Advanced Software Design for StatAlign

Hrafn Eiríksson, Viktor Jonsson, David Wood

September 13, 2011

Abstract

We present the work we undertook during a six week summer school from 11th July to 19th August at the University of Oxford. Improvements were made to the phylogenetic analysis program StatAlign as part of the progress towards a new release. We implemented both consensus tree and consensus network calculating and drawing algorithms in addition to parallelising the main Markov Chain Monte Carlo calculations using a $(MC)^3$ technique. Improvements to the GUI were also made.

1 Introduction

StatAlign is a java-based software package simultaneously performing sequence alignment and phylogeny reconstruction. It was first released in 2008 [8]. Begining from an inital aligment by score-based methods and then improving this with Markov Chain Monte Carlo (MCMC) methods it samples the joint posterior distribution of alignments after a user specified burn-in time. A range of substitution models is availible to the user and the TKF92 model [14] is used for insertions and deletions to calculate the MCMC probabilities.

Several similar software packages for phylogeny reconstruction exist such as MrBayes [9], BaliPhy [11] etc. What sets StatAlign apart from these is joint sampling of tree and alignment space (as opposed to fixing the alignment before sampling from tree space). The usage of a fixed alignment can bias the reconstructed phylogeny [13]. In addition to this, StatAlign aims to allow the user to continuously monitor the sampling process through an easy-to-use GUI.

The main goal of this project was to improve the usability of StatAlign through the incorporation of consensus tree and network visualisation. This would give users a powerful tool to monitor the progress of the MCMC. A major disadvantage of StatAlign compared to other software is that it is single threaded, limiting the performance for users with access to multi core computers or clusters. Therefore the secondary goal was to parallelise StatAlign.

2 Theory

2.1 Consensus Trees

One of the phylogenetic analysis tools that we wanted to equip StatAlign with was a so-called *consensus tree* analyser. A consensus tree is generated from a sample of input trees and summarises most or all of the features that the input trees agree on into a single output tree. Our aim was to make StatAlign generate consensus trees in *real-time* during the sampling process.

There exist some well established algorithms to construct consensus trees. In their paper Amenta et al. [2] describe a linear-time algorithm to do this. Sul and Tiffani further extended this algorithm in their paper [12] to a point where it could easily handle large-scale tree collections with many taxa in a short time. Both algorithms base their approach on representing phylogenies in terms of their *bipartitions*. Removing an arbitrary edge e from a tree splits the tree into two sets of leaves: a bipartition. If e is an internal edge (an edge connecting two internal vertices) the bipartition is considered *non-trivial* as opposed to a trivial bipartition (where e connects a internal vertex to a leaf). Both algorithms go through each of the trees in the sample and determine their non-trivial bipartitions. They then use a memory-efficient hash mechanism to store the bipartitions and the frequency of which they occur in the sample of trees. This information is then used to construct a consensus tree which can either be a *majority consensus tree* (a tree containing *most* of the bipartitions of the sample trees) or a *strict consensus tree* (a tree which only contains features that occur in *all* of the sample trees).

