Weekly Report (RFS Implementation)

Reza Manshouri
Department of Computer Science and Computer Engineering
Texas A&M University

January 13, 2017

Abstract

Recently, we found a new technique, Edge Ratchet, which seems to have the potential to improve upon RF Supertree (RFS) algorithm. Our initial implementation of the algorithm, was able to improve upon RFS on small data sets. However, on larger data sets, our algorithm was not able to finish in 48 hours time limit. Thus, we decided to implement RFS and use it as a subroutine in our algorithm to improve both running time and accuracy of our algorithm. In this document, I will present Robinson Foulds Supertree algorithm implementation. Results of the experiments on its effectiveness on both accuracy and running time will be presented.

1 Introduction

Robinson Foulds Supertree (RFS) is one of the successful supertree algorithms for supertree construction which yields relatively high quality supertrees. Recently, we found a new technique, Edge Ratchet (ER), with which we were able to improve upon RFS in some data sets. Our ER-RFS algorithm performs a pre-specified number of ratchet iterations. Each ratchet iteration, consists of two hill-climbing search using SPR as edit operation. The first hill-climbing search is done on weighted data where we randomly select a subset of edges in source trees and re-weight them. The objective is to minimize weighted RF distance. In the second hill-climbing search of each ratchet iteration, we use un-weighted data and try to minimize RF distance.

Each of these searches continue until we reach a local optimum. Then, the local optimum is used as initial supertree for the next hill climbing search.

In our initial implementation, we used an exhaustive search for local search in SPR neighbourhood. Since the exhaustive search is very expensive and not practical for data sets with even 100 taxa, we used random neighbourhood selection. In this approach, in each iteration of hill-climbing search, we randomly chose a small percentage of the neighbourhood, say 1%, and searched through it exhaustively to find the best neighbour. However, this approach has two main drawbacks. Firs, even with this approach, two complete ratchet iterations took more than 48 hours for Marsupials data set with 267 taxa. Although we could have reduced the neighbourhood percentage to even less than 1%, then we would not have much confidence on the solution accuracy. Second, random selection of a small portion of neighbourhood is highly prone to result in low quality solutions since we simply ignore a huge percentage of neighbours in each iteration. Because of these issues we were not able to further investigate Edge Ratchet technique and test its effectiveness (to improve over RFS) in larger data sets.

Thus, we decided to implement RFS and use it as a subroutine in our hill-climbing algorithm so that we can test ER's effectiveness to improve supertree accuracy. The RFS algorithm can be directly used in the second phase, hill-climbing search on un-weighted data, of each ratchet iteration. For the second phase, we expect using RFS algorithm results in much faster algorithm ($O(N)$ improvement $O(N^3)$ naive algorithm). Further, we expect the accuracy to be improved as well. This is because in RFS we find the best neighbour in each local search (in contrast to our algorithm which does not guarantee this).

We might be able to apply RFS algorithm to the first phase, weighted data, as well. However, I think it is not trivial. I will investigate this in the following week.

2 Algorithm

The heart of RFS algorithm is the fast local search algorithm to solve SPR problem. In this problem, we are given a supertree and some source trees, and the question is to find the SPR neighbour of the supertree which has the lowest (cumulative) RF distance to source trees. In the naive way, we need $O(N^3)$ time to find the best SPR neighbour of the current supertree: we

have $O(N^2)$ SPR neighbours, and we need $O(N)$ for calculating RF distance. Their contribution is to provide a $O(N^2)$ algorithm for this problem. Their actual algorithm, however, solves Restricted-SPR problem. In this problem, in addition to supertree and sourc tree(s), we are given which node in the supertree to be pruned. The question is to find the best regraft location for which RF distance is minimized. They present a $O(N)$ algorithm for this problem, which directly yields a $O(N^2)$ algorithm for SPR problem.

Thus the algorithm essentially is for Restricted-SPR problem. The algorithm achieves this by not actually calculating RF distance for each neighbour. Instead, the algorithm, in $O(N)$, keeps track of the bipartitions in the source trees that exists in current tree but not in the neighbour, and vice versa. This helps us find the best place to regraft without actually calculating RF distance for each possible place.

3 Implementation

As it was mentioned above, the goal here is to implement a $O(N)$ algorithm for the Restricted

The algorithm was developed on the SPR supertree algorithm's source code, i.e. I used their data structure and basic methods. However, I had to modify both data structure and some methods. The algorithm was implemented exactly the same way as it is described in the paper except one part. The only major difference is the implementation of an algorithm to find LCA of a given set of nodes in a tree. They used a complicated $O(N)$ algorithm, but we used our own $O(N)$ algorithm as it is described in the following paragraph.

Every node in the tree has new int field called lca-count. The tree is being traversed in post-order manner. When reaching a leaf, I check if it is among the leaf-set of interest. If it was, I will increase its parent's lca-count by one. When traversing an internal node, firs I check if its lca-count equals the size of leaf-set of interest. If it was, we have found the lca. If it's not, I increase its parent's lca-count by its own lca-count.

After I implemented the algorithm, I performed many tests to make sure everything works properly. First I performed some unit-tests to make sure each method 1- works as it is expected to work, and 2- does not produce unwanted effects (on data structure or variables). Then, I put print statements throughout the algorithm execution to make sure all variables are updated

correctly and the final values are not flawed. Several bugs showed up during my tests, and all of the are resolved now.

Most parts of the algorithm was straight forward to implement. However, there are several parts in which further improvement is possible (either implementation wise or algorithmic wise). I have made a list of those places and will work on them. Further, there is a memory leak in my program which I need to resolve before putting it into RF-RFS. There is another suspicious observation. Further, although the algorithm was able to fins a SPR neighbour with lowest RF distance, the number of possible SPR neighbours for the algorithm was not equal to that of old algorithm (exhaustive search). I will find resolve this issue tomorrow.

4 Results

Finally I was able to run the algorithm, and use it to solve Restricted-SPR problem. To test the algorithm, I modified the old algorithm so that it solves Restricted-SPR problem as well. For the sake of initial test, I used only one source tree which has the same set of taxa as supertree (These are the assumptions they make in the paper. But it is straight forward to generalize the algorithm for a more general case where we have multiple source trees with different number of taxa).

Thus I gave both algorithms one supertree, one source tree, and one node in supertree to be pruned. The new algorithm works correctly in the sense that the SPR neighbour it finds has the same RF distance as that of old algorithm (i.e. exhaustively try all regraft places, calculate its RF distance, and keep track of supertree with lowest RF distance). However, the neighbours are not necessarily the same (they just have the same RF distance to source tree).

Further, as I expected, the algorithm is much faster as well. For example, in case the supertree has 7 taxa, the old algorithm took 0.002942 sec, but the new algorithm took 0.000398 sec. For when supertree has 30 taxa, the new algorithm took 0.001825 sec, while the old one took 0.02814 sec. It seems that, even without further improvement, we have at least order of magnitude improvement over old algorithm. After I perform more tests, we will have a better feeling on how much it actually improves over the old one.

I plan to resolve the issues I mentioned above by the end of this weekend. There is a couple of more tests I also want to perform before using the

algorithm in our ER-RFS algorithm by the end of this weekend. Then, I will put this implementation of RFS in our ER-RFS algorithm (by Tuesday), and I immediately start to run experiments with it and compare the results with what we had before.