Preliminary Results in Replacing a Sorted Array With a Splay Tree in a Memory Intensive Program

Kevin Spires
Computer Science
Bemidji State University
kspires@mail.com

Abstract

ATrExML is a phylogenetic program that uses maximum-likelihood analysis of nucleotide sequences to produce evolutionary trees. Due to the time-consuming nature of maximum-likelihood analysis of nucleotide sequences, increasing the efficiency of ATrExML is essential to improving the quality of the results produced by the program. We predicted we could improve the runtime performance of ATrExML by replacing an extensively used data structure, a sorted array, with a more efficient data structure, a splay tree. The data structure in question stores the evolutionary trees that are being considered as the program executes. The correctness of the new code was verified by running both the original and modified code on the same data sets (i.e. nucleotide sequences), and by verifying that they both produced the same output. We timed each execution and preliminary results indicate that our hypothesis was correct; we have achieved runtime performance increases of up to 63%.

Introduction

The goal of our research project is to increase the runtime efficiency of ATrExML (Stamatakis et al., Aug 2002; Stamatakis et al., Nov 2002), a phylogenetic program for maximum-likelihood analysis of nucleotide sequences. ATrExML generates a large number of evolutionary trees by employing maximum-likelihood analysis to examine nucleotide sequences. Some of the evolutionary trees that are produced during this analysis are more likely statistically to be supported by the nucleotide data than others that are produced. The goal of ATrExML is to find as many evolutionary trees as possible that are not statistically significantly different from the best tree found. ATrExML was created by modifying TrExML (Wolf et al., 2000), which in turn was originally derived from fastDNAml (Olsen et al., 1994). The goal of fastDNAml is to locate only the single most likely evolutionary tree supported by the nucleotide sequences that it analyzes. fastDNAml uses a sorted array to hold candidate trees (typically hundreds or a few thousand) during the computation process.

Due to the time-consuming nature of maximum-likelihood analysis of nucleotide sequences, increasing the efficiency of ATrExML is essential to improving the quality and quantity of the results produced by the program. We hope to improve the runtime performance of the program by replacing an extensively used data structure, a sorted array, with a more efficient data structure, a splay tree. The data structure in question stores the evolutionary trees that are being considered as the program executes. ATrExML inherited this sorted array from its predecessor, fastDNAml. This sorted array efficiently serves fastDNAml in achieving its goal, but ATrExML requires more extensive use of this data structure. The divergent goal of ATrExML necessitates changing the data structure to optimize efficiency.

Modifications

Choosing a more efficient data structure

ATrExML uses two “BestList” structures to store the numerous evolutionary trees that are generated and manipulated throughout the execution of the program. The BestLists are of fixed size, as determined by the sequence file that is being used. As the program is executed, evolutionary trees are removed from the first BestList one at a time. After a tree is removed it is used to generate a collection of new trees. Each new tree is evaluated and it is stored in the second BestList if it is more likely than the worst tree in the second BestList (or if the second BestList is not full). Both BestLists are kept in sorted order, from the most likely tree to the least likely tree. Since the BestLists are sorted in this manner, trees near the beginning of the first BestList are more likely to be precursors of trees in the second BestList.

The BestList structure relies on its sorted array component to physically store the evolutionary trees that are being considered as the program executes. Only two operations are performed on this array. Insertions are the most common operation. An insertion into the array requires O(n) pointer updates. The other operation involves traversing the array from the most likely evolutionary tree to the least likely evolutionary tree. With a sorted array, locating the next evolutionary tree during a traversal should take O(1) time. However, the actual implementation in ATrExML (inherited from fastDNAml) uses binary search to locate the next evolutionary tree. Thus, traversing from one evolutionary tree to the next is inefficient, taking O(log n) time. The total time taken to traverse the array is O(n log n) time.

Improving the efficiency of inserting evolutionary trees into the BestList was our primary concern. Since ATrExML has tendencies to access similarly valued trees in order (i.e., it will work with the beginning or end of the list for a while), we conjectured that replacing the array with a splay tree would allow the program to perform more efficiently. Splay trees are self-adjusting binary search trees. Any operation performed on a splay tree causes it to rearrange itself to keep the most recently accessed values near the root of the tree. Splay trees offer optimal performance when nodes that are close together are accessed subsequently. Operations on a splay tree have an amortized run time of O(log n). In particular, locating any node in the splay tree requires O(log n) amortized time. For more information on splay trees, see any standard text on data structures and algorithms (e.g., Goodrich and Tamassia 2002, p. 185).

Implementation

To ease discussion, we will refer to the splay tree version of ATrExML as SATrExML. To create SATrExML, we replaced the sorted array component of the BestList structure with an implementation of a splay tree. Next, we had to identify all functions that utilized the previously existing data structure. All of these functions were then modified to interface appropriately with the splay tree implementation. In addition, we added routines to carry out the required splay tree rotations.