Evidently we could base our approach on the

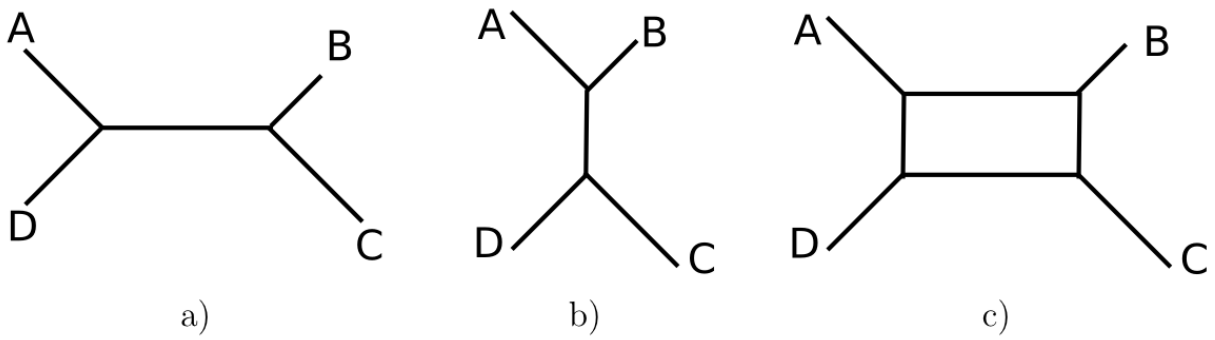


Figure 1: The bipartitons a) CD—AB and b) BC—AD shown are incompatible as by the definition in the text. If both occur with highfrequency, we can represent ambiguity between them in a box form as in c).

one described above, but this would not be optimal since we wanted our program to build the consensus trees *continuously* as we received the tree samples. Keeping track of the old samples and running the whole algorithm over and over again seemed redundant. Instead we adjusted the algorithm to use the same hash table throughout the sampling process. This way the determination and hashing of the bipartitions only needed to be done once for each sample tree. The part of the algorithm that actually constructed the consensus tree was implemented mostly unchanged and was triggered for each tree sample. This seemed to be a good approach as the majority of the time is spent on determining and hashing the bipartitions. At this point our algorithm was only generating the topology of the consensus trees–i.e. without any branch lengths. How eaxctly to determine the branch lengths of consensus trees is a subject of an ongoing debate and we decided to settle on a simple mean averaging method.

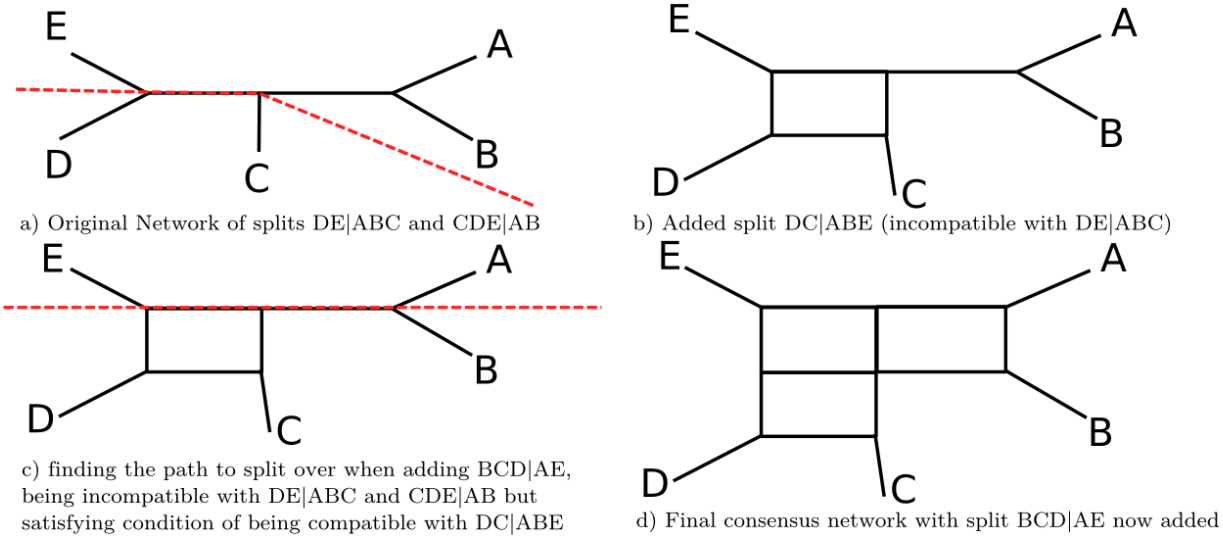
2.2 Consensus Networks

Phylogenetic tree samples frequently show highly ambiguous regions, where two or more bipartitons occur with high frequencies but are incompatible with one another. The program defines two bipartitons as being *incompatible* if neither side of one of the bipartitons cannot be written as a subset of one of the sides in the other bipartiton. As shown in figure 1, a single edge cannot represent two incompatible bipartitons. They can be shown instead as a box where contracting over one of the dimensions returns one of the incompatible trees.

