Software for Systematics and Evolution

Syst. Biol. 72(1):242–248, 2023

https://doi.org/10.1093/sysbio/syad002
Advance Access Publication January 27, 2023

Lagrange-NG: The next generation of Lagrange

Ben Bettisworth[1,*], Stephen A. Smith[2], and Alexandros Stamatakis[1,3]

[1]*Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies, Heidelberg, Germany*
[2]*Ecology and Evolutionary Biology, University of Michigan, Ann Arbor, United States*
[3]*Institute for Theoretical Informatics, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[*]*Correspondence to be sent to: Computational Molecular Evolution Group, Heidelberg Institute for Theoretical Studies, Heidelberg, Germany; E-mail: ben.bettisworth@h-its.org*

Received 21 April 2022; reviews returned 9 January 2023; accepted 20 January 2023
Associate Editor: James Albert

Abstract.— Computing ancestral ranges via the Dispersion Extinction and Cladogensis (DEC) model of biogeography is characterized by an exponential number of states relative to the number of regions considered. This is because the DEC model requires computing a large matrix exponential, which typically accounts for up to 80% of overall runtime. Therefore, the kinds of biogeographical analyses that can be conducted under the DEC model are limited by the number of regions under consideration. In this work, we present a completely redesigned efficient version of the popular tool Lagrange which is up to 49 times faster with multithreading enabled, and is also 26 times faster when using only one thread. We call this new version Lagrange-NG (Lagrange-Next Generation). The increased computational efficiency allows Lagrange-NG to analyze datasets with a large number of regions in a reasonable amount of time, up to 12 regions in approximately 18 min. We achieve these speedups using a relatively new method of computing the matrix exponential based on Krylov subspaces. In order to validate the correctness of Lagrange-NG, we also introduce a novel metric on range distributions for trees so that researchers can assess the difference between any two range inferences. Finally, Lagrange-NG exhibits substantially higher adherence to coding quality standards. It improves a respective software quality indicator as implemented in the SoftWipe tool from average (5.5; Lagrange) to high (7.8; Lagrange-NG). Lagrange-NG is freely available under GPL2. [Biogeography; Phylogenetics; DEC Model.]

Lagrange-NG implements the DEC (Dispersion Extinction and Cladogensis) model of geographic range evolution Ree et al. (2005). A geographic range, in this context, describes the broadly defined distribution of the habitat of a particular species. The evolution of this range is assumed to follow the phylogeny, or the biological evolution of a species or clade. The DEC model takes, as a minimum, a phylogenetic tree, and a set of regions. The phylogenetic tree is assumed to be the true phylogeny of the included species, and the regions are the generalized areas of potential habitation for the species in question. The DEC model constructs a list of states based on the valid set of regions that a particular species could inhabit. With these components, the DEC model constructs a transition matrix between states using two parameters, an extinction parameter and a dispersion parameter. Using this transition matrix, the likelihood of the model parameters can be computed and used to optimize the model parameters. Once the optimal model parameters have been found, the most likely ancestral ranges can be found by computing the model "backwards."

However, computing likelihoods under the DEC model is computationally challenging. This is because: (i) the geographical regions are splayed into $2^r = s$ states (Landis et al., 2013) as each possible range distribution is represented as a state; and (ii) the computation of the respective transition matrix is in $\mathcal{O}\left(s^3\right)$. Thus, computing a single likelihood of the DEC model requires $\mathcal{O}\left(\left(2^r\right)^3\right) = \mathcal{O}\left(2^{3r}\right)$ time. In other words, the likelihood computation is exponential with respect to the number of regions under study. Therefore, the scalability of data analyses under the DEC model is limited to the number of regions, that is, only a small number of 6 to 10 regions can be analyzed in a reasonable time (Landis et al., 2013; Matzke, 2013).

The most expensive inference step is the computation of the transition matrix that often accounts for 80% or more of overall runtime. As in standard likelihood-based phylogenetics, the transition matrix is computed via a matrix exponential, albeit on a substantially larger matrix. Substantial research effort has been invested into finding the best way to compute the matrix exponential (Moler and Loan, 2003), but it still remains challenging to compute efficiently as well as accurately. In addition, unlike in standard phylogenetics, the DEC model is nonreversible (i.e., uses a nonsymmetric rate matrix), which limits the number of applicable numerical methods for computing the matrix exponential, typically to less precise ones. In the following, we present the Lagrange-NG (Lagrange-Next Generation) software, an almost complete rewrite of the popular and widely used Lagrange software by Ree et al. (2005), and more specifically the unpublished but available to use C ++ version of Lagrange developed by Smith which can be found on GitHub at https://github.com/rhr/lagrange-cpp. In this work, when we refer to the original Lagrange, we are referring not to the Python version from Ree and Smith (2008), but to the unpublished C ++ version. As the primary challenge to computing the likelihood under the DEC model is to efficiently calculate the matrix exponential, Lagrange-NG relies on a relatively recent method of computing a matrix

exponential based on Krylov Subspaces (Moler and Loan, 2003) for a moderate to large number of regions (six and more) in the default mode of operation.

