorithms compared to their memory complexity.

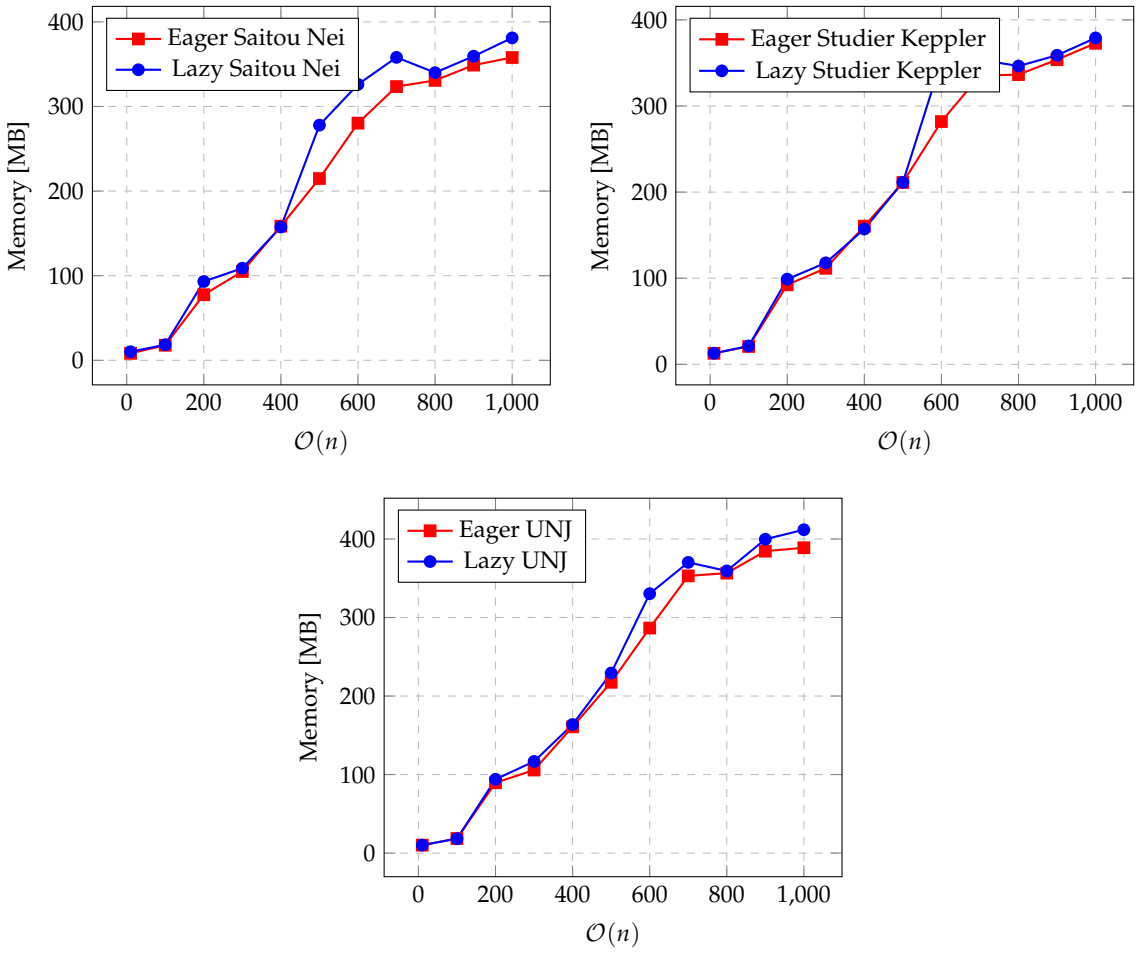

Fig. S8. Peak memory usage in megabytes for NJ algorithms compared to their memory complexity.