LSHPlace: Fast phylogenetic placement using locality-sensitive hashing

DANIEL G. BROWN* and JAKUB TRUSZKOWSKI
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo ON N2L 3G1 Canada
browndg,jmtruszk@uwaterloo.ca

We consider the problem of phylogenetic placement, in which large numbers of sequences (often next-generation sequencing reads) are placed onto an existing phylogenetic tree. We adapt our recent work on phylogenetic tree inference, which uses ancestral sequence reconstruction and locality-sensitive hashing, to this domain. With these ideas, new sequences can be placed onto trees with high fidelity in strikingly fast runtimes. Our results are two orders of magnitude faster than existing programs for this domain, and show a modest accuracy tradeoff. Our results offer the possibility of analyzing many more reads in a next-generation sequencing project than is currently possible.

Keywords: Phylogenetic placement; Nearest neighbour search;

1. Introduction

In the past few years, advances in sequencing technology have enabled the study of microbial communities from diverse environments, such as soil,[1] ocean,[2] and the human body.[3] Microbial ecologists are interested in the diversity of bacteria in a given environment, their evolutionary origins, and their metabolic relationships. They answer these questions by collecting sequence data from environmental samples and then comparing them to reference sequences from known microbial lineages.

Phylogenetics provides a natural framework for investigating the microbial diversity in these environments. Most microorganisms can be approximately located on the tree of life for bacteria, and then communities or environments can be characterized by the relative abundance of certain taxa.[4] Or, unusual sequences can be a focus for further investigation and directed sequencing.[5] The first step, however, is to locate each sequence on the tree.

While many phylogenetic algorithms have been developed over the years, the current flood of sequence data from next-generation sequencing presents new challenges for traditional methods. The massive amounts of data generated by NGS make traditional phylogenetic inference computationally prohibitive; indeed, in metagenomic contexts, a common first step is to cluster a data set, possibly consisting of millions of reads and instead analyze just the hundreds or thousands of cluster centres, which discards much valuable data.[4] Another problem is that environmental sequencing produces short sequence reads, instead of full gene sequences. This is partly due to inherent difficulties of assembling reads in the presence of sequences from many different species. For example, reads generated by Illumina have length ≈ 200 bp, which does not provide sufficiently strong phylogenetic signal for full phylogenetic inference. Maximum likelihood phylogenies from incomplete sequences tend to be biased towards grouping highly overlapping sequences together.[6]

These problems have recently motivated researchers to focus on placing individual environmental sequences into a fixed phylogeny, instead of performing full phylogenetic inference

on the entire set of sequences. This has several advantages. The computational cost is greatly reduced, as the number of topologies that need to be considered is linear in the size of the tree for each sequence. By considering each query sequence separately, we also hope to avoid the biases associated with sequence overlaps. There are currently several programs performing this task, called *phylogenetic placement* .[6–8] Unfortunately, their speed is insufficient to place millions of reads, as we see in Section 4.3.

Recently, we have developed a fast and accurate phylogenetic reconstruction algorithm.[9] Our algorithm uses hash tables to identify closely related sequences without having to estimate all $\binom{n}{2}$ distances between sequences. Here, we adapt our technique to phylogenetic placement. More specifically, we develop the first algorithm that places sequences in a known reference phylogeny with running time sub-linear in the number of taxa in the reference tree. We show that our methods are both theoretically grounded and practically useful.

Our algorithm is based on several ideas. First, we use a hash table technique known as locality-sensitive hashing[10] to find a sequence in the tree that is close to the query sequence, in sublinear time. To make sure that such a sequence exists in the tree, we infer ancestral sequences at internal nodes of the reference tree. Finally, we use local search to find the optimal placement of the query sequence in a neighbourhood of the tentative placement discovered by locality-sensitive hashing. The running time of this procedure is determined by the number of hash tables needed to find a close enough sequence in the tree, which in turn depends on the lengths of the edges of the tree. Specifically, if p is an upper bound on the mutation probability on any edge, and $p < 1/2 - \sqrt{1/8}$, then we show that we can do the locality-sensitive hashing, which is the runtime-determining step, in $O(n^{\gamma(p)} \log^2 n)$ time for each sequence to place, where $\gamma(p)$ is always less than 1; the overall runtime is thus $O(mn^{\gamma(p)} \log^2 n)$ to place m new sequences onto a reference tree of n taxa. In practice, we choose to use a constant number of hash tables, reducing this phase to $O(\log n)$ time, though there is some runtime required for local search.