Alongside the improvements to the matrix exponential, many so-called "micro-optimizations" (e.g., passing function arguments by reference instead of value, using more efficient data structures to store regions, or eliminating unnecessary computation) have been implemented that further accelerate computations. We have also implemented a task-based hybrid multithreading approach, which increases the rate of analyses by up to a factor of eight for datasets with exceeding 200 taxa. Furthermore, we improve upon the numerical stability compared to the original software, and fix a major bug which we discovered during development. Finally, to verify that Lagrange-NG produces analogous results as the original implementation, we devised a novel method of comparing range distribution on trees, which is based on the Earth mover's distance metric. A similar application of the Earth mover's distance has been successfully applied to phylogenetic placement, though this method and application is distinct (Evans and Matsen, 2012).

Software Description

Lagrange-NG constitutes an nearly complete rewrite of the original (unpublished) C ++ version of Lagrange. Of the 4600 lines of code present, only 5% are present from the original code base. This redesign retains the complete functionality of Lagrange, but is computationally more efficient, and implements a parallelization of DEC calculations. Lagrange-NG implements four major improvements to Lagrange. First, it supports parallelism via a hybrid task-based parallelization scheme which utilizes both coarse and fine-grained parallelism. The coarse-grained parallelism of Lagrange-NG assigns tasks to workers, which for this work can be thought of as threads for the purposes of this paper. For more details, please see the Supplementary Materials. Second, it deploys more efficient numerical methods and algorithms which were developed relatively recently to compute the matrix exponential, for example, an algorithm based on Krylov subspaces which we use in Lagrange-NG. Third, it introduces general improvements and optimizations, that is, micro-optimizations, which individually do not notably increase efficiency, but put together yield a substantial improvement. Finally, the fourth improvement is a substantial increase in coding standards adherence and hence, software quality, as measured by the coding standards adherence evaluation tool and benchmark SoftWipe (Zapletal et al., 2021). The SoftWipe score of the original Lagrange software is 5.5, while our nearly complete rewrite increases this to a score of 7.8. While the original score of 5.5 is fairly average, the new score of 7.8 places Lagrange-NG 3rd in the list of 51 scientific software tools written in C or C ++ that are contained in the SoftWipe benchmark.

Importantly, during the process of improving the code quality, a potentially serious bug was discovered which, as far as the authors can determine, affects not only the C ++ version of Lagrange but also applies to the older Python version of Lagrange which uses matrix exponentiation to compute the transition matrix. In order to correct numerical instabilities, the transition matrix was normalized such that the rows summed to 1.0 after the matrix exponential computation. During normal computation, this operation will have little effect on the results. However, if the rate matrix is sufficiently ill conditioned, the computation exhibits an extreme numerical instability such that any results produced are meaningless. If the matrix is then normalized at this point, then results produced with this matrix are made to appear sensible. Therefore, any error in the computational process is hidden from the user, and the results of the computation will be perceived as plausible. Fortunately, as long as the matrix remains unnormalized, these errors are easy to detect, as several analytical conditions are no longer met (such as the rows no longer summing to 1.0). We are not aware of any approaches to recover from these errors, but at least the user is not misled into thinking that meaningless results are plausible. This normalization error is exceedingly rare, as the authors never observed it in the thousands of datasets analyzed for this paper. Despite this, the error *can* occur, and *will* by Murphy's law. As such, we are convinced that in the event of this bug, the user should be appropriately informed. Therefore, in this case, Lagrange-NG simply fails, and alerts the user to what occurred.

In addition, we identified and corrected a configuration error in the process of building Lagrange, where important compiler optimization options were not properly utilized. Fixing this configuration error alone increased the computational efficiency of the original Lagrange by up to 10×. While this error is easy to overlook, yet trivial to fix, we assume that many past Lagrange analyses were conducted using the unoptimized code. Nonetheless, in this work when we perform benchmarks with Lagrange, we do them with this configuration error fixed.