It was also important to optimize the splay tree's memory usage. ATrExML does not release the large blocks of memory necessary for storing evolutionary tree arrangements once they have been initialized; it reuses the same evolutionary tree structures to improve efficiency. SATrExML's splay tree was designed to mimic ATrExML's sorted array in regards to this issue. SATrExML also reuses the evolutionary tree structures to improve efficiency.

Materials and methods

Experimental data

The performance of the SATrExML was compared to that of ATrExML (Version 1.1.0) using small ribosomal subunit RNA (16S-rRNA) gene sequences from 13 Archaebacteria and three Eubacteria. The sequences were originally retrieved in aligned format from the Ribosomal Database Project's (Maidak et al., 1997) World Wide Web site (http://www.cme.msu.edu/RDP/html/index.html) and consist of (with RDB codes and accession numbers in [...;...]): *Methanospirillum hungatei* [Msp.hungat; M60880], *Halococcus morrhuae* [Hc.morrhua; X00662], *Methanothermus fervidus* [Mt.fervid1; M32222], *Thermococcus celer* [Tc.celer; M21529], *Pyrodictium occultum* [Pyr.occult; M21087], *Methanococcus jannaschii* [Mc.jannasc; M59126], *M. voltae* [Mc.voltae; M59290], *Methanobrevibacter arboriphilicus* [Mbb.arbori; C.R. Woese, unpublished], *Methanolobus oregonensis* [Mlo.oregon; U20152], *Haloferax volcanii* [Hf.volcani; K00421], *Pyrococcus abyssi* [Pc.abyssi; L19921], *Methanopyrus kandleri* [Mpy.kandl1; M59932], *Pyrobaculum aerophilum* [Pyb.aeroph; L07510], *Aquifex pyrophilus* [Aqu.pyrop; M83548], *Escherichia coli* [E.coli; J01695], and *Bacillus subtilis* [B. Subtilis; K00637, M10606, X00007]. Columns containing gaps or ambiguous characters were removed from the alignment yielding a revised alignment with 1200 sites. Finally, because the divergence between the species most likely spans more than a billion years, the nucleotides were recoded to purines R = A or G) or pyrimidines (Y = C or T). We chose this initial data set to test SATrExML because it was also used to test TrExML and ATrExML when they were first created.

Hardware and software

We conducted the tests on 16 sequences and recorded the run-time, as that given by the gettimeofday function in the time.h library, on a 1 Gb DDR RAM, two AMD Athlon (tm) MP 2000+ (only one processor was used for these experiments) system with the Redhat Linux 8.0 (2.4 SMP Kernel) operating system. Executable code was generated using the GNU C compiler (Version 3.2) with the -O3 compiler option. We also performed functional analysis with gprof, requiring the -pg compiler option. For all tests, the A parameter of the sequence files was set at 8. The A parameter determines the number of species for which the tree-space is fully explored. We varied the K parameter of the sequence files from 5,000 to 105,000 in increments of 5000. The K parameter determines the maximum number of trees of a given size that are kept in the BestList data structure.

Comparative metrics

The performance of SATrExML was compared to that of ATrExML by examining the total CPU time taken by the program. Measurements were taken with the gettimeofday function, a standard C function that can be used to measure CPU time. Only the K parameter, which determines the maximum number of trees kept, was modified with each run. All other parameters were held constant throughout the series of tests. We also examined function CPU time with gprof. We verified the correctness of SATrExML by asserting that the output produced by both programs was identical with the Unix diff command.

Table 1. Runtime Efficiency Improvements Based on CPU Time

Maximum number of trees kept	SATrExML CPU time in seconds	ATrExML CPU time in seconds	Efficiency increase percentage	Maximum number of trees kept	SATrExML CPU time in seconds	ATrExML CPU time in seconds	Efficiency increase percentage
5000	1022.90	1075.01	5	60000	11530.43	21500.08	46
10000	2032.81	2267.56	10	65000	12486.65	24311.63	49
15000	2981.87	3473.34	14	70000	13396.62	27416.45	51
20000	3916.57	4819.00	19	75000	14419.64	30738.89	53
25000	4877.59	6307.47	23	80000	15315.08	34288.08	55
30000	5811.59	7973.33	27	85000	16317.91	38398.71	58
35000	6775.91	9859.40	31	90000	17187.88	41187.59	58
40000	7713.34	11878.16	35	95000	18191.45	44775.97	59
45000	8701.70	14197.87	39	100000	19148.24	50180.13	62
50000	9622.58	16382.67	41	105000	20150.52	54598.25	63
55000	10606.38	18761.42	43				