A novel algorithmic idea in this paper is to build the hash tables from slowly-evolving alignment columns. This reduces the probability of a mutation having occurred between two sequences in one of the hash table columns, which causes the algorithm to require fewer hash tables to guarantee a hash table collision.

We evaluate our algorithm on a number of synthetic and real data sets. The accuracy of our algorithm is lower than that of pplacer,[6] while its running time is around 2 orders of magnitude faster, making it a useful tool for handling large data sets. The current implementation of the local search phase of our method is distance based, and we expect that its accuracy could be substantially improved using maximum likelihood, at modest cost in running time.

2. Related work

Many tools determine the taxonomic origin of environmental sequences. These tools can be roughly divided into three categories: those based on phylogenetic modelling, those based on sequence composition, and those based on homology search. Gerlach[11] provides a recent survey of these methods.

Recently, researchers have developed several tools for placing environmental sequences onto a reference phylogeny. These methods generally take $O(n)$ time to insert a sequence

in a tree of n taxa. MLTreeMap[8] was the first tool designed for this task. the Evolutionary Placement Algorithm[7] and pplacer[6] offer more efficient implementations of the same approach, which places a sequence optimally at each edge of the phylogeny, and then assigns the overall maximum likelihood placement as the answer for each individual sequence.

Some software pipelines for analyzing metagenomic data sets employ full phylogenetic reconstruction. These include TreePhyler[12] and CARMA.[13] While much progress has been made in fast phylogeny reconstruction in recent years,[9,14,15] reconstructing the full tree remains much slower than other classification methods.

Another approach is to build a classifier to discriminate between taxonomic groups at different levels of the Linnaean hierarchy. These classifiers do not attempt to model phylogenetic relationships between different taxa, but instead treat the problem as a supervised classification problem at each level of the hierarchy. The features used are usually derived from the k-mer composition of the sequence, which bypasses the need for aligning sequences. Many such classifiers have been developed, including PhyloPythia,[16] TACOA,[17] and PhyMM.[18] Taxy[19] uses mixture models and k-mers to estimate the relative abundance of different taxonomic groups in a set of sequences, without attempting to classify each sequence in detail. Taxonomic classifiers are often faster than phylogeny-based methods, but they do not offer the same explanatory power. Moreover, classifiers based on k-mers tend to behave badly on short sequences, as they lack sufficient information to distinguish different clades.

Yet another class of approaches uses BLAST[20] to determine evolutionary proximity of sequence reads to known species. Here, environmental sequences are expected to generate BLAST hits with the sequences they are closely related to. Several algorithms exist for mapping sets of BLAST hits to taxonomic classifications, including MEGAN[21] and SOrt-ITEMS.[22] Unfortunately, if the only close relatives to a sequence in a tree are internal nodes, this approach will fail; our method will avoid this problem due to our inference of internal sequences. The BLAST-based approach is also very fragile to short reads.

3. The algorithm

3.1. *Overview*

The input to the algorithm consists of three parts: the reference phylogeny T, on n taxa, which includes tree edge lengths; the multiple alignment A of the n sequences in the reference phylogeny; and a set of m query sequences, each aligned to that reference alignment. Our goal is to assign each query sequence to the edge where it joins the tree.

Our algorithm consists of the following steps. The first three are a preprocessing phase, and the resultant data structures could be stored for use, if a given tree and multiple alignment are going to be used to analyze many different sets of reads.

(1) Estimate evolutionary rates for each column of the reference alignment.
(2) Reconstruct ancestral sequences at each internal node of the reference phylogeny using maximum likelihood.
(3) Build a collection of hash tables, keyed on slowly-evolving columns of A. Add the keys for both leaf sequences and ancestral inferences into the hash tables.

