

Statistical Alignment via k -Restricted Steiner Trees

Rune Lyngsø

Background

When relating a set of sequences by a phylogeny, we are essentially constructing a Steiner tree connecting the sequences in the space of all finite sequences. Finding an optimal Steiner tree is in most formulations hard, so population genetics and phylogenetics have often used spanning trees as an approximation for computational expediency. In this assessment you will be asked to investigate an intermediate between spanning trees and Steiner trees.

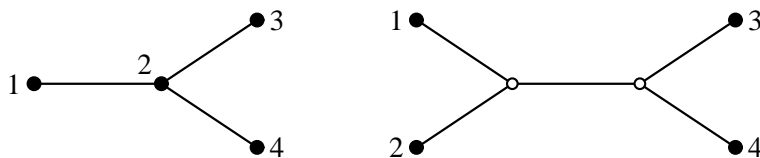


Figure 1: A spanning tree and a Steiner tree on the nodes $\{1, 2, 3, 4\}$. Terminal nodes (the nodes that need to be connected) are shown as solid circles, while Steiner nodes (auxiliary nodes) are shown with hollow circles. There are n^{n-2} possible spanning trees on n labelled nodes. If labelled nodes have degree 1 and unlabelled nodes have degree 3, there are $(2n - 5) \cdot (2n - 7) \cdot \dots \cdot 1 = \frac{(2n-5)!}{2^{n-3}(n-3)!}$ Steiner tree topologies on n labelled nodes.

A spanning tree on a set of labelled nodes connects the nodes by a tree where each edge connects two of the labelled nodes. So all nodes in the tree are labelled. An example of a spanning tree is shown in Fig. 1. A Steiner tree on a set of labelled nodes also connect the nodes by a tree, but may introduce further auxiliary internal nodes. Hence, some or all of the internal nodes in the tree may be unlabelled. An example of a Steiner tree is shown in Fig. 1.

Constructing a multiple alignment for a given phylogeny corresponds to determining the ‘location’ of the Steiner (*i.e.* unlabelled) nodes in finite sequence space. If the phylogeny is inferred along with the alignment we are solving a variant of the

Steiner tree problem. The multiple alignment problem is hard when the number of sequences to align is unbounded. So for a large number of sequences we may consider to compute all pairwise alignments, and use the pairwise distances between the nodes to approximate the optimal phylogeny by a spanning tree. This approximation may be rather crude. If we can align more than two sequences but not all the input sequences, we would expect better approximations to be obtained if we reconstruct the phylogeny from small sets of multiply aligned sequences rather than just pairs of aligned sequences.

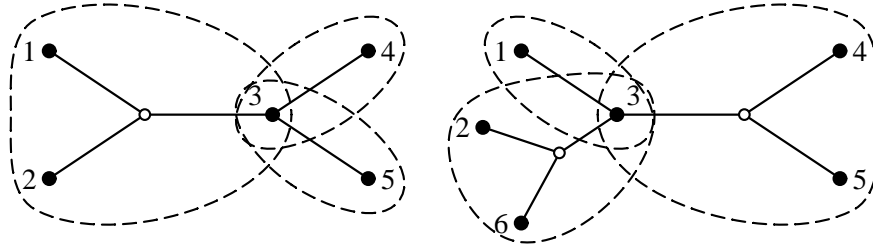


Figure 2: Two examples of 3-restricted Steiner trees. If the node terminal node labelled 3 had been a Steiner node, the left tree would have been a 4-restricted Steiner tree and the right tree would have been a 5-restricted Steiner tree, and both trees would have been unrestricted Steiner trees. The dashed lines encloses the full components of the trees.

This general approach of stitching together a good approximation of the overall Steiner tree from Steiner trees on subsets of the terminal nodes is in general the approach taken by Steiner tree approximation algorithms ([1] mentions only one exception). A Steiner tree on a subset of nodes where all internal nodes are Steiner nodes is called a full component. This concept is illustrated in Fig. 2. A Steiner tree where the maximum number of terminal nodes in any full component is at most k is called a k -restricted Steiner tree. The full components of a Steiner tree is a set of subtrees spanning the terminal nodes. The subtrees are independent except for shared terminal nodes, and connects together to the full Steiner tree through these shared terminal nodes.

In alignment terms, a full component corresponds to an alignment of a subset of the input sequences. In statistical alignment, the evolution in the subtrees rooted at an internal node is independent conditional on the state, i.e. sequence, of the node. Usually this is of little assistance as we need to condition on an infinite number of possible states for internal nodes. But when internal nodes corresponds to observed sequences, there is only one state to consider. So the statistical alignment probability conditional on a given phylogeny topology corresponding to a k -restricted Steiner tree can be obtained by combining the statistical alignment

probabilities obtained for each full component conditional on the subtopology for that component. More generally, any Steiner tree method based on optimal Steiner trees for small subsets of nodes translates into a statistical alignment method based on statistical alignment of small subsets of the sequences.