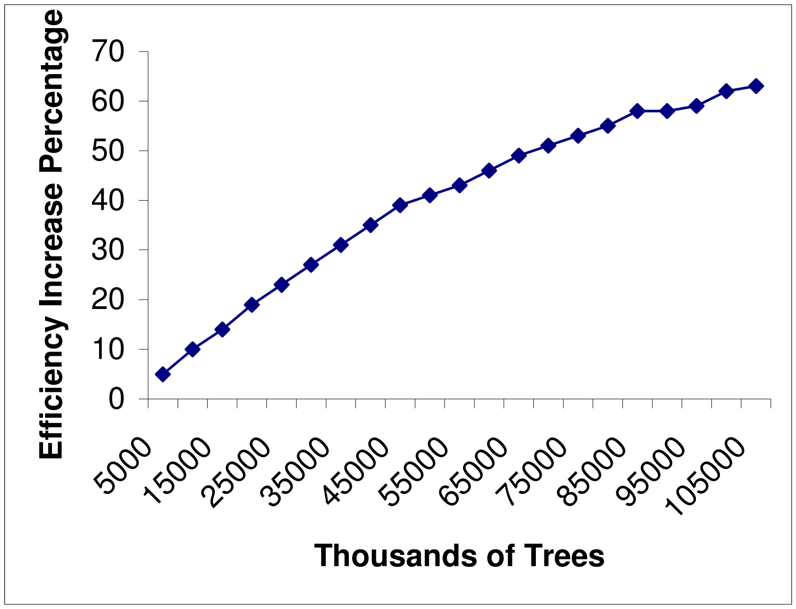

Figure 1. Runtime Efficiency Improvement of SATrExML over ATrExML

Table 2. Expected Runtime Efficiency Improvements Based on Functional Analysis

Maximum number of trees kept	Expected efficiency increase percentage	Actual efficiency increase percentage
5000	**3.41**	**5**
10000	**8.31**	**10**
15000	**8.08**	**14**
20000	**8.73**	**19**
25000	**6.98**	**23**
30000	**4.12**	**27**
35000	**5.65**	**31**
40000	**6.55**	**35**
45000	**6.33**	**39**
50000	**5.68**	**41**
100000	**5.84**	**62**

Results and discussion

Preliminary results

The correctness of the new code was verified by running both the original and modified code on the same data sets (i.e. nucleotide sequences), and by verifying that they both produced the same output. We timed each execution and preliminary results indicate that our hypothesis was correct; by replacing the array with an implementation of a splay tree, we have achieved runtime performance increases of up to 63% (Table 1).

Initially, the performance increases were inconsistent and unpredictable. Running the tests on different systems produced dramatically different results. Additionally, the modified code did not always outperform the original code. We believe that these runtime fluctuations were due to the extensive memory usage of SATrExML (at times greater than 500 MB) and because the splay tree was not optimized to efficiently acquire and release memory. ATrExML reuses evolutionary tree structures to increase efficiency. Our first implementation of SATrExML did not reuse evolutionary tree structures, and run-time fluctuations occurred. After optimizing the splay tree's memory usage, by reusing evolutionary tree structures, SATrExML became more predictable and efficient. We were able to develop a more consistent runtime performance increase by optimizing the splay tree's memory usage.

Based on functional analysis (Table 2), we would only anticipate runtime performance increases of up to 9%. We would also expect the performance increase to stay relatively constant as the number of evolutionary trees kept is increased. The actual results that we experienced exhibit roughly linear growth as the number of evolutionary trees kept is increased (Figure 1). We believe that the splay tree performs better with the memory management system of the operating system (i.e., paging and caching) than a sorted array does, especially as the size of the structure increases. Preliminary testing of SATrExML's performance on systems with small memory, where extensive swapping is required, suggests that SATrExML performs poorly in this setting. Further tests are being conducted to determine whether this hypothesis is correct.

Future Work

Future work will involve more extensive testing of SATrExML to validate the correctness of the program and to provide additional verification of our preliminary results (i.e., efficiency increase). We plan to test SATrExML with additional data sets that include varying numbers of sequences, sites, and categories. We also need to test the compatibility of SATrExML with other untested ATrExML options. Another interesting question centers on the origin of evolutionary trees in the second BestList. Do they always come from trees ranked near the beginning of the first BestList? If so, substantial computation time savings could be achieved by processing only a percentage of the evolutionary trees from the first BestList.

References

Goodrich, M. T., Tamassia, R. (2002).
Algorithm design: Foundations, analysis, and internet examples, John Wiley & Sons, New York.

Olsen, G., Matsuda, H., Hagstrom, R., & Overbeek, R. (1994).
Fastdnaml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood, *Comput. Appl. Biosci.*, 10:41-48.

Stamatakis, A., Ludwig, T., Meier, H., & Wolf, M. J. (Aug 2002).
Efficient use of subtree equality vectors for the acceleration of sequential and parallel phylogenetic tree calculations based on the maximum likelihood method, *Proceedings of the IEEE Computer Society Bioinformatics Conference.*

Stamatakis, A., Ludwig, T., Meier, H., & Wolf, M.J. (Nov 2002).
Accelerating parallel maximum likelihod-based phylogenetic tree calculations using subtree equality vectors, *SuperComputing 2002.*

Wolf, M. J., Easteal, S., Kahn, M., McKay, B., & Jermiin, L. (2000).
Trexml: A Maximum likelihood program for extensive tree-space exploration, *Bioinformatics*, 16(4):383-394, 2000.