(4) For each query sequence y:

(a) Look for collisions between y and the hash tables. Choose the closest sequence x colliding with y.
(b) Examine the neighbourhood of x in T. For each edge e in that neighbourhood, estimate the distance between e and y. Output the closest edge to y.

In what follows, we explain the details and the motivation behind each of these steps. For a more detailed discussion of the theoretical issues in this method, the reader is referred to our previous paper on phylogeny reconstruction.[9]

3.2. *Locality-sensitive hashing*

For a given query sequence y, we want to find sequences in T whose distance to y is small. We use a classical result by Indyk and Motwani[10] who solve a similar problem using a collection of randomized hash tables. We design hash tables so that the probability of y colliding with a similar sequence is high, and the probability of colliding with a distant sequence is low. We independently construct many such hash tables so that the probability y collides with a close sequence in at least one hash table is high. *Locality-sensitive hashing* has been applied to many problems in bioinformatics, such as motif finding.[23]

Specifically, Indyk and Motwani solve a related problem, the (r_1, r_2)-approximate Point Location in Equal Balls ((r_1, r_2)-PLEB):

Input: A set of n sequences P in $\{0,1\}^k$, a query sequence q, and radii $r_1 < r_2$

Output: Does there exist a sequence $p \in P$ within Hamming distance $r_1 k$ from q? If so, output "yes" and a sequence within $r_2 k$ of q. If there is no sequence in P within Hamming distance $r_2 k$ from q, output "no". Otherwise, output either "yes" or "no".

Indyk and Motwani's solution constructs n^{r_1/r_2} hash tables, each keyed on $c \log n$ randomly chosen sequence positions, where c depends only on r_2. Given y, a point within distance r_1 of it has probability at least n^{-r_1/r_2} of colliding with y in each hash table, while points further than r_2 from q have $O(1/n)$ probability of colliding. After inspecting a constant number of collisions with q, we can find, with constant probability, a point whose distance from q is at most r_2; if we boost by running $O(\log n)$ times independently, the success probability is $1 - n^{-\alpha}$, for any choice of α. Finding an (r_1, r_2)-approximate near neighbour for a query point q with high probability takes $O(n^{r_1/r_2} \log n)$ hash table lookups, each on a key of length $O(\log n)$ bits.

For large trees, parameter r_2 can be very close to the expected Hamming distance of unrelated sequences.[9] For binary characters, r_2 converges to $\frac{1}{2}$; similar constants can be computed for other mutation models, alphabets, and baseline letter frequencies.

Finding all neighbours within r_1 normalized Hamming distance of a sequence y thus takes $O(n^{2r_1+\epsilon} \log^2 n)$ time with high probability. The additive constant ϵ, which converges to 0 as n grows, accounts for error in distance estimates and that r_2 converges to 1/2 as n grows. We use $O(n^{2r_1+\epsilon} \log n)$ hash tables, each of which requires $O(\log n)$ time to examine, and we take $O(\log^2 n)$ time examining the hash table hits.

3.3. *Ancestral states*

For locality-sensitive hashing to work, we have to ensure that our hash tables contain sequences sufficiently similar to the query sequence. If the query sequence branches out from an edge in the reference phylogeny that is not adjacent to a leaf, its distance from any leaf sequence may be too long for locality-sensitive hashing to work if we only index leaf sequences. We reconstruct ancestral sequences in the reference tree, using maximum likelihood, and add these reconstructions to the tables as well. The crucial question is how well the reconstructed ancestral sequences resemble the actual historical sequences: if a query is near an internal node in the true tree, and that node's sequence has been reconstructed well, it will likely result in a hash table hit, even with few hash tables.

There are several known upper bounds on the probability of incorrectly reconstructing an ancestral character from the leaf characters. This error probability depends on the edge lengths in the phylogeny. If most edges in T are very long, the number of errors in the reconstructed sequences will grow with the level of the internal node, and the reconstructed states at deep nodes of the tree will be junk, so adding them to the hash table will be pointless. On the other hand, if the edges are short enough, speciation outpaces mutation, and it is possible to reconstruct the ancestral state with guaranteed accuracy that does not depend on the size of the tree.