Lagrange-NG can be downloaded from GitHub at https://github.com/computations/lagrange-ng along with a tool to plot the results at https://github.com/computations/lagrange-ng-plotter. To build the software, the only requirements are a C ++ compiler, and CMake. Optionally, Lagrange-NG can be built with the respective system versions of the Intel Math Kernel Library (2022) (also known as MKL) and NLOpt (Nelder and Mead, 1965; Johnson, 2021). If a system version of MKL is not present, Lagrange-NG will build with OpenBLAS (2022) instead.

Please see the Supplementary Material for a more thorough description of Lagrange-NG.

Performance

To assess the performance of Lagrange-NG relative to the original implementation, we randomly

generated a large number of synthetic datasets with a varying number of regions and executed Lagrange and Lagrange-NG to record the respective runtimes. We generated 100 random datasets with either 5, 6, or 7 regions, and all had 100 taxa, to obtain a total of 300 datasets. Furthermore, we ran an additional series of parallel performance evaluations on Lagrange-NG with eight threads assigned to coarse grained parallelization using the same datasets. The results of this performance assessment are shown in Fig. 1. In addition, we assessed the performance of only Lagrange-NG on datasets with 8, 9, 10, 11, or 12 regions when using eight workers. For the experiments with eight regions, we generated 100 datasets, for the experiments with 9 or 10 regions, we generated 30 datasets, and for the experiments with 11 or 12 regions, we generated 10 datasets. Results from this performance assessment are shown in Fig. 2. To assess the parallel scaling of Lagrange-NG, we generated 100 datasets with 500 taxa a six regions. Using these datasets, we measured the execution time of Lagrange-NG with 1, 4, 8, 16, and 32 cores. We took the mean runtime for each core count and used this value to compute speedups as seen in Fig. 3.

Additional performance tests are available in the Supplementary Material.

Metric Performance

To demonstrate the performance of the Wassserstein metric, we use a case study with three regions: A, B, and C. These three regions induce a state space with 8 states, with the states being "Empty," "A," "B," "C," "AB," "AC," "BC," and "ABC." For each of these states, we generated a "basis" distribution, which is a distribution vector containing 1.0 in the entry corresponding to the state, and 0.0 everywhere else. For example, the basis distribution representing "A" would be

$$(0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0)$$

We then computed the pairwise distance between all of the basis vectors. The results of this computation is summarized in Fig. 4. As it can be seen from this summary, our Wasserstein metric behaves as intended, which is to say states which have fewer regions in common with each other are farther away.

Validation

Lagrange-NG re-implements core numerical routines of Lagrange. Such changes in numerical routines are often associated with difficult and subtle bugs as well