The consensus network is an extension of the consensus tree and indeed we begin creating the network from this tree. The network approach allows us to lower the minimum threshold that we imposed on the frequency of the occurrence of bipartitons to include them in the consensus tree. The new value can be represented by p_{min}, the minimum proportion of trees in the sample in which the bipartiton must occur. We restrict the value p_{min} using the formula in equation 1:

$$p_{min} > \frac{1}{1+d} \tag{1}$$

where d is the maximum number of dimensions that we can use to draw the network produced from any sample of trees [4]. To clarify the meaning of *dimension*, a zero dimensional network has no non-trivial bipartitons and is a basic star-shaped network, a one dimensional structure is a standard tree with only lines and a two dimensional network includes boxes and can be drawn on a page with boxes but without lines needing to cross over. A three dimensional network would include cubes to demonstrate three-way incompatibility. Equation 1 is explained simply: if we have a sample of trees we know that no individual tree in that sample can contain two bipartitons that are incompatible with each other. So if we only include bipartitons that occur in the majority of trees in the sample then we will have only bipartitons that are all compatible with one another (as any two bipartitons must occur in at least one of the sample trees). Extending this idea, say a bipartiton occurs in at least a third of each of the sample trees. The remaining less than two-thirds of the sample trees can then only contain at most one bipartiton or set of compatible bipartitons that occurs in over a third of the trees that is incompatible with the initial one. Hence, only considering bipartitons occurring in over a third of the trees in the sample, each bipartiton can only be incompatible with at most one other bipartiton. Two incompatible bipartitons can be represented by a square as shown in figure 1. Extending to three dimensions, and a



a) Original Network of splits DE|ABC and CDE|AB

b) Added split DC|ABE (incompatible with DE|ABC)

c) finding the path to split over when adding BCD|AE, being incompatible with DE|ABC and CDE|AB but satisfying condition of being compatible with DC|ABE

d) Final consensus network with split BCD|AE now added

Figure 2: Bipartitions are added along the dotted red lines into the networks by finding paths of incompatible bipartitons as shown. Note that in c) there are edges that represent incompatible bipartitons that are not included in the final path.

limit of a quarter, we can represent three incompatible bipartitons as a cube.

The program stores the network as a collection of bipartitons (with lists of member edges) and nodes (which can be either taxa or internal) but there is no longer a defined root. The first step is to simply convert the tree representation to this form and the non-majority tree bipartitons are then added one by one. For each bipartiton we find all of the bipartitons in the set that it is incompatible with. If there are no incompatible bipartitons, the edges on a node will need to be split between two nodes that are linked by an edge of the new bipartiton. Finding this node requires beginning from a taxon on one side of the bipartiton we want to add and moving recursively through all joining nodes until reaching a node with an edge joining it that represents a bipartiton that no longer occurs on the same side as the starting taxon in the bipartiton we want to add. If we have several incompatible bipartitons then we need to find the edges to copy, which can seem complicated when we already have several boxes with incompatible bipartitons being represented by more than one edge. As shown in the figure, there should be a path along which to split the whole network into two. To find this the program finds all paths that can be made by joining together edges of incompatible bipartitons without using the same bipartiton more than once in each path. The longest path of these that has end points with edges that represent bipartitons that create subsets on both sides of the bipartiton we are adding should represent all incompatible bipartitons in a single, unique longest path. All nodes and edges along this path are copied to create a copied path with new edges between corresponding old and new nodes (see figure 2). Each path then represents one side of the added bipartiton and the edges joining the old path are redistributed between the two paths depending on which side of the added bipartiton their bipartiton occurs on. In higher dimensional graphs, there would be more than one longest path satisfying the above conditions, all the nodes and edges of which would need copying.

The program extends the equal angle method proposed by Meacham [4] for drawing trees to draw networks. The tree drawing method simply begins at a node in the tree with a full revolution of angle space assigned to it (360 degrees) from which directions of adjoining edges can take. This angle space then divided up among adjoining edges as they are added so that the percentage of angle space assigned to an adjoining node is proportional to the number of taxa that need to be drawn from it. The edges to each node are given the direction and thus drawn along the centre of the assigned angle space. In networks, boxes cause complications that the algorithm treats. See Appendix A for more details. StatAlign produces both a network tree view and zoomable consensus network as part of its GUI.