Evans *et al.*[24] have shown that for Cavender-Farris characters, if all the edge lengths correspond to mutation probabilities less than $\frac{1}{2} - \sqrt{\frac{1}{8}}$, then internal sequences will be reconstructed with accuracy at least a constant strictly above $\frac{1}{2}$. Similar results exist for more realistic evolutionary models, including the GTR model for DNA sequences.[25] Gascuel and Steel[26] established a similar result for trees generated from the birth-death process. This is an important complement to the result by Evans *et al.*, since birth-death trees will usually have a limited number of long edges. While the mathematical details of these results differ depending on the evolutionary model and branch length distribution, they all suggest the possibility of reasonably accurate ancestral sequence reconstruction for trees whose branches have moderate lengths.

The following is a useful bound by Steel.[27]

Theorem 3.1. *Let T be a phylogenetic tree where all mutation probabilities across edges are equal to $p_g < 1/8$. The probability p_{err} of incorrectly reconstructing the root state using Fitch parsimony is bounded by*

$$p_{err} < \frac{1}{2} - \frac{\sqrt{(1-4p_g)(1-8p_g)}}{2(1-2p_g)^2} < 1 - 4p_g$$

We have shown[9] that this bound also applies to trees with variable edge lengths if Felsenstein's maximum likelihood algorithm is used, rather than Fitch parsimony. Felsenstein's algorithm has optimal probability of correctly reconstructing ancestral states among all possible algorithms.

With the bound on p_{err}, we can determine the number of hash tables (and thus the runtime) required to find a node in the tree that is close to the true placement of the query sequence

y. Suppose sequences evolve according to the Cavender-Farris model. Let g_{err} be the distance corresponding to a mutation probability of p_{err}. If the evolutionary distance between y and the node that joins it to the tree is g_{query}, the effective evolutionary distance between the query and the nearest sequence in the tree is at most $g_{query} + g/2 + g_{err}$. This corresponds to a mutation probability of $\frac{1}{2} - \frac{1}{2}(1-2p_{query})(1-2p_g)^{1/2}(1-2p_{err})$. The number of hash tables required is thus bounded by

$$n^{1-(1-2p_{query})(1-2p_g)^{1/2}(1-2p_{err})}$$

Table 1 shows the running times of the hashing time for a single query, for different values of p_g, assuming for simplicity that $p_{query} \leq p_g$.

In practice, we do not know the upper bound on the distance of query sequences to the tree. The values in Table 1 should be understood only as illustration of the theoretical basis for our method. In the current version of the software, the number of hash tables per alignment region is fixed at 4, as this value gave good results for phylogenetic tree reconstruction, and maintains moderate memory use.

Table 1. Runtime of inserting a new taxon into a tree with n taxa, as a function of the maximum edge mutation probability

p_g	0.01	0.02	0.05	0.075	0.10
runtime	$n^{0.05}\log^2 n$	$n^{0.10}\log^2 n$	$n^{0.27}\log^2 n$	$n^{0.43}\log^2 n$	$n^{0.61}\log^2 n$

3.4. *Local search*

Our near-neighbour search procedure finds sequences that are in the vicinity of the correct placement for query sequence y, with high probability. However, it does not guarantee that the optimal placement is one of the edges adjacent to the node found. We estimate the distance between y and each edge $e = (x_1, x_2)$ near of the colliding sequence x as:

$$\hat{d}(y, e) = \frac{1}{2}(\hat{d}(x_1, y) + \hat{d}(x_2, y) - \hat{d}(x_1, x_2)),$$

estimating $\hat{d}(a, b)$ via the probabilistic model of evolution.