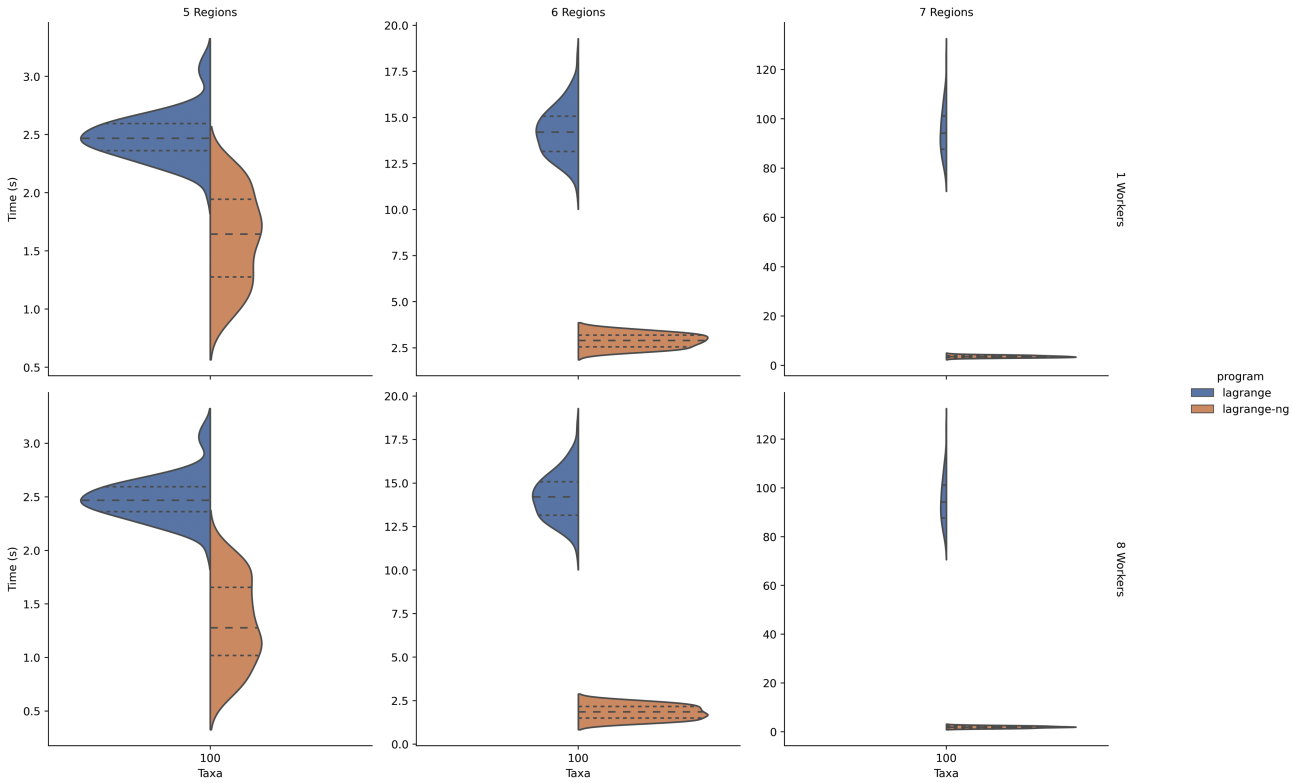

Figure 1. Comparison of runtimes between Lagrange (left) and Lagrange-NG (right) with sequential Lagrange-NG (top) and parallel Lagrange-NG using 8 cores (bot). Results were obtained by generating 100 random datasets. Note that the original Lagrange was not run with any multi-threading, as it does not support it. Instead, the data has been replicated for comparison's sake. Times are in seconds. The figure was generated using seaborn (Waskom, 2021).

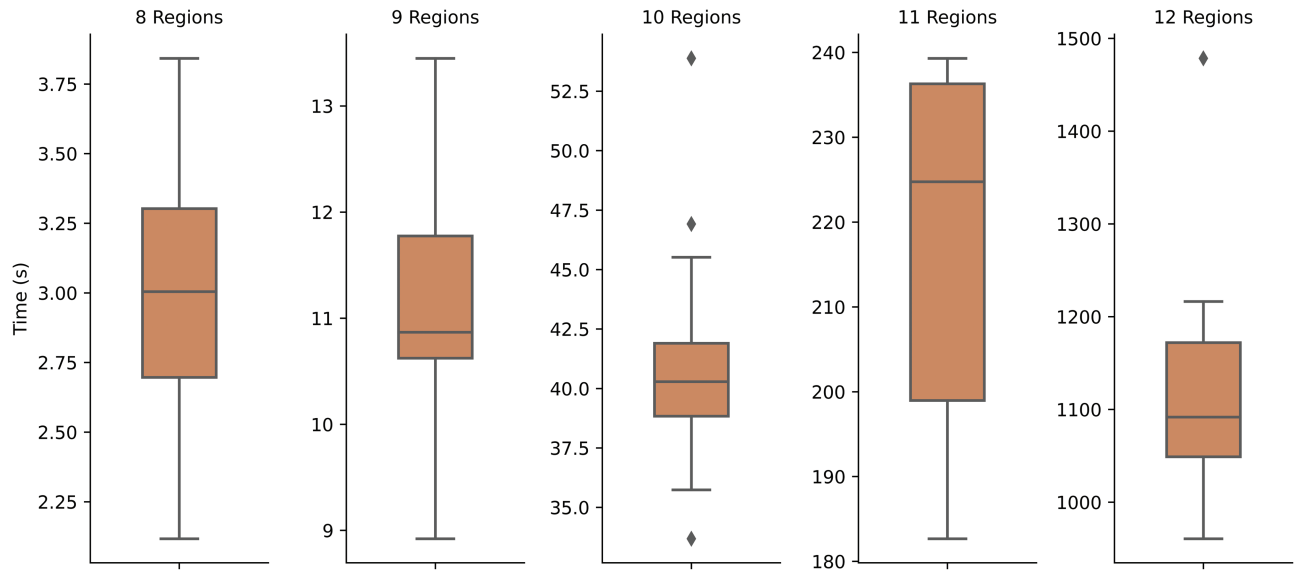

FIGURE 2. Runtimes for Lagrange-NG on a larger number of regions when using 8 cores. Results were obtained by generated random datasets with 100 taxa and 8, 9, 10, 11, or 12 regions. We generated 100 random datasets for the 8 region cases; for the 9 and 10 region cases we generated 30 datasets; and for the 11 and 12 region cases we generated 10 datasets.