2.3 Parallelisation of MCMC

A major issue with MCMC is the convergence rate of the sampling process. This can be of high concern when the posterior space is large and contains

several regions of high probability separated by regions of low probability. In such cases the samples needed to reach convergence can grow unmanageably large. The questions of convergence and mixing in the chain are not easily answered [6]. A step towards alleviating this problem is the introduction of Metropolis coupled MCMC (or $(MC)^3$) [5]. This extension to regular MCMC adds heated chains that run in parallel to the main cold chain. Heat in this case is a parameter in the range [0 1] and is used to adjust the acceptance probability as seen in Equation 2 as defined in [1].

$$R_i = \min\left[1, \left(\frac{f(X \mid \psi_i')}{f(X \mid \psi_i)} \frac{f(\psi_i')}{f(\psi_i)}\right)^{\beta_i} \frac{q(\psi_i)}{q(\psi_i')}\right] \quad (2)$$

Here ψ_i denotes the current state of the Markov chain i and ψ_i' represents the proposed state. $f(X \mid \psi_i')$ is the likelihood of the new state and $f(\psi_i')$ is the prior probability of the new state. $q(\psi_i)$ and $q(\psi_i')$ represent the backward and forward proposal probabilities, respectively. β is the heat of chain i. The new state, ψ_i', is accepted with a probability of R_i. The ratio can be compared by the regular acceptance ratio used in MCMC and seen in Equation 3.

$$R_i = \min\left[1, \frac{f(X \mid \psi_i')}{f(X \mid \psi_i)} \frac{f(\psi_i')}{f(\psi_i)} \frac{q(\psi_i)}{q(\psi_i')}\right] \quad (3)$$

By taking the power of a number smaller than one the heated chains sample from flattened distributions which allows the chains to move across the sample space more rapidly. Samples are not taken from the heated chains as these represent a skewed distribution. The introduction of swaps between chains can still improve the final results as this can facilitate longer jumps in posterior space for the cold chain improving mixing. The acceptance ratio for swaps between two chains k and j is defined as follows,

$$R = \min\left[1, \frac{f(\psi_k \mid X)^{\beta_j} f(\psi_j \mid X)^{\beta_k}}{f(\psi_j \mid X)^{\beta_j} f(\psi_k \mid X)^{\beta_k}}\right]. \quad (4)$$

The $(MC)^3$ method was first introduced as a way to improve mixing but it also allows for a straightforward way of parallelising the MCMC process [1]. A single markov chain is inherently difficult to parallelise but when several chains are used they can be easily distributed among cores and/or clusters. The benefits, however, of running heated chains and introducing swaps compared to just running several cold chains in parallel and sampling from all is debated [3]. A heated chain is essentially using CPU time to generate a lot of samples that are just thrown away. Running several cold chains in parallel multiplies the amount of samples generated and allows for estimations of mixing by comparing the sets of samples generated by the different chains. MrBayes [9] implements $(MC)^3$ while BAli-Phy [11] implements parallel MCMC.

There are two main paradigms when writing parallel programs: Message Passing Interface (MPI) and shared memory. In the latter all processes share the same memory and care has to be taken to avoid race conditions and similar issues. In the former, processes have their own allocated memory and communication is facilitated through send and receive buffers. As the name shared memory indicates, no special methods are needed to communicate among processes as they can directly access the relevant data. MPI adds extra overheads to facilitate communication but gains in thread safety and is also easier to implement on clusters. Shared memory can be implemented on clusters but it requires more sophisticated methods such as distributed shared memory.

We examined several different Java frameworks for both approaches. We quickly ruled out the shared memory approach since we would need to ship distributed shared memory software with our program, which would in turn need installation and so forth. After careful consideration, MPJ Express [10] was chosen with the reasoning that it uses the well developed MPI standard and that it is an improvement over the former mpiJava. It has also been shown to offer good performance both on single machines and clusters.