If x_1 and x_2 are reconstructed ancestral sequences and reconstruction errors at x_1 and x_2 are independent, then they will not bias the estimate $\hat{d}(y, e)$, as the additive terms in distance estimates associated with the reconstruction error will cancel out. We examine all edges within distance $\hat{d}(x, y)$ of node x. The edge e^* with smallest estimate is chosen as the placement of y, with pendant edge length $\hat{d}(x, e^*)$. If T has many very long and very short edges, this will prove slow; in practice this situation is quite rare. If we assume a minimum size for each edge, and a maximum size beyond which we consider $\hat{d}(x, y)$ too long to be meaningfully estimated, the neighborhood is of constant size.

For simplicity of implementation, the current implementation of the algorithm ignores possible dependencies between reconstruction errors. These dependencies could be ameliorated by using some of the techniques discussed in our previous paper.[9] We leave that as future work.

3.5. *Accommodating for different read locations*

In some cases, the sequences being placed are short reads, much smaller than the total length of the alignment, and their positions are distributed randomly across the alignment.[28] We need multiple sets of hash tables to make sure that each read is covered by at least one set of LSH tables. We solve this problem by using a sliding window approach. We construct groups of hash tables, each corresponding to short regions of the alignment, so that each query sequence maps to at least one set of hash tables. Specifically, if all reads have at least k' bases, we divide the alignment into blocks of length $k'/2$, and build a separate set of hash tables for each block. We then sort the reads by their starting position in the alignment and progressively process each block of hash tables, from the beginning to the end of the alignment. This ensures that we never have to store more than one set of hash tables in memory. While the sorting process adds a factor of $\log m$ to the running time per query sequence (where m is the number of query sequences), in practice this cost is negligible, and can be avoided with counting sort.

3.6. *Choosing slow-evolving sites*

So far, we have assumed all sites in the alignment evolve at the same relative rate. This is not true in practice, as genomes, proteins and RNAs contain conserved regions or individual sites. Rate heterogeneity across sites poses both statistical [29] and computational[30] challenges for phylogenetic inference. However, in our case, we can take advantage of rate heterogeneity across sites to improve the speed and accuracy of our algorithm. The running time of the LSH procedure depends on the effective mutation rate between the query sequence and the nearest sequence in the tree. By choosing hash table keys from slowly evolving columns, we reduce the number of hash tables needed to ensure a collision with the same probability, or, equivalently, we enable more distant sequences to be placed correctly using the same number of hash tables. This is particularly important since biologists are often interested in detecting previously unknown clades, many of whom are only distantly related to the known organisms.[5]

We identify slowly-evolving columns by estimating the maximum likelihood relative evolutionary rate for each column in the reference alignment, using the classical Newton-Raphson method. On the other hand, we also discard near-constant columns where the most common character appears in more than 95% sequences, as these are not informative.

An important question is how many slowest-evolving columns to choose. If too many columns are chosen, their average mutation rate will be relatively high. On the other hand, if we choose too few columns, this will lead to a huge variance in the Hamming distances on these columns, which will cause hashing to be less informative about the true evolutionary distances between sequences. We choose to randomly choose from the 200 slowest-evolving columns (not including the near-constant columns), or the bottom 50% slowest-evolving columns, for short alignments.

4. Experiments

4.1. *Data sets*

Our experiments used both simulated and real data. We generated synthetic short read data sets from a simulated 16S rRNA alignment on $n = 78132$ sequences from the FastTree paper.[14]

The length of the alignment was $m = 1287$ bases. The data were generated as follows. First, k taxa were chosen at random from the alignment, and the reference subtree induced by these taxa on the true tree was recorded. We then generated 100,000 simulated short reads. Each short read was generated by sampling with replacement from the $n-k$ sequences not included in the reference tree, and keeping m' contiguous positions, starting at a uniformly-chosen position. The read length m' was chosen from a Gaussian distribution with mean 200 and standard deviation 20.

For the real data set, we used a metagenomic 16S rRNA Illumina library from Alert, Nunavut, Canada.[31] The reads were located in the V3 variable region of the ribosomal RNA. The sequences were clustered at 97% identity, which resulted in 27848 sequences with a mean length of 152 bases, aligned to a reference alignment of 2759 sequences using Infernal.[32] For placement, we used a reference 16S tree from the Living Tree Project.[33] The diameter of the tree was 1.46 mutations per site, and the average distance between two nodes was 0.54.