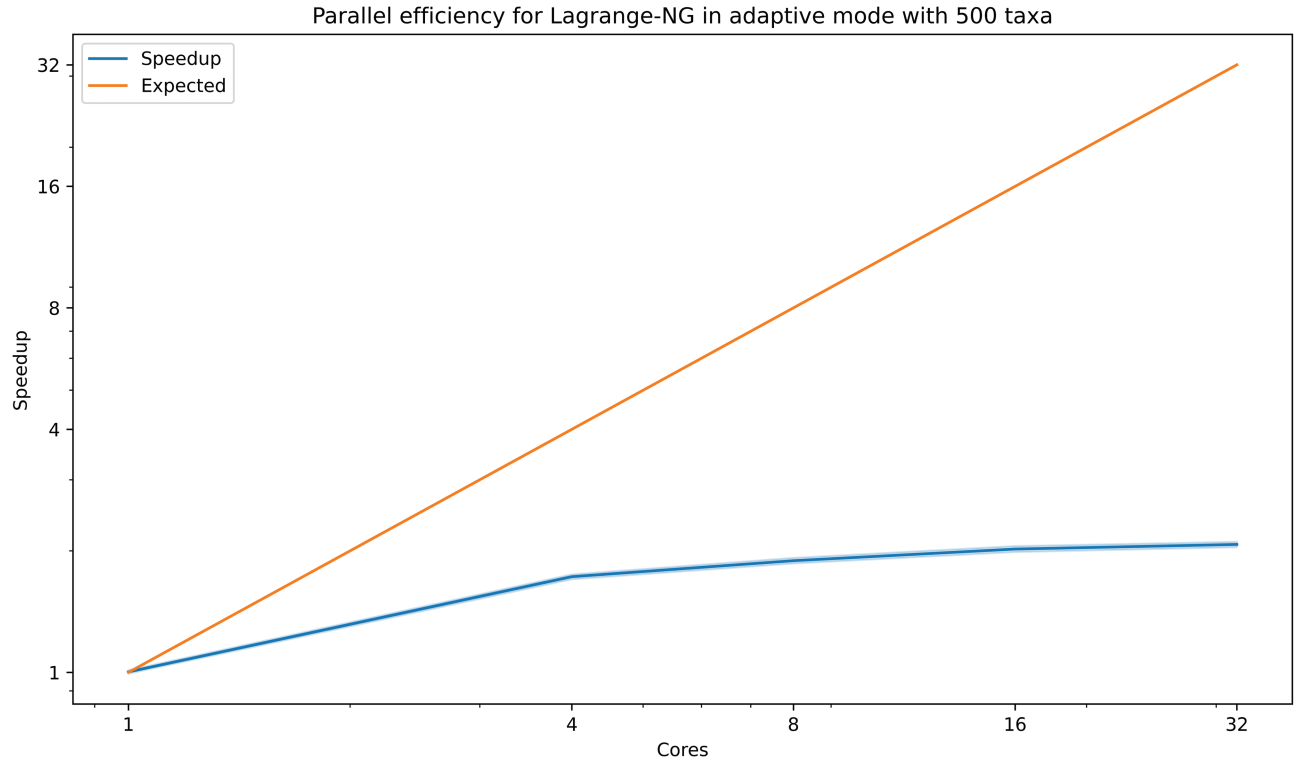

FIGURE 3. Parallel efficiency plot for a datasets with 500 taxa and 6 regions. Please notice the log-log scaling. The actual values plotted are 1.7, 1.8, 2.0, 2.1 for 4, 8, 16, and 32 cores, respectively. The ratio of the realized speedup to the optimal speedup is 0.43, 0.24, 0.13, 0.06 for 4, 8, 16, and 32 cores, respectively.

as slight numerical deviations. We sought to ensure that Lagrange-NG and Lagrange produced the same results. To this end, we developed a pipeline to (i) generate random datasets, (ii) run both, Lagrange, and Lagrange-NG, and (iii) compare the results of the two programs. To compare results, we developed a measure to evaluate the distance between distributions of ancestral ranges on trees based on the Wasserstein metric (Vaserstein, 1969). We provide a summary here; further details are provided in the Supplementary Material.