3 Results

3.1 Performance of MCMC vs. $(MC)^3$

Heated chains were implemented as suggested in [1] but during testing they showed very poor performance even at low temperatures. As the temperature is increased, fewer state proposals for alignment and topology are accepted. This seems counter-intuitive as heated chains are designed to traverse the state space more rapidly. The cause for this is the state proposal probability mechanism. Consider a cold chain presently in a state with low likelihood. We now use an intelligent method to generate a new state with higher likelihood. Due to how the proposal method is designed the probability of proposing the new good state is higher than proposing a change from the new state

to the current poor one. The last term in Equation 3 becomes small, lowering the chance to accept this new state. However since the state has higher posterior probability the other terms make up for this and the state is likely to be accepted. For a heated chain we reduce the influence of the posterior probability. This essentially means that if we propose two good states we have very little chance of accepting them. We have succeeded in making it easier for the chain to move towards lower probability but made it harder for it to move to higher probability. To alleviate this problem we attempted to flatten out the proposal distributions as well. The first attempt at doing this was to rescale the transition matrices for the HMMs that are used to propose new alignments. However this had a minor effect on acceptance rates. One issue was that the flattening of the transition probabilities might be too coarse and we ended up with an HMM proposing many poor alignments.

Even though acceptance rates drop for the heated chains it is still plausible that they transverse the posterior space more rapidly by making bigger although fewer steps. The performance of the heated chains can also be evaluated by comparing the alignments generated with that of the cold chain as well as the log likelihood of the samples generated. The alignment comparison was done by first generating a template "true" alignment with a cold chain and then comparing the individual alignments at each sample with this true alignment. A properly adjusted heated chain would be expected to have similar top scoring alignments as the cold chain but with larger variation between samples and a lower average similarity. The results did not show this. The alignments produced by heated chains had lower accuracy on average but also lower rates of change due to the drop in acceptance rate for alignment changes. For some data sets the hot chains also exhibited diverging alignments where they would keep adding all gap columns to the alignments. An example of this behaviour can be seen in Figure 3.

A heated chain would be expected to reach similar log likelihood levels as the cold chain but exhibiting larger dips in likelihood. A simple comparison of log likelihood levels for a set of chains with different heating on two different data sets can be seen in Figure 3. In Figure 3(a) the heated chains have a significantly lower, but stable, likelihood. All chains implemented the rescaling of the transition matrices for the proposal HMMs. This was done by taking each element in the matrix to the power of the heat parameter and then re-normalising them to sum to one. One of the chains used the square root of the heat parameter for matrix rescaling but the effect in the results was minor. Figure 3(b) shows an example of the diverging alignments. The likelihood for all chains drops low but at different rates depending on the heat level. Even for very long runs the cold chain never diverged.

3.2 Consensus Trees and Networks

The network drawing algorithm worked well under tests as shown in figure 4. This part of the program seemed to perform reasonbly fast under tests and output networks were checked with those from other network drawing software such as SplitsTree 4 [7] which draws networks based on bipartitions.