4.2. *Accuracy*

Table 2 shows the results of our algorithm and pplacer on the synthetic data sets. Our algorithm was less accurate than pplacer, but placed reads within 3 edges from the correct location over 90% of the time. For larger trees, both algorithms tended to place sequences farther from their correct edges in terms of the topological distance (TD). However, the evolutionary distance (ED) between placements and correct locations tended to be lower for larger trees. Larger reference trees contained more short edges which were hard to distinguish for both algorithms. The magnitude of these effects was similar for LSHPlace and pplacer.

Table 2. Accuracy of LSHPlace and pplacer on three simulated data sets. LSHplace is less accurate, but still reasonably close for all data set sizes.

	data set								
	huge.1, 1000 taxa			huge.1, 5000 taxa			huge.1, 10000 taxa		
method	% correct	ED	TD	% correct	ED	TD	% correct	ED	TD
LSHPlace	51.2	0.054	1.12	48.6	0.033	1.17	46.1	0.028	1.34
Pplacer	79.0	0.011	0.30	74.0	0.007	0.39	69.1	0.006	0.51

The predictions of both programs disagreed much more on the real data set. Only 10% of reads were placed on the same edge by both Pplacer and LSHPlace. The average topological distance between a pplacer prediction and an LSHPlace prediction for the same sequence was 14 edges, with evolutionary distance 0.20 mutations per site. The medians of these distances were 10 and 0.15, respectively. We suspect that the accuracy of our algorithm might have been impacted by the presence of many near-constant sites in the alignment, which could have had an adverse effect on the accuracy of distance estimates. Obviously, future work is key to learning more about the accuracy difficulties we encounter with real data.

4.3. *Running times and scalability*

The running times for both programs are shown in Table 3. Each runtime corresponds to placing 100,000 reads into a tree of the given size. In all cases, LSHPlace is around 1.5 to 2

orders of magnitude faster than pplacer.

Table 3. The time to place 100000 taxa into a tree, as a function of the number of taxa in the tree

taxa	1000	5000	10000
our algorithm	7m	12m	17m
pplacer	3.8h	6.4m	18.8h

5. Conclusion

We have presented LSHplace, a new algorithm for phylogenetic placement. By using locality-sensitive hashing, and including inferred ancestral sequences in the hash tables, our algorithm allows us to approximately locate new sequences onto an existing phylogenetic tree extremely rapidly; a local-search procedure allows us to then find an optimal placement quickly for each new sequence. Our work can be used in a variety of domains, but we expect it will be especially useful in the context of metagenomic sampling, where millions of sequence reads are generated and analyzed at the same time to characterize environments. Experimental results, while preliminary, are encouraging, and show that our algorithm speeds up the process of placement by two orders of magnitude; we currently also take an accuracy penalty, but we expect this may be ameliorated by incorporating maximum likelihood inference into the program, which we are currently exploring. Future work will also explore the importance of alignment quality in the placement process.

6. Acknowledgement

We thank Andre Masella, Josh Neufeld and Michael Lynch for sharing data with us, and Erick Matsen for helpful conversations. This work is supported by the Government of Ontario, through an Early Researcher Award to DGB, and by the Natural Sciences and Engineering Research Council of Canada. JT is supported by a David R. Cheriton Scholarship at the University of Waterloo.

7. References

References

1. H. K. Allen, L. A. Moe, J. Rodbumrer, A. Gaarder and J. Handelsman, *ISME Journal* **3**, 243 (2009).
2. D. B. Rusch, A. L. Halpern, G. Sutton, K. B. Heidelberg *et al.*, *PLoS Biology* **5**, p. e77 (2007).
3. P. J. Turnbaugh, R. E. Ley, M. Hamady, C. M. Fraser-Liggett, R. Knight and J. I. Gordon1, *Nature* **449**, 804 (2007).
4. J. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman *et al.*, *Nature Methods* **7**, 335 (May 2010).
5. M. Lynch, A. K. Bartram and J. D. Neufeld, *ISME Journal* (2012), to appear.