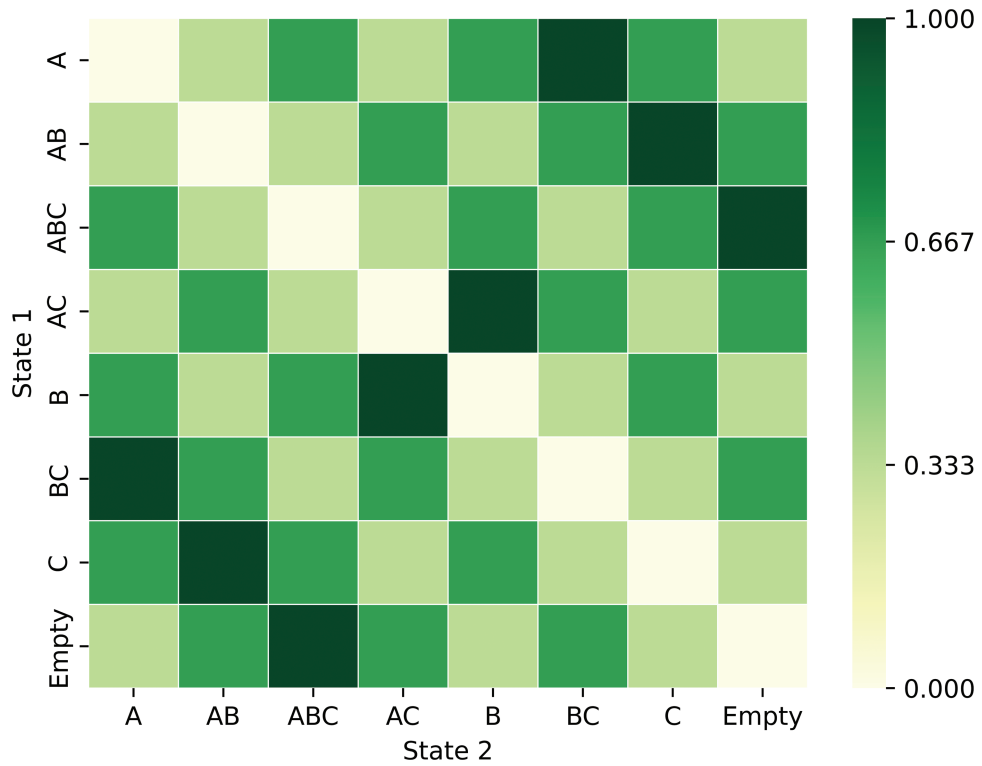

FIGURE 4. Plot showing the pairwise Wasserstein metric between the "basis" distributions on three regions: A, B, and C. Here, the "basis" distributions are a set of distributions, one for each state, with 1.0 in the corresponding entry for that state. Distributions are labeled by which entry contains the 1.0. States are ordered by a Gray code for aesthetic reasons only.

To compute the distance between two ancestral range distributions on a given tree, we embed the range as a vertex weight into a hypercube graph, and then use that graph to compute the Wasserstein metric between the distributions for each node. This metric is then normalized to be between 0.0 and 1.0 by dividing by the maximum total distance, and subsequently averaged over all nodes. The intention of this metric is to better account for what the states represent in terms of range distributions in the real world. By transforming the problem into a distance through a hypercube graph, we better match the intuition that states with less regions in common are more distant.

We ran this comparison for 100 iterations on datasets comprising 10, 50, and 100 taxa, and a number of regions between 2 and 6. This yielded 15 parameter sets, for a total of 1,500 tests.

Biological Examples

While we conducted extensive tests on simulated data, we also verify that Lagrange-NG behaves correctly on empirical datasets. To this end, we reproduced the results from a previous study on sloths from Varela et al. (2019). In addition, we took the opportunity to reproduce the results using the tools specified in the paper as this gave us an opportunity to compare with BioGeoBEARS (Matzke, 2013), a similar tool.

In order to reproduce results, we downloaded the supplementary data from the Dryad repository associated with the publication. To run the analyses with BioGeoBEARS and Lagrange-NG, we had to slightly modify the data. This involved correcting some taxon names so that they matched between the tree and the region data, and also removing the outgroup from the tree as there was no region data included for the outgroup. These modifications appear to be in line with what the original authors must have done, because the results from both BioGeoBEARS and Lagrange-NG match the results reported in the article. Both BioGeoBEARS and Lagrange-NG were run with the same dataset on the same computer. Despite the fact that the original study limited the number of regions to 5, we decided to also measure Lagrange-NG's performance with no region limit, to show that Lagrange-NG can analyze large empirical datasets without a region limit.

Validation

The validation of Lagrange-NG with respect to Lagrange, was surprisingly successful, despite substantial modifications of nearly all critical code paths and numerical routines. Of the 1,500 tests, 0 produced

results with differences over the tolerance of 1×10^{-4} when computed using our novel distance method, indicating that the results are equivalent between the two tools.