Project

As already mentioned, most Steiner tree approximation algorithms with guaranteed performance are based on updating a partial solution with new full components. The algorithm with the best known approximation ratio [1] is outlined below.

Algorithm 1 Loss-Contracting Algorithm (k -LCA) for Steiner tree approximation

Input: A set of terminal nodes S and optimal Steiner trees on all subsets of S of size at most k

T = minimum spanning tree on S

H = complete graph on S

while there is a k -restricted full component K with $\text{gain}_T(K) > 0$ **do**

Find k -restricted full component K with maximal $\text{gain}_K(T) / \text{loss}(K)$

$H = H \cup K$

$T =$ minimum spanning tree on $T \cup K_{\text{loss-contracted}}$

Output the minimum spanning tree on H

There are a few elements of this algorithm that needs further explaining, namely the availability of full components, the gain_T and lossfunctions and $K_{\text{loss-contracted}}$. The k -restricted full components of interest are simply the full components observed in the Steiner trees on subsets of S of size k that is part of the input.

The $\text{gain}_T(K)$ function is the gain in score that can be obtained by adding the Steiner nodes and edges of K to T , i.e. the cost of T minus the cost of the minimum terminal node spanning tree on the union of K and T . This is illustrated in Fig. 3.

Apart from the obvious gain of updating a partial solution with a full component, there is also a potential loss. If the Steiner nodes of the full component do not coincide with the Steiner nodes of the optimal Steiner tree, we will incur an extra cost of connecting these to the rest of the solution. This is captured by the lossfunction, where $\text{loss}(K)$ is the minimum cost of connecting the Steiner nodes of K to the terminal nodes of K . That is, $\text{loss}(K)$ is the cost of a minimum cost forest on K where each tree in the forest is required to contain a terminal node. This is illustrated in Fig. 4. The algorithm greedily adds the full component maximising the ration between gain and loss until there are no full components resulting in a further gain.

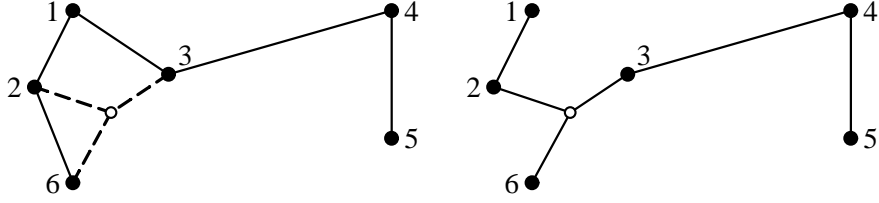


Figure 3: The left hand illustration shows a minimum spanning tree T (solid edges and nodes) on the terminal nodes in the right hand graph of Fig. 2 and the full component K on the nodes 2, 3, and 6 (dashed edges and hollow node). The minimum terminal node spanning tree on the union of T and K is shown to the right; the gain of adding K to T ($\text{gain}_T(K)$) is the cost of T minus the cost of this tree.

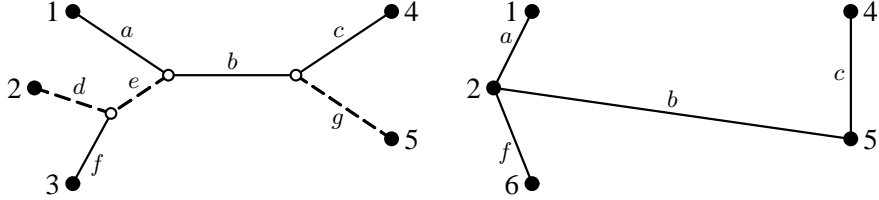


Figure 4: The left hand illustration shows a full component K on five terminal nodes with the loss edges shown with dashed lines. There are two Steiner nodes in the connected component of terminal node 2 and one Steiner node in the connected component of terminal node 5. With the annotated edge weights $\text{loss}(K) = d + e + g$. The right hand illustration shows $K_{\text{loss-contracted}}$ with the edge weights inherited from the non-loss edges of K .

The main improvement of the algorithm results from not fixing an entire full component that is chosen, but only fixing the loss edges. So when contracting a full component K it is not contracted into a single node. Instead, the nodes of each connected component, i.e. the loss edges, are contracted. The resulting $K_{\text{loss-contracted}}$ is a spanning tree on the terminal nodes of K with two nodes connected if the two connected components they were the terminal nodes in were directly connected by an edge. This is illustrated in Fig. 4. The cost of the loss edges, $\text{loss}(K)$, was offset against the gain, $\text{gain}_T(K)$, when the full component was chosen. Consequently, we only need to maintain the costs of the non-loss edges in $K_{\text{loss-contracted}}$ – if some of the edges of $K_{\text{loss-contracted}}$ appear in the final solution, adding the loss edges and Steiner nodes removed by the contracting costs at most $\text{loss}(K)$. This

is also illustrated in Fig. 4.

k	2	4	8	16	32	64	128	256	512	1024
ratio	1.88	1.80	1.74	1.71	1.68	1.67	1.65	1.64	1.63	1.63

Table 1: Dependence of approximation ratio on k .