6. F. A. Matsen, R. B. Kodner and E. V. Armbrust, *BMC Bioinformatics* **11**, p. 538 (2010).
7. S. A. Berger, D. Krompass and A. Stamatakis, *Systematic Biology* **60**, 291 (2011).
8. M. Stark, S. A. Berger, A. Stamatakis and C. von Mering, *BMC Genomics* **11**, p. 461 (2010).
9. D. G. Brown and J. Truszkowski, Fast reconstruction of phylogenetic trees using locality-sensitive hashing, in *Proceedings of WABI*, (Ljubljana, Slovenia, 2012).
10. P. Indyk and R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, in *Proceedings of STOC*, (New York, 1998).
11. W. Gerlach, Taxonomic classification of metagenomic sequences (2012), PhD thesis. Bielefeld University.
12. F. Schreiber, P. Gumrich, R. Daniel and P. Meinicke, *Bioinformatics* **26**, 960 (2010).
13. W. Gerlach, S. Jünemann, F. Tille, A. Goesmann and J. Stoye, *BMC Bioinformatics* **10**, p. 430 (2009).
14. M. N. Price, P. S. Dehal and A. P. Arkin, *Mol. Biol. Evol.* **26**, 1641 (2009).
15. D. G. Brown and J. Truszkowski, Towards a practical $O(n \log n)$ phylogeny algorithm, in *Proceedings of WABI*, (Saarbrücken, Germany, 2011).
16. A. C. McHardy, H. G. Martn, A. Tsirigos, P. Hugenholtz and I. Rigoutsos1, *Nature Methods* **4**, 63 (2007).
17. N. N. Diaz, L. Krause, A. Goesmann, K. Niehaus and T. W. Nattkemper, *BMC Bioinformatics* **10** (2009).
18. D. R. Kelley and S. L. Salzberg, *BMC Bioinformatics* **11**, p. 544 (2010).
19. P. Meinicke, K. P. Aßhauer and T. Lingner, *Bioinformatics* **27**, 1618 (2011).
20. S. F. Altschul, T. L. Madden, A. A. Schffer, J. Zhang, Z. Zhang *et al.*, *Nucleic Acids Research* **25**, 3389 (1997).
21. D. H. Huson, S. Mitra, N. Weber, H.-J. Ruscheweyh and S. C. Schuster, *Genome Research* **21**, 1552 (2011).
22. M. M. Haque, T. S. Ghosh, D. Komanduri and S. S. Mande, *Bioinformatics* **25**, 1722 (2009).
23. J. Buhler and M. Tompa, *J. Comp. Biol.* **9**, 225 (2002).
24. W. Evans, C. Kenyon, Y. Peres and L. J. Schulman, *The Annals of Applied Probability* **10**, 410 (2000).
25. S. Roch, *Science* **327**, 1376 (2010).
26. O. Gascuel and M. Steel, *Mathematical Biosciences* **227**, 125 (2010).
27. M. A. Steel, Distributions on bicoloured evolutionary trees. (1989), PhD thesis. Massey University.
28. M. Sultan, M. H. Schulz, H. Richard, A. Magen, A. Klingenhoff *et al.*, *Science* **5891**, 956 (2008).
29. J. Felsenstein, *Inferring Phylogenies* (Sinauer, 2001).
30. A. Stamatakis, Phylogenetic models of rate heterogeneity: a high performance computing perspective, in *IPDPS*, (IEEE, 2006).
31. A. K. Bartram, M. D. J. Lynch, J. C. Stearns, G. Moreno-Hagelsieb, and J. D. Neufeld, *Applied and Environmental Microbiology* **77**, 3846 (2011).
32. E. P. Nawrocki, D. L. Kolbe and S. R. Eddy, *Bioinformatics* **25**, p. 1713 (2009).
33. P. Yarza, M. Richter, J. Peplies, J. Euzeby and R. A. others, *Systematic and Applied Microbiology* **31**, 241 (2008).