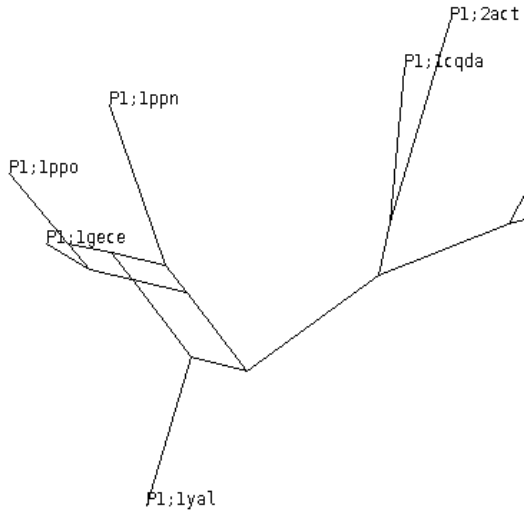
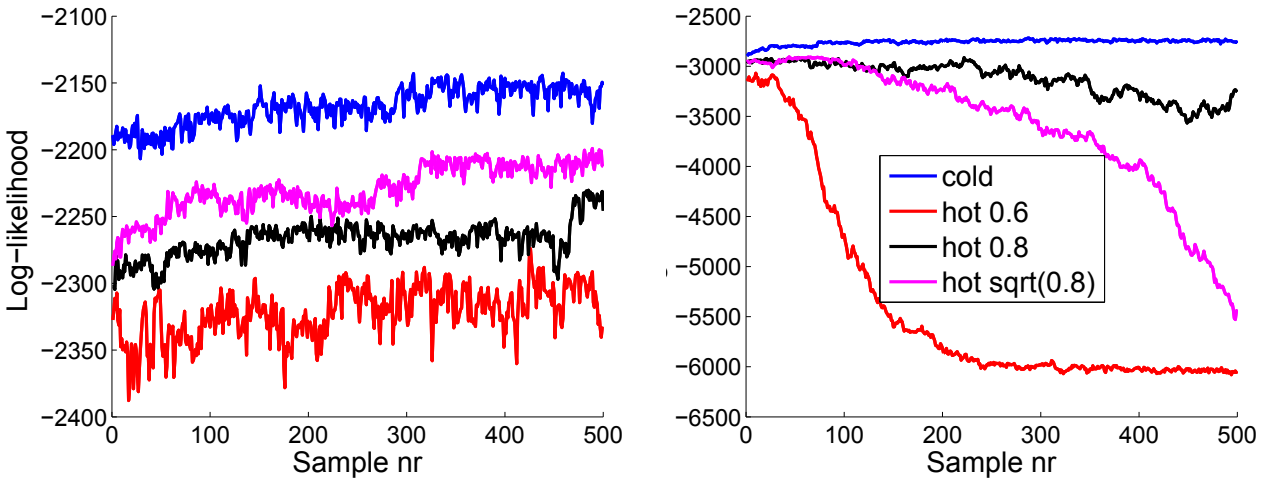


Figure 4: Part of a consensus network as drawn by StatAlign during testsing, showing multiple boxes.

4 Discussion

Many adjustments have been made to StatAlign since its last version. Not only have major features been added as we have described above, but also quite a few bugs have been eliminated. In our opinion there are, however, a few things that need to be fixed before a new version of StatAlign will be released.

- The introduction of parallelism through Metropolis coupled MCMC (or $(MC)^3$) relies heavily on the soundness of the underlying MCMC framework, making the implementation vulnerable to flaws in the framework. As of this point our $(MC)^3$ implementation seems to suffer from this kind of a flaw that is making itself noticed by causing alignments to diverge. It is hard to tell whether this is



(a) (b)

Figure 3: Two examples of the performance of heated chains in the current implementation. (a) shows an well behaving data set where the heated chains have lower but stable likelihoods. (b) is a more difficult data set which when used together with heated chains exhibits a rare bug that cause the alignments to diverge.

easy to fix or not as the bugs can be hidden deep inside the MCMC framework. Regardless, for the $(MC)^3$ to function properly a good way of flattening proposal densities is needed in order to tune the performance of the heated chains.

- Another problem with the current parallelisation implementation is that it does not adhere to StatAlign's philosophy of allowing users to monitor and visualise the progress of the sampling process. This rises from the fact that the current implementation can only be run through a command-line interface, due to the nature of the MPJ express framework.

- The consensus network visualiser should be tested further with some real data and compared with other approaches such as that of SplitsTree. Weightings should also be added and the user could be allowed to extend the threshold (p_{min}) down to the minimum amount before 3 dimensons are required.

- The algorithm has been written in a way that should be compatible with creating higher dimensional networks. The conjecture that if more than one longest path connecting incompatible bipartions is possible when adding a bipartition requires drawing in more dimensions could be explored. Ultimately a 3D version of the drawing algortihm could be written, perhaps simply using a multiple 2D level set up.

5 Acknowledgements

We would like to thank supervisors Adam Novák, Rune Lyngsø and Jotun Hein as well as the lecturers and other students on the summer school that provided a stimulating atmosphere in which to spend the summer.