The mean sequential speedup between Lagrange-NG over Lagrange on one core for 5, 6, and 7 regions is 1.54, 4.93, and 26.63, respectively. The overall time-to-solution speedup of Lagrange-NG with 8 cores over sequential Lagrange on one core for 5, 6, and 7 regions is 1.88, 7.75, and 49.2, respectively. For datasets with larger regions, Lagrange-NG analyzed these datasets with a mean time of 3.00 s, 11.12 s, 40.75 s, 217.01 s, and 1130.60 s for 8, 9, 10, 11, and 12 regions, respectively.

The speedup from adding more cores when compared to the 1 core execution is 1.7, 1.8, 2.0, 2.1 for 4, 8, 16, and 32 cores, respectively, on simulated datasets with 500 taxa and 6 regions, as can be seen in Fig. 3. The ratio between realized speedup and idealized speedup for this experiment is 0.43, 0.24, 0.13, 0.06 for 4, 8, 16, and 32 cores, respectively. Additional results for parallel speedups can be seen in the Supplemental Material.

In addition, Lagarange-NG is substantially faster than BioGeoBEARS. On the empirical dataset, Lagrange-NG computed the result in about 7 seconds using 8 cores, while BioGeoBEARS required about 14 min to analyze the data using 80 cores. BioGeoBEARS and Lagrange-NG inferred different optima for model parameters, with BioGeoBEARS achieving a slightly better log-likelihood score (−216.127 vs. −224.396). It is unclear if these likelihoods are directly comparable. Nonetheless, this does not affect the respective qualitative results as BioGeoBEARS and Lagrange-NG agree on the most likely distribution for every node.

The analysis of this dataset with no region limit using Lagrange-NG produced similar results to the analysis with the 5-region limit, albeit with a better likelihood (−217.023). The time for this analysis was about 2 s using 8 cores.

Comparing runtimes with RevBayes is difficult, as RevBayes and Lagrange-NG produce different kinds of results. RevBayes is a Bayesian analysis software, and as such produces a distribution of parameter values as the posterior. On the contrary, Lagrange-NG finds only the parameter values with the highest likelihood. Nonetheless, the limited range of available tools to compare with necessitates us to use RevBayes as a comparison. We let RevBayes run for ≈ 7 h on the sloth dataset, and in that time RevBayes managed to perform ≈ 30 iterations, which is ≈ 14 min per iteration. Using 3000 iterations as the default number of iterations, a number of iterations suggested by the tutorial for DEC analysis using RevBayes, the full analysis would take ≈ 29 d.

Discussion

We have shown that computation of likelihood-based biogeographical models can be greatly accelerated without sacrificing result quality. An 8-fold increase in speed over the original implementation, and a 28-fold increase in speed over BioGeoBEARS represents a step forward, especially when taking the time complexity of the matrix exponential into account. In addition, we retain this speed even on datasets with a large number of regions and no region limit, enabling for more fine grained as well as exploratory analyses of biogeographical data.

Readers might wonder why the execution time for the analysis of the empirical dataset with a maximum number of regions is the slower than the analysis without a maximum number of regions. Ostensibly, a smaller number of regions should lead to a faster execution, but the runtimes shown contradict this. However, for this specific dataset, when using a maximum number of regions Lagrange-NG's adaptive mode detected numerical issues on nearly all results involving the matrix exponential, and therefore had to fall back to the slower, but safer, method of computing results. Indeed, if we force Lagrange-NG to use the faster mode computing results using the matrix exponential, the numerical errors are so excessive that a result cannot be computed. As a happy accident, this showcases the utility of Lagrange-NG's adaptive mode, where it was able pick the best method of computation without intervention from the user.

However, the parallel efficiency of 0.5, or even lower as core counts increase, is rather sub-optimal. Yet, as detailed in the Supplementary Material, the parallel efficiency increases with increasing taxon and region numbers (strong scaling). This means that, as datasets become more taxon rich or as the size of the transition matrix grows, and run-times increase, Lagrange-NG will utilize its parallel computational resources more efficiently.

Future work on Lagrange-NG includes extending the range of models that can be computed by the DIVA/DIVALIKE and BAYAREA family of models (Ronquist, 1997; Sanmartín et al., 2001; Landis et al., 2013). In addition, the current models can be further optimized in three areas, although we expect unspectacular performance improvements.