The approximation ratio of k -LCA is $\rho_k(1 + \frac{1}{2} \ln(\frac{4}{\rho_k} - 1))$ where $\rho_k \leq 1 + \frac{1}{1 + \lceil \log_2 k \rceil}$ is the worst-case ratio of the cost of the optimal k -restricted Steiner tree and the optimal Steiner tree (the numbering may be off by one – 2-restricted Steiner trees are spanning trees that are only guaranteed to cost at most twice as much as Steiner trees, not $1 + \frac{1}{1 + \lceil \log_2 2 \rceil} = 1.5$ times as much). This clearly converges to $1 + \frac{\ln 3}{2} \approx 1.55$ for $k \rightarrow \infty$. However, it is easily clear that the convergence is rather slow, with asymptotic growth of $O(\log \log k / \log k)$. The ratios computed for the first ten relevant powers of 2 are shown in Tab. 1.

As each full component in the alignment case requires a multiple alignment computed, it is probably unrealistic assuming k to be more than four and five in cases with many sequences. With m sequences the brute force approach would be to compute all multiple alignments¹ of k sequences, i.e. $\binom{m}{k}$ alignments. Multiply this by the time it takes to align k sequences using statistical alignment and it is evident that k has to be set to well less than infinity. However, the approximation ratio is a worst-case performance, and in real cases we may see a much better approximation even for small values of k .

A reasonable outline of the project is as follows.

- Implement and test brute force approach where all k -restricted full components, i.e. alignments of subsets of k sequences, are enumerated. This should address the problem of interfacing between the statistical alignment program and the Steiner tree approximation program. Furthermore, without too much work it should allow us to test how well the approximation works on the largest possible data sets that can be handled exactly before proceeding.
- Determine rules for when a full component can be ignored a priori. This may be difficult, for one thing because we would have to establish that the gain of all remaining full components are zero to safely terminate the algorithm.

¹We will actually need the phylogeny including branch probabilities rather than any alignment, but for brevity will refer to the input from the statistical alignment subprocess as alignments; once the Steiner tree approximation has been computed, it should be relatively easy to stitch a global multiple alignment together from e.g. the most probable local alignments of the sequences in the full components

- Develop and test heuristics not guaranteed to preserve the approximation guarantee. One obvious thing to explore is a good stopping criteria. Given the incremental nature of k -LCA, where a partial solution is continuously improved until no further improvement is possible, stopping prematurely will not render the work done so far useless. For every iteration the partial solution is guaranteed to improve.

However, it should be remembered that computing all k -restricted full components is likely to be the dominant term in the time used for a full execution. So if this is done already in the first iteration there is probably very little to gain from a stopping criteria. In light of this, a good heuristic for estimating $\text{gain}_T(K) / \text{loss}(K)$ without actually computing K would be tremendously helpful.

The outline above is based on the assumption that we want a certain quality of the solution produced. In many cases it will be more realistic to assume a certain amount of time is available. Again the incremental nature of the approximation algorithm makes it easy to prematurely terminate the iteration and report the solution constructed so far. However, in cases where a time limit is specified it would probably be a good idea for the implementation to decide the appropriate value of k , compromising between the increased power of larger k versus the increase in running time, rather than require the user to specify this.

The k -LCA start from a spanning tree and adds Steiner nodes to improve the solution. An alternative would be to start from a Steiner tree topology and iteratively contract edges to terminal nodes (thereby bringing them in as internal nodes) until the resulting tree is k -restricted. Bioinformatics has a long tradition for inferring phylogenies from data. Starting from e.g. a neighbour-joining tree this would provide a very fast method of obtaining a k -restricted Steiner tree, assuming the edges to contract are determined reasonably efficiently. Just choosing the shortest branch ending at a terminal node would probably not work too well, as we may perform numerous superfluous edge contractions in sequence clusters before getting to the edge contractions making the part of the tree describing old speciations k -restricted. A somewhat slower approach would be for every edge incident to a terminal node to compute the neighbour-joining trees of the new subsets of sequences corresponding to full components that are obtained when contracting the edge, and then choose the edge that increases the overall cost the least.

A disadvantage of the last method proposed, i.e. contracting edges in a neighbour-joining tree, is that the topology to a large extent is fixed by the original neighbour-joining tree. Contrary to the k -LCA this means that the method will be quite poor for phylogeny reconstruction, and would probably mostly be useful for data probability approximation and constructing a global alignment from local most prob-

able alignments. However, it is likely to be much faster, so one could imagine a two-stage method, where the neighbour-joining based algorithm is used initially to estimate optimal evolutionary parameters. Once the evolutionary parameters have been fixed, a k -LCA based method is then applied once to reestimate phylogeny.

References

- [1] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proceedings of the 11th Annual Symposium on Discrete Algorithms (SODA)*, 2000.