References

[1] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinformatics*, 20(3):407–415, 2004.

[2] N. Amenta, F. Clarke, and K. S. John. A linear-time majority tree algorithm. In *In: Proc. 3rd Workshop Algs. in Bioinformatics (WABI03*, pages 216–227. SpringerVerlag, 2003.

[3] R. G. Beiko, J. M. Keith, T. J. Harlow, and M. A. Ragan. Searching for convergence in phylogenetic markov chain monte carlo. *Systematic Biology*, 55(4):553–565, 2006.

[4] Joseph Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc, 2004.

[5] C. J. Geyer. Markov chain Monte Carlo maximum likelihood. In Keramidas, editor, *Computing Science and Statistics of the 23rd Symposium on the Interface*, pages 156–163. Interface foundation, Fairfax Station, 1991.

[6] J. P. Huelsenbeck, F. Ronquist, R. Nielsen, and J. P. Bollback. Bayesian inference of phylogeny and its impact on evolutionary biology. *Science*, 294(5550):2310–2314, December 2001.

[7] Daniel H. Huson and David Bryant. Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, 23(2):254–267, February 2006.

[8] Á. Novák, I. Miklós, R. Lyngsø, and J. Hein. Statalign: an extendable software package for joint bayesian estimation of alignments and evolutionary trees. *Bioinformatics*, 24(20):2403–2404, 2008.

[9] F. Ronquist and J. P. Huelsenbeck. MrBayes 3: Bayesian phylogenetic inference under mixed models. *Bioinformatics*, 19(12):1572–1574, August 2003.

[10] A. Shafi, B. Carpenter, and M. Baker. Nested parallelism for multi-core hpc systems using java. *J. Parallel Distrib. Comput.*, 69(6):532–545, 2009.

[11] M. A. Suchard and B. D. Redelings. Bali-phy: simultaneous bayesian inference of alignment and phylogeny. *Bioinformatics*, 22(16):2047–2048, 2006.

[12] S. Sul and T. L. Williams. An experimental analysis of consensus tree algorithms for large-scale tree collections. In *Proceedings of the 5th International Symposium on Bioinformatics Research and Applications*, ISBRA '09, pages 100–111, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] J. L. Thorne and H. Kishino. Freeing phylogenies from artifacts of alignment. *Molecular Biology and Evolution*, 9(6):1148–1162, 1992.

[14] J.L. Thorne, H. Kishino, and J. Felsenstein. An evolutionary model for maximum likelihood alignment of dna sequences. *J Mol Evol*, 34:91, 1992.

A Consensus Network Drawing

In networks, including boxes adds restrictions to the equal angle algorithm that make it much more complicated. Firstly, there is no longer freedom of order of edges as they are assigned angle space on a node. If a node is the corner of a box then this constrains certain edges to be neighbours and so on a node the program builds up lists of neighbouring edges and adds any of these undrawn edges in order of the list. As these edges form boxes, the program has a maximum angle of just below half a revolution between two assigned directions of neighbouring edges. Secondly, each bipartiton has a direction assigned to it when the first edge representing it is drawn, and all subsequent edges of this bipartiton must be drawn in this direction so as to ensure that boxes form joined up. It is important to note, however, that edges with to draw with pre-specified directions at a node still need to receive angle space for their subsequent nodes but their directions will not be within the assignable angle space for this node as this was assigned to another node. Therefore they receive angle space that is closest to their direction, but only half of what would be expected from counting the taxa on the bipartiton side to avoid over-counting from the other edge that closes this box. When drawing the program needs to make sure we assign direction to edges as we move away from the first node we pick, always drawing edges of a bipartiton from the same side of that bipartiton and so the program draws nodes outwards, running through all that are a certain number of nodes away from the starting node before moving to those further degrees away.