We produced a version of Lagrange-NG that utilized GPU acceleration for the matrix computations. Unfortunately, this method failed to produce acceptable speedups even for large datasets (10–11 regions). This is in line with the performance results of previous attempts to accelerate likelihood computations for phlyogenetic tree inference on GPUs (Izquierdo-Carrasco et al., 2013). The fundamental difficulty is that the tree-based nature of the computation that induces a decreasing degree of parallelism as we approach the root, leaves many computational units starved for work, as is the case with the existing CPU-based course-grained parallization. It is possible that further development would produce better results, but we believe that by the time that datasets become large enough to observe large speedups, the analysis will simply be infeasible due to the exponential nature of the problem.

For remaining improvements to the computational efficiency of Lagrange-NG, the first is to further refine the matrix exponential routine. While the current implementation is extremely fast, the implementation in Lagrange-NG has not been thoroughly optimized for this particular use case. In addition, one could further refine the work allocation for the coarse grained parallelization approach. The current method of assigning tasks is straight-forward, and can be improved upon by becoming aware of which nodes are "most blocking" of other tasks. It might be possible to devise an algorithm that can minimize the "task starved" period of computation, either as a clever assignment method, or as a so-called "work stealing" scheme.

Supplementary Material

Data available from the Dryad Digital Repository: http://dx.doi.org/10.5061/dryad.mw6m905zv

Acknowledgments

We would like to thank Qihao Yuan for his work implementing the Krylov subspace-based implementation of the matrix exponential. We would also like to thank Lucas Czech for his help on the results plotting tool for Lagrange-NG.

Data Availability

The software, tools, and data used for this paper are available online at https://github.com/computations/lagrange-ng.

Funding

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 764840. In addition, this work was funded by the Klaus Tschira Foundation. The funding sources had no influence on topic choice, experimental design, analysis, or interpretation of the results in this article.

References

Evans S.N., Matsen F.A. 2012. The phylogenetic Kantorovich–Rubinstein metric for environmental sequence samples. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 74:569–592.

Intel Math Kernel Library. 2022. Developer Reference for Intel® oneAPI Math Kernel Library - C.

Izquierdo-Carrasco F., Alachiotis N., Berger S., Flouri T., Pissis S. P., Stamatakis A. 2013. A Generic Vectorization Scheme and a GPU Kernel for the Phylogenetic Likelihood Library. Pages 530–538 *in* 2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum.

Johnson S. G. 2021. The nlopt nonlinear-optimization package. https://nlopt.readthedocs.io/en/latest/Citing_NLopt/.

Landis M.J., Matzke N.J., Moore B.R., Huelsenbeck J.P. 2013. Bayesian Analysis of Biogeography when the Number of Areas is Large. Syst. Biol. 62:789–804.

Matzke N. J. 2013. BioGeoBEARS: BioGeography with Bayesian (and Likelihood) Evolutionary Analysis in R Scripts. University of California, Berkeley Berkeley, CA.

Moler C., Loan C.V. 2003. Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later. SIAM Rev. 45:3–49.

Nelder J.A., Mead R. 1965. A Simplex Method for Function Minimization. The Computer Journal 7:308–313.

OpenBLAS: An optimzed BLAS library. 2022. OpenBLAS: An optimized BLAS library.

Ree R.H., Moore B.R., Webb C.O., Donoghue M.J. 2005. A Likelihood Framework for Inferring the Evolution of Geographic Range on Phylogenetic Trees. Evolution 59:2299–2311.

Ree R.H., Smith S.A. 2008. Maximum Likelihood Inference of Geographic Range Evolution by Dispersal, Local Extinction, and Cladogenesis. Syst. Biol. 57:4–14.

Ronquist F. 1997. Dispersal-Vicariance Analysis: A New Approach to the Quantification of Historical Biogeography. Syst. Biol. 46:195–203.

Sanmartín I., Enghoff H., Ronquist F. 2001. Patterns of animal dispersal, vicariance and diversification in the Holarctic. Biol. J. Linn. Soc. 73:345–390.

Varela L., Tambusso P.S., McDonald H.G., Fariña R.A. 2019. Phylogeny, Macroevolutionary Trends and Historical Biogeography of Sloths: Insights From a Bayesian Morphological Clock Analysis. Syst. Biol. 68:204–218.

Vaserstein L.N. 1969. Markov Processes over Denumerable Products of Spaces, Describing Large Systems of Automata. Problemy Peredači Informacii 5:64–72.

Waskom M.L. 2021. seaborn: statistical data visualization. Journal of Open Source Software 6:3021.

Zapletal A., Höhler D., Sinz C., Stamatakis A. 2021. The SoftWipe tool and benchmark for assessing coding standards adherence of scientific software. Sci. Rep. 11:10015.