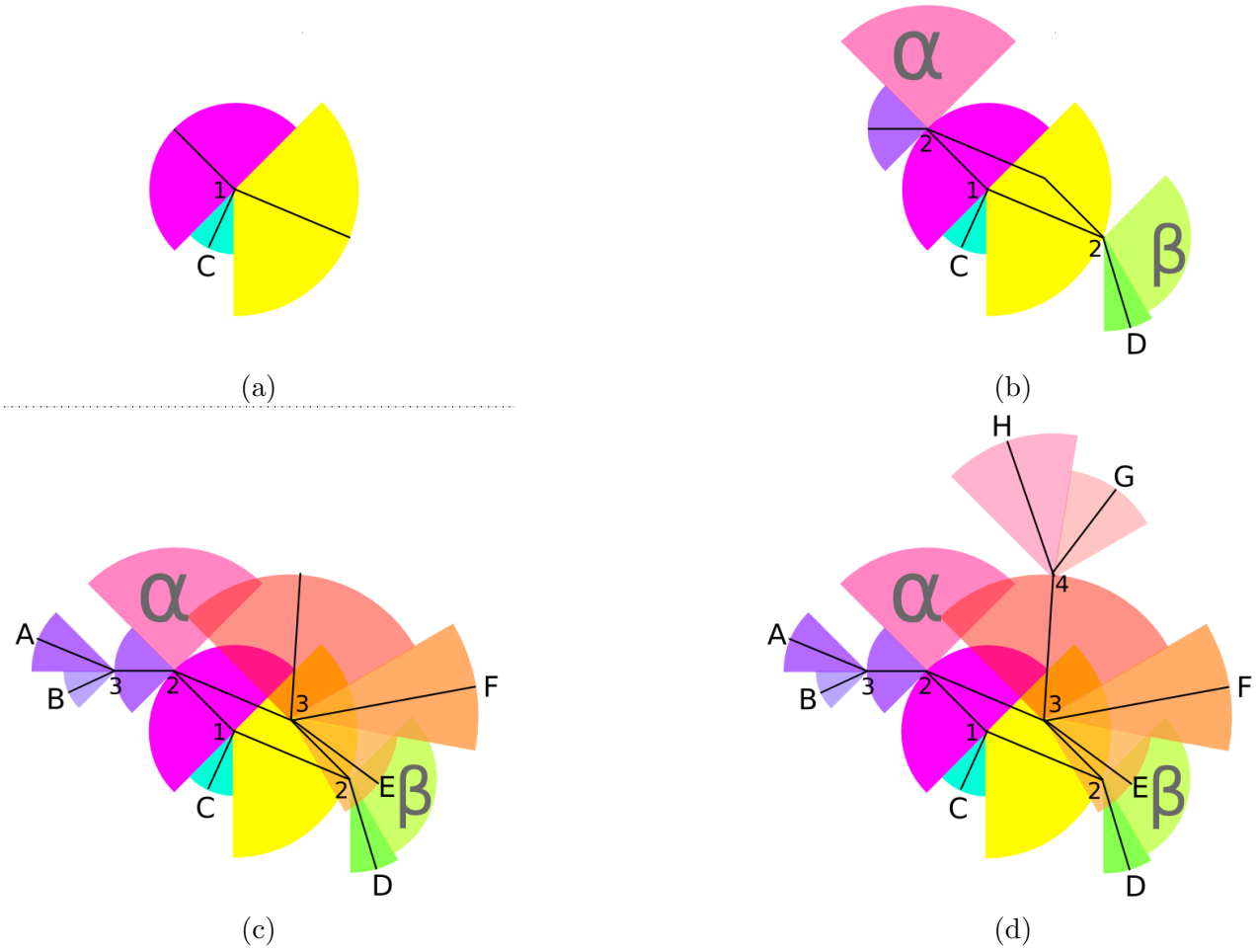


Figure 5: The network drawing algorithm progresses from a) to d), beginning from the node labeled 1 and cycles through nodes with increasing numeric labels as they are drawn in place by the algroithm described in the text. The coloured cones represent the divisions of angle space as the algrotihm progresses outwards. The edges are drawn at the centre of the assigned angle space cones, with the exception of regions α and β which are assigned angle space for edges that must take directions that are pre-defined as they belong to a bipartiton which has already had an edge